Animation of Deformable Bodies with Quadratic Bézier Finite Elements

Adam W. Bargteil and Elaine Cohen
University of Utah

In this article, we investigate the use of quadratic finite elements for graphical animation of deformable bodies. We consider both integrating quadratic elements with conventional linear elements to achieve a computationally efficient adaptive-degree simulation framework as well as wholly quadratic elements for the simulation of nonlinear rest shapes. In both cases, we adopt the Bézier basis functions and employ a co-rotational linear strain formulation. As with linear elements, the co-rotational formulation allows us to precompute per-element stiffness matrices, resulting in substantial computational savings. We present several examples that demonstrate the advantages of quadratic elements in general and our adaptive-degree system in particular. Furthermore, we demonstrate, for the first time in computer graphics, animations of volumetric deformable bodies with nonlinear rest shapes.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Animation*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Finite-element methods, adaptive simulation, deformable bodies, natural phenomena, physics-based animation

1. INTRODUCTION

Over the last decade there has been an explosion in the use of finite element methods for computer animation of deformable bodies. These methods have been used to animate numerous special effects in films and are commonly used to animate deformable bodies in video games. Computer graphics has generally favored linear simulation elements, both for defining geometry as well as field variables, due to their simplicity and computational efficiency. On the contrary, outside of computer graphics higher-order methods are often preferred, in part for improved accuracy and convergence properties. While in computer graphics we are not necessarily concerned with traditional notions of accuracy or convergence, high-order elements do offer several important advantages over linear elements. In particular, high-order elements allow us to animate nonlinear geometry and offer a natural way to add degrees of freedom to a simulation mesh, allowing low-resolution meshes to provide animation quality of much higher resolution.

This work was supported in part by gifts from Disney Interactive Research and Adobe Systems Incorporated and National Science Foundation awards CNS-0855167, IIS-1249756, IIS-1117997, and IIS-1314896.
Authors addresses: A. W. Bargteil (corresponding author) and E. Cohen, Department of Computer Science, University of Utah, UT; email: adamb@cs.utah.edu.

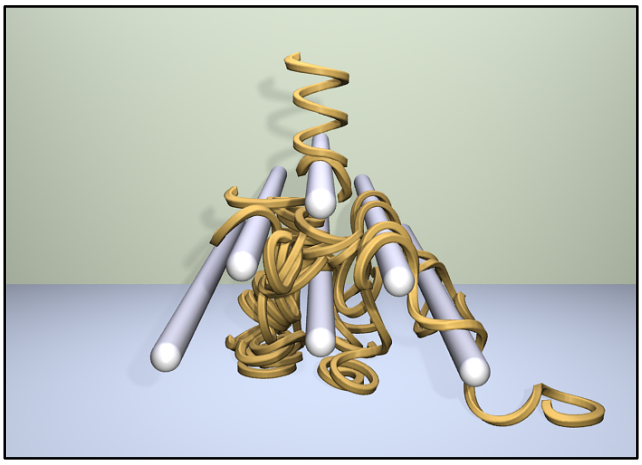

Fig. 1: *Helices with quadratic rest shapes dropped onto some obstacles.*

In this article, we develop a quadratic tetrahedral element based on the Bézier basis. We then integrate this element with linear elements to achieve an adaptive simulation system where elements undergoing large deformation are higher order than elements undergoing small deformations. By adopting a co-rotational linear strain formulation that allows us to precompute per-element stiffness matrices and speeds force computations, we avoid online numerical quadrature and achieve performance that is orders of magnitude faster than more general quadratic elements. We also adapt the co-rotational formulation to elements with nonlinear rest shapes (see Figure 1).

The result is a simulation system that can animate nonlinear geometry and, for a modest overhead compared to linear finite-element simulation, can locally increase degrees of freedom without resorting to remeshing strategies.

2. RELATED WORK

Since the pioneering work of O'Brien and Hodgins [1999], tetrahedral finite-element methods have been an extremely popular approach for animating deformable bodies. Müller and colleagues [2002; 2004] introduced a co-rotated finite-element formulation that admits the use of linear strain metrics without incurring the usual distortions under large deformations. This approach is particularly appealing in a computer graphics context because with linear strain the stiffness matrix becomes constant and can be precomputed. For this reason, we adopt this formulation. Building on this approach, Irving and colleagues [2004] developed an extremely robust technique that handles degenerate and inverted elements. Parker and O'Brien [2009] used the co-rotational formulation for simulations of deformation and fracture in a real-time videogame

UU IR Author Manuscript

environment. More recently, researchers have pointed out that traditional implementations of the co-rotational model ignore rotational derivatives when computing the stiffness matrix [Chao et al. 2010] and that even with these derivatives there may still be stability issues [Stomakhin et al. 2012].

We are not the first computer graphics researchers to use high-order elements in simulation. Roth and colleagues [1998] used the Bézier basis for tetrahedral finite-element simulation more than a decade ago. They avoided numerical integration by integrating forces analytically, as we do for linear rest shapes. However, they were concerned with modeling the effects of plastic surgery and sought equilibrium solutions, ignoring dynamics. More recently, Mezger and colleagues [2009] used quadratic elastoplastic co-rotational finite-element simulations for shape editing. Weber and colleagues [2011] also used quadratic Bézier co-rotational elements for interactive simulation of deformable models and created a GPU implementation [Weber et al. 2013]. None of these approaches considered adaptive-degree elements or quadratic rest states.

Remion and colleagues [1999] performed high-order simulations of one-dimensional objects using splines. Similarly, Kaldor and colleagues [2008] used cubic splines to simulate knitted cloth. Grinspun and colleagues [2002] performed several simulations with high-order basis functions. Kaufmann and colleagues [2009] used non-nodal quadratic polynomial basis functions defined over arbitrary polyhedra in a discontinuous Galerkin method. A number of other researchers have looked at using arbitrary polyhedral elements. Such elements require more complex basis functions. Wicke and colleagues [2007] considered basis functions derived from mean-value coordinates [Ju et al. 2005] and Martin and colleagues [2008] used basis functions derived from harmonic coordinates [Joshi et al. 2007]. In a similar vein Kaufmann and colleagues [2009] present an Extended Finite-Element Method (X-FEM) approach to modeling high-resolution fractures of shells by updating basis functions stored as textures. Bickel and colleagues [2009] used radial basis functions to model heterogeneous, nonlinear materials. The radial basis functions were fit to captured deformations that were assumed to be samples of a locally linear stress-strain relationship.

Of course, high-order analysis is routine in other fields of engineering and numerical analysis. Our finite-element approach is an instance of the more general *hp-FEM* method pioneered by Babuška and colleagues (see, e.g., Babuška and Suri [1990]) in the field of solid mechanics. This method admits both hierarchical (h), namely geometric division of elements, and polynomial (p), namely increasing polynomial degree, refinements. Our approach, then, is an instance of p-refinement [Babuška et al. 1981]. Working in fluid mechanics, Orszag [1969] pioneered spectral methods which also take advantage of high-order bases. Usually globally supported functions are used, though the spectral-element method [Patera 1984], which combines finite-element and spectral techniques, uses a locally supported high-order piecewise polynomial basis. The text by Hughes [1987] briefly covers high-order tetrahedral finite elements as well as the use of transition elements that allow for the integration of high- and low-order elements in quadrilateral meshes. When our elements contain both linear and quadratic edges, they may be thought of as such transition elements. More recently, there has been interest in *isogeometric* analysis, which seeks to unify geometric representations with finite-element basis functions [Borden et al. 2011].

Debunne and colleagues [2001] introduced spatial adaptivity in computer graphics by using a non-nested multiresolution hierarchy of tetrahedral meshes. Another approach to adaptivity was proposed by Grinspun and colleagues [2002]. Instead of using geometric refinement, creating smaller tetrahedra in areas of interest, they refine basis functions. This approach is somewhat similar in spirit to our approach, however, instead of adding basis functions with increasingly limited areas of influence, we increase the degree of our basis functions. An advantage of their approach is that it can allow for arbitrary differences in refinement over the mesh, while we are limited to just two different degrees of polynomials. If greater resolution is necessary their approach could be used to complement ours.

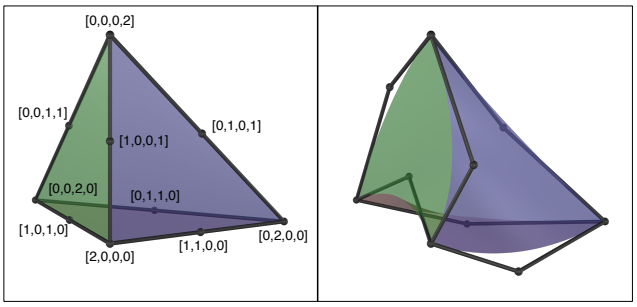

Fig. 2: *The left image shows the* canonical tetrahedron*, with its quadratic control points labeled by their multi-indices. The right image shows the quadratic deformation induced by the deformed control mesh.*

3. METHODS

In this section we first derive the quadratic Bézier element and then present how we integrate these elements with linear elements to create an adaptive simulation system.

3.1 Barycentric Coordinates and Bernstein Polynomials

Barycentric coordinates allow us to express quantities inside a simplex in terms of values at the vertices. For a point inside a tetrahedron, the barycentric coordinates, $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \alpha_2, \alpha_3]$, have the properties that $\alpha_j \geq 0$ and $\sum \alpha_j = 1$. The second property implies that the barycentric coordinates are not independent. We remove the redundancy by considering $\alpha_0 = 1-\alpha_1-\alpha_2-\alpha_3$ as a function of the other barycentric coordinates. One way to view barycentric coordinates is as the position of a point $[\alpha_1, \alpha_2, \alpha_3]$ in a canonical tetrahedron with $\boldsymbol{v}_0 = [0,0,0]$, $\boldsymbol{v}_1 = [1,0,0]$, $\boldsymbol{v}_2 = [0,1,0]$, and $\boldsymbol{v}_3 = [0,0,1]$ (see Figure 2). In the following section we will define mappings of this canonical tetrahedron to both world and material space. Note that because $\boldsymbol{v}_0$ lies at the origin, $\boldsymbol{v}_j - \boldsymbol{v}_0 = \boldsymbol{v}_j$. We take advantage of this slight abuse of notation in the formulas that follow.

We generalize the usual linear basis function over each element by using the Bézier basis which is given by the Bernstein polynomials.

$$B^n_{\boldsymbol{i}}(\boldsymbol{\alpha}) = \frac{n!}{i_0!i_1!i_2!i_3!}\alpha_0^{i_0}\alpha_1^{i_1}\alpha_2^{i_2}\alpha_3^{i_3} \tag{1}$$

Here, n is the degree of the polynomial, the point $\boldsymbol{\alpha}$ is represented in barycentric coordinates, and $\boldsymbol{i} = [i_0, i_1, i_2, i_3]$ is a *multi-index* such that $|\boldsymbol{i}| = \sum i_j = n$ and all $i_j >= 0$.

We chose the Bézier basis for several reasons. First, the Bézier basis is very intuitive to work with. Because the control points are

UU IR Author Manuscript

naturally associated with the vertices and edges of the tetrahedron it is easy to understand connectivity among tetrahedra and the effect of raising and lowering the order on the control points as well as handling adjoining tetrahedra of different degrees. Second, the basis functions are all nonnegative. Third, because the basis functions sum to one and are nonnegative, the Bézier basis has the *convex hull property*. Though we do not take advantage of it in our current implementation, this property is especially useful for accurate collision detection. Finally, the Bézier basis is very familiar in the computer graphics community. The Bézier basis does have one drawback compared to other bases: they are not *hierarchical* in the sense that the set of functions that span the space of quadratic polynomials do not include the functions used to span linear polynomials. This fact means that as we change the order of our elements, we do not simply add basis functions; we move to a different set of functions.

3.2 Quadratic Elements

We now generalize the standard formulation of elasticity for linear finite elements used in computer graphics (see, for example, O'Brien and Hodgins [1999]) to Bézier tetrahedra.

We begin by defining a mapping from points, $\boldsymbol{u}$, in *material* space to points, $\boldsymbol{x}(\boldsymbol{u})$, in *deformed* or *world* space. Let $\boldsymbol{m_i}$ be the positions of control points in material space and $\boldsymbol{p_i}$ be the positions of control points in world space. Then, given the (four-dimensional) barycentric coordinates, $\boldsymbol{\alpha}$, of a point in a Bézier tetrahedron,

$$\boldsymbol{u}(\boldsymbol{\alpha}) = \sum_{|\boldsymbol{a}|=n} \boldsymbol{m_a} B^n_{\boldsymbol{a}}(\boldsymbol{\alpha}) \qquad \boldsymbol{x}(\boldsymbol{\alpha}) = \sum_{|\boldsymbol{a}|=n} \boldsymbol{p_a} B^n_{\boldsymbol{a}}(\boldsymbol{\alpha}). \quad (2)$$

$\boldsymbol{x}(\boldsymbol{u})$ is then given by

$$\boldsymbol{x}(\boldsymbol{u}) = \left(\sum_{|\boldsymbol{a}|=n} \boldsymbol{p_a} B^n_{\boldsymbol{a}}\right) \circ \left(\sum_{|\boldsymbol{b}|=n} \boldsymbol{m_b} B^n_{\boldsymbol{b}}\right)^{-1} (\boldsymbol{u}). \quad (3)$$

Unfortunately, this mapping involves the inverse of a polynomial function, for which there is no closed form. For now we restrict ourselves to the simpler case of linear rest shapes, where all quantities can be integrated analytically, and develop our adaptive simulation approach. We consider nonlinear rest shapes in Section 3.7.

A linear mapping from the canonical tetrahedron to material coordinates can be expressed as

$$\boldsymbol{u}(\boldsymbol{\alpha}) = \boldsymbol{m}_0 + \sum_{i=1}^{3} (\boldsymbol{m}_i - \boldsymbol{m}_0)\alpha_i. \quad (4)$$

The gradient of this mapping can be represented as a matrix, the columns of which are $(\boldsymbol{m}_i - \boldsymbol{m}_0)$. Let $\boldsymbol{\beta}$ be the inverse of this matrix.

3.2.1 *Deformation Gradient.* The mapping $\boldsymbol{x}(\boldsymbol{u})$ is often referred to as the deformation function. The gradient of this function is called the deformation gradient and is denoted as $\boldsymbol{F}$. In $\mathcal{R}^3$, $\boldsymbol{F}$ is a 3×3 matrix, with the ij-entry given by

$$F_{ij} = \frac{\partial x_i}{\partial u_j} = \sum_{k=1}^{3} \frac{\partial x_i}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial u_j} = \sum_{k=1}^{3} \frac{\partial x_i}{\partial \alpha_k} \beta_{kj}, \quad (5)$$

where, invoking the Bézier basis,

$$\frac{\partial x_i}{\partial \alpha_k} = \sum_{|\boldsymbol{a}|=n} p_{\boldsymbol{a}i} \frac{\partial B^n_{\boldsymbol{a}}(\boldsymbol{\alpha})}{\partial \alpha_k} = \sum_{|\boldsymbol{a}|=n} n p_{\boldsymbol{a}i} \left(B^{n-1}_{\boldsymbol{a}-[k]}(\boldsymbol{\alpha}) - B^{n-1}_{\boldsymbol{a}-[0]}(\boldsymbol{\alpha}) \right). \quad (6)$$

Note, we are using $[k]$ to denote the multi-index with the kth element set to one (for example, $[2] = [0, 0, 1, 0]$), n is the polynomial degree, $p_{\boldsymbol{a}i}$ is the ith coordinate of the point $\boldsymbol{p_a}$. Subtraction of multi-indices is analogous to vector subtraction, except that any multi-index containing a -1 yields a basis function that is identically zero. Some substitution yields

$$F_{ij} = \sum_{k=1}^{3} \left(\sum_{|\boldsymbol{a}|=n} n p_{\boldsymbol{a}i} \left(B^{n-1}_{\boldsymbol{a}-[k]}(\boldsymbol{\alpha}) - B^{n-1}_{\boldsymbol{a}-[0]}(\boldsymbol{\alpha}) \right) \right) \beta_{kj}. \quad (7)$$

Note that $\boldsymbol{F}$ is spatially varying.

3.2.2 *Stress.* For its computational efficiency we use the popular rotated linear stress model [Müller and Gross 2004; Irving et al. 2004; Parker and O'Brien 2009]. To do so we compute the polar decomposition $\boldsymbol{F} = \boldsymbol{Q}\tilde{\boldsymbol{F}}$ and then compute the co-rotational strain, $\tilde{\boldsymbol{\epsilon}}$, and stress, $\boldsymbol{\sigma}$ as

$$\tilde{\boldsymbol{\epsilon}} = \frac{1}{2}\left(\tilde{\boldsymbol{F}} + \tilde{\boldsymbol{F}}^T\right) - \boldsymbol{I} \qquad \boldsymbol{\sigma} = \lambda \mathrm{Tr}(\tilde{\boldsymbol{\epsilon}})\boldsymbol{I} + 2\mu\tilde{\boldsymbol{\epsilon}}, \quad (8)$$

where λ and μ are Lamé constants of the material. Note that while $\tilde{\boldsymbol{F}} = \tilde{\boldsymbol{F}}^T$, this form for $\tilde{\boldsymbol{\epsilon}}$ is required when we take the gradient of the elastic energy to arrive at forces (see Equation (10)). Because $\boldsymbol{F}$ is spatially varying, the best-fitting rotation varies as well. However, we assume that the variation is small and compute the polar decomposition using just the control points at the vertices of the tetrahedron. This approach gives the same rotation, regardless of whether the tetrahedron is linear or quadratic, and works well in practice.

3.2.3 *Elastic Forces.* For linear, constant-strain elements, elastic forces induced by an element upon a control point can be computed by integrating the product of stress and the gradient of the basis functions over the element [Bonet and Wood 2008]. Because of the relationship between cross products and matrix inversion in three dimensions, this amounts to multiplying the stress by area-weighted normals [Teran et al. 2003]. With our quadratic elements it is slightly more complicated. In our case, the c-dimensional component of the force on node $\boldsymbol{a}$ is given by

$$\tilde{f}_{\boldsymbol{a}c} = 6v \int_E \sum_{i,j} \sigma_{ci} \frac{\partial B^2_{\boldsymbol{a}}}{\partial \alpha_j} \beta_{ji} d\boldsymbol{\alpha}, \quad (9)$$

where v is the volume of the element in material space. Some substitution yields

$$\begin{aligned}\tilde{f}_{\boldsymbol{ac}} = 6\lambda v \sum_{\boldsymbol{b},i,j,k} \tilde{x}_{\boldsymbol{bi}}\beta_{ji}\beta_{kc}\int_E \frac{\partial B^2_{\mathbf{b}}}{\partial\alpha_j}\frac{\partial B^2_{\mathbf{a}}}{\partial\alpha_k}d\boldsymbol{\alpha} \\ + 6\mu v \sum_{\boldsymbol{b},i,j,k} \tilde{x}_{\mathbf{bc}}\beta_{ji}\beta_{ki}\int_E \frac{\partial B^2_{\mathbf{b}}}{\partial\alpha_j}\frac{\partial B^2_{\mathbf{a}}}{\partial\alpha_k}d\boldsymbol{\alpha} \\ + 6\mu v \sum_{\boldsymbol{b},i,j,k} \tilde{x}_{\mathbf{b}i}\beta_{jc}\beta_{ki}\int_E \frac{\partial B^2_{\mathbf{b}}}{\partial\alpha_j}\frac{\partial B^2_{\mathbf{a}}}{\partial\alpha_k}d\boldsymbol{\alpha} \\ - (18\lambda + 12\mu)\, v \sum_i \beta_{ic}\int_E \frac{\partial B^2_{\mathbf{a}}}{\partial\alpha_i}d\boldsymbol{\alpha}, \end{aligned} \tag{10}$$

where

$$\begin{aligned}&\int_E \frac{\partial B^2_{\mathbf{b}}}{\partial\alpha_j}\frac{\partial B^2_{\mathbf{a}}}{\partial\alpha_k}d\boldsymbol{\alpha} = \\ &\int_E 4\left(B^1_{\boldsymbol{b}-[j]}(\boldsymbol{\alpha}) - B^1_{\boldsymbol{b}-[0]}(\boldsymbol{\alpha})\right)\left(B^1_{\boldsymbol{a}-[k]}(\boldsymbol{\alpha}) - B^1_{\boldsymbol{a}-[0]}(\boldsymbol{\alpha})\right) = \\ &4\int_E B^1_{\boldsymbol{b}-[j]}(\boldsymbol{\alpha})B^1_{\boldsymbol{a}-[k]}(\boldsymbol{\alpha}) - 4\int_E B^1_{\boldsymbol{b}-[0]}(\boldsymbol{\alpha})B^1_{\boldsymbol{a}-[k]}(\boldsymbol{\alpha}) - \\ &4\int_E B^1_{\boldsymbol{b}-[j]}(\boldsymbol{\alpha})B^1_{\boldsymbol{a}-[0]}(\boldsymbol{\alpha}) + 4\int_E B^1_{\boldsymbol{b}-[0]}(\boldsymbol{\alpha})B^1_{\boldsymbol{a}-[0]}(\boldsymbol{\alpha}). \end{aligned} \tag{11}$$

Each of these last four integrals evaluate to 0 if either of the multi-indices contain a -1, $1/120$ if they contain two different multi-indices, and $1/60$ if the multi-indices are the same.

There are two principal differences between Equation (10) and that for the linear case. First, a quadratic tetrahedron has ten control points, rather than just four. Second, with a linear tetrahedron the integrals of the products of basis functions evaluate to 0 or $\pm 1/6$, where 1/6 is the volume of the canonical tetrahedron. In the quadratic case the integrals are the sum of four terms that are either 0, $\pm 1/15$, or $\pm 1/30$. The rest of the terms are the same as in linear elements. The last term in Equation (10) is a "force offset" that accounts for the fact that forces are based on displacements not absolute positions. Müller and Gross [2004] had a similar term.

Forces computed with Equation (10) are in the rotated space. To rotate them to world space, we multiply by $\boldsymbol{Q}$. The damping forces are similarly computed, though there are no force offsets and the control point velocities are used instead of positions. The stiffness and damping matrices are symmetric,[1] positive semi-definite, and are straightforward to compute by taking the gradient of Equation (10), which is linear in positions.

3.3 Precomputation

Our implementation is able to use several explicit and implicit integrators. Implicit integration requires building a system matrix that includes terms from the stiffness and/or damping matrices. Fortunately, because we use a rotated linear strain model, the stiffness and damping matrices for each element are constant through all time, up to a rotation. Thus, at the beginning of the simulation, we precompute each element's linear and quadratic stiffness and damping matrices. By requiring that the material parameters for damping have the same ratio as λ and μ, a model known as Rayleigh damping, the element stiffness and damping matrices are identical up to a constant, and only one need be stored. While we have not found this restriction to be problematic in practice, more general damping models could be employed with increased storage. The global stiffness and damping matrices incorporate different rotations, however, and cannot be precomputed. The stiffness matrix for a linear element has only 78 unique entries. However, the stiffness matrix for a quadratic element is a dense 30×30 symmetric matrix and contains 465 unique entries. We also precompute the "force offsets" for each element (another 30 numbers) and store the $\boldsymbol{\beta}$ matrices and area-weighted normals (for force computations in linear elements) for a total of 594 numbers per element. This equates to more than 4.5 kilobytes of storage per element. This may seem a large memory footprint, but when one considers that dividing a linear tetrahedron at edge centers results in 8 tetrahedra, which yields 624 unique matrix entries and 72 force offsets, it becomes clear that quadratic tetrahedra require less memory per degree of freedom than linear tetrahedra. Furthermore, on modern computers the memory requirements are negligible—even our largest experiments required less than 150MB of memory. However, streaming these matrices when computing forces does result in a significant number of cache misses.

3.4 Adaptivity

In a typical simulation there are, on average, a large number of elements that are very near their rest configuration. If the rest shape is linear, performing the added computations associated with quadratic elements for these tetrahedra is not an efficient use of computational resources. For this reason, our system incorporates both linear and quadratic elements. To do so, we must treat edges that are incident to both linear and quadratic tetrahedra specially.

From the point of view of quadratic tetrahedra, there is a degree of freedom along such edges, but from the point of view of the linear tetrahedra there is not. We handle this case by constraining the position and velocity of a control point along such an edge to be the average of the values at the endpoints. Any force that the quadratic tetrahedron would exert on the control point is distributed to the endpoints, half to each. Additionally, such control points are removed as degrees of freedom from any implicit solve. Any values they would have added to the global system matrix are redistributed to the endpoints. In the case that two edges in a quadratic tetrahedron are both constrained, the entries for the interactions between them are divided among the four endpoints, 1/4 to each. It is possible that one of the endpoints is shared by both edges, in which case it gets 1/2 of the value.

When all of the elements incident to an edge become quadratic, the control point associated with the edge is added as a degree of freedom. The mass of the control point is determined using a lumped mass formulation as $1/10$ the mass of the tetrahedron and the masses at the endpoints are reduced by $1/20$ the mass of the tetrahedron. The new control point's velocity is initialized to the average of the endpoints' velocities. Consequently, momentum is preserved during refinement. Adopting a full mass matrix will ensure that kinetic energy is also preserved, but would also increase complexity and runtimes.

When deciding which elements should be linear and which should be quadratic, it is convenient to take an edge-centric point of view and decide what degree each edge should be. A variety of rules could be used to determine the degree of an edge. We have adopted the following, simple approach. Each tetrahedron does the full

[1] Symmetry can be verified by manipulating the gradient of Equation (10).

quadratic elastic and damping force calculations. Then, the position of the edge midpoint is predicted assuming both linear and quadratic states and the results are compared. If the difference in the predicted position is larger than a threshold, the edge becomes quadratic. If the difference is smaller than another (smaller) threshold then the edge is linear. We additionally require edges to maintain the same state for a minimum number of timesteps to avoid oscillations (5 in our examples). This approach does require that we multiply by a 30×30 matrix instead of a 12×12 matrix for each element and represents the most significant difference in cost between linear and quadratic elements.

A consequence of adaptive degree elements is that the nonzero structure of the global system matrix is constantly changing. Our implementation allocates memory to store the entire stiffness matrix with all quadratic tetrahedra at the beginning of the simulation. We store the matrix in 3×3 blocks and only the lower triangular portion of the matrix is stored. When an edge becomes quadratic the control point along that edge becomes active and its row in the matrix is enabled. Additionally, all the other control points that the newly enabled point interacts with have another block turned on in their row. As mentioned before, stiffness values for inactive control points are distributed to edge endpoints. In addition to adding degrees of freedom to the stiffness matrix, we remove degrees of freedom every timestep as well. When removing degrees of freedom we must increase the mass at the endpoints of the edge and adjust their velocities to preserve momentum.

In this way, we have opted for a *reduced coordinates* formulation to handle constraints. Another option is Lagrange multipliers, which would allow the particularly elegant approach of choosing the degree of edges based on the magnitude of their associated Lagrange multipliers. In fact, this approach would open the door to characterizing the constrained solution as a Linear-Complimentarity Problem (LCP) with a bound on the magnitude of the allowed Lagrange multiplier. However, our system has variable size no larger than $|\mathcal{V}| + |\mathcal{E}|$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. On the other hand, the system augmented with Lagrange multipliers has size $|\mathcal{V}|+2|\mathcal{E}|$ characterized as an LCP or, if the constraints $\mathcal{C}$ are constant throughout the solve, $|\mathcal{V}| + |\mathcal{E}| + |\mathcal{C}|$. Given that $|\mathcal{V}|$ is substantially smaller than $|\mathcal{E}|$, we felt the additional cost was prohibitive.

3.5 Time Integration

We have experimented with a number of explicit and implicit time integration schemes, specifically, symplectic forward Euler, two variations of linearly implicit Euler (one that computes velocity updates as proposed by Baraff and Witkin [1998] and another that computes new velocities directly), the variational integrator of Kharevych and colleagues [2006], and a mixed implicit-explicit Newmark scheme [Bridson et al. 2003]. Of these, we favor the linearly implicit Euler that solves for new velocities directly. Specifically, the system we solve is

$$\left(\boldsymbol{M} - \Delta t^2 \boldsymbol{K} - \Delta t \boldsymbol{D}\right) \boldsymbol{v}_{t+\Delta t} = \boldsymbol{M}\boldsymbol{v}_t + \Delta t \left(\boldsymbol{f}_e + \boldsymbol{f}_b\right), \quad (12)$$

where $\boldsymbol{M}$, $\boldsymbol{K}$, and $\boldsymbol{D}$ are the mass, stiffness, and damping matrices, respectively; Δt is the timestep, $\boldsymbol{v_t}$ and $\boldsymbol{v_{t+\Delta t}}$ are the velocities before and after the timestep, and $\boldsymbol{f}_e$ and $\boldsymbol{f}_b$ are elastic and body forces. We then update positions,

$$\boldsymbol{x}_{t+\Delta t} = \boldsymbol{x}_t + \Delta t \boldsymbol{v}_{t+\Delta t}. \quad (13)$$

We found that this scheme has the desirable properties of affording large timesteps, largely avoiding artificial damping, and requiring only a single Newton iteration. In contrast, the explicit and Newmark schemes are not competitive due to timestep restrictions, and computing velocity updates introduces substantial artificial damping. The variational integration scheme produces results very similar to our preferred scheme. In fact, some algebraic manipulation reveals that the schemes are quite similar, the main differences being that the variational scheme multiplies the stiffness matrix by $1/4$ and uses an average of force evaluations taken $\Delta t/2$ before and after the timestep. The elegance and accuracy of the variational scheme come at the cost of additional Newton iterations and force evaluations, which results in significantly longer computation times.

3.6 Timestep Summary

In this section we give an overview of the computations involved in our adaptive approach.

Compute forces. Elastic and damping forces are computed by looping over the elements, assembling a matrix of position differences $(\boldsymbol{x}_i - \boldsymbol{x}_0)$, and computing a polar decomposition. Our fast polar decomposition uses Jacobi iterations and is warm-started with the rotation from the previous timestep. Forces are then computed by rotating the current position, multiplying by the stiffness matrix, and rotating back

$$\boldsymbol{f_{ab}} = \boldsymbol{Q}\boldsymbol{K_{ab}}\boldsymbol{Q}^T\boldsymbol{x_b}, \quad (14)$$

where $\boldsymbol{f_{ab}}$ is the force exerted on control point $\boldsymbol{a}$ by control point $\boldsymbol{b}$. The force offsets must also be added (see Equation (10)) for elastic forces. We cache the rotation from the polar decomposition for later assembly of the global system matrix.

Choose degree. For each edge decide whether it should be linear or quadratic. If there is a change in status, enable or disable rows and columns in the global system matrix.

Handle constrained control points. For each constrained control point, set its position and velocity to the average of the values at the endpoints and apply forces that were intended for constrained edge midpoints to the edge endpoints, half to each.

Compute global system matrix. For each element, use the cached polar decompositions to rotate the element stiffness and damping matrices and place them into the global stiffness matrix, taking care to distribute contributions from constrained edge control points.

Update positions and velocities. Positions and velocities are updated according to the timestepping scheme. The scheme may require additional force computations. Collisions are usually handled between velocity and position updates. Self- and inter-object collisions are handled with the open-source *El Topo* library [Brochu and Bridson 2009].

So, what are the differences between an implementation of our approach and linear finite-element simulation? The answer is, *not much*. Our approach does require the step of determining the degree of the elements and a bit of care in dealing with the changing nonzero structure and constrained control points when assembling the global system matrix. There are also differences in the size of

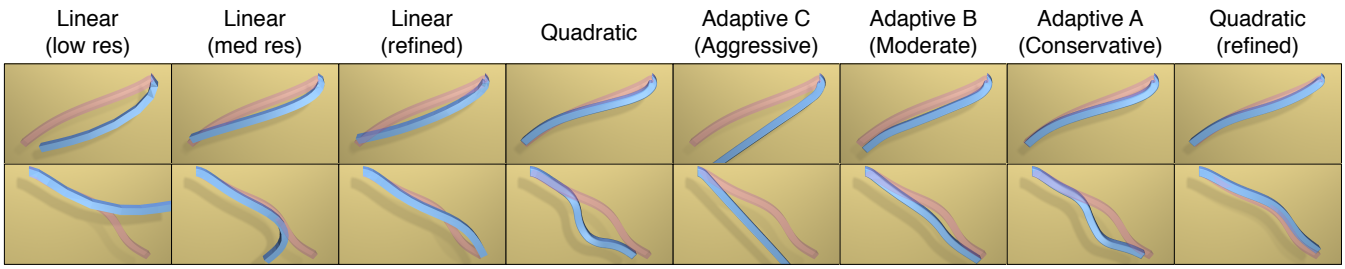

Fig. 3: *A simple simulation of a swinging bar. Despite the coarse mesh, the quadratic elements are able to quite closely match the high-resolution simulation (in pink). Two frames from each animation are displayed in the figure. See Table I for resolution details.*

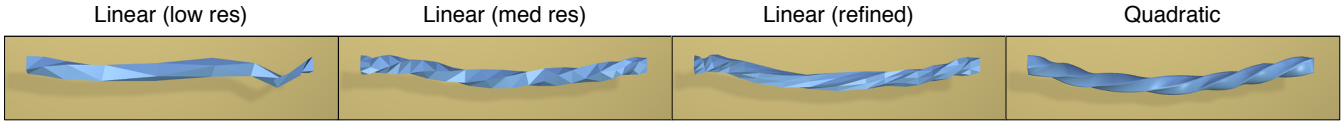

Fig. 4: *A twisting bar. Notice the continuous quadratic curves along the edges of the quadratic case. Adaptive simulations are nearly identical because all tetrahedral edges quickly become quadratic.*

things. We deal with 30×30 matrices instead of 12×12. While such computations are more time consuming, they are not conceptually any more difficult. Consequently, we believe an implementation of our approach is only slightly more difficult than linear finite elements.

3.7 Quadratic Rest Shapes

It is also possible to simulate some objects that have nonlinear rest shapes, such as mechanical parts modeled using Computer Aided Design (CAD) software or objects modeled using maya. While automatic volumetric meshing of such objects is an unexplored problem, simulation with such meshes is relatively straightforward.

We now discuss computation of the stiffness matrices in the case of quadratic rest shapes. While the $\boldsymbol{\beta}$ matrices in Equation (10) are constant for linear rest shapes, for nonlinear rest shapes they vary over the element and cannot be taken out of the integral. Consequently, we must resort to numerical quadrature. Specifically, at a given quadrature point we form the matrix $\partial \boldsymbol{u}/\partial \boldsymbol{\alpha}$ and invert it. We then multiply the relevant terms (see Equation (10)) from $\boldsymbol{\beta}$ and the derivatives of the Bernstein polynomials (which are sums of barycentric coordinates). The result is weighted by the determinant of $\partial \boldsymbol{u}/\partial \boldsymbol{\alpha}$ to account for the volume of the rest pose. The volume of the element is

$$\int_E \det\left(\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\alpha}}\right). \tag{15}$$

We experimented with a variety of quadrature schemes. Simple Monte Carlo techniques did not work, even for linear elements. We were generally successful with several other rules including the Keast Rule with 45 points [Burkardt 2007], the Newton Cotes rule with 84 points [Burkardt 2007], and the 46-point rule developed by Zhang and colleagues [2009]. All achieved similar results, working on most, but not all, of the cases that we tested. For highly nonlinear rest poses, the polynomial approximation that underlies numerical quadrature is a very poor approximation of the inverse polynomial function we wish to integrate. The result is stiffness matrices that yield implausible behavior. It is possible that even higher-order quadrature could result in usable system matrices, but we doubt this is the case under extreme deformations. For example, the failure case depicted in Figure 5 contains partially inverted tetrahedra—something that becomes possible with quadratic elements.

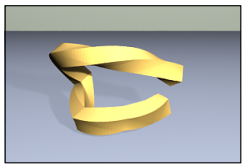

Fig. 5: *An example where numerical quadrature fails to compute accurate system matrices.*

4. RESULTS AND DISCUSSION

All of our tetrahedral meshes were generated with NETGEN [Schöberl 1997]. Collisions were detected and handled using the relevant sections of El Topo [Brochu and Bridson 2009]. All our examples used between two and four timesteps per (slow motion) frame. Unfortunately, El Topo is not really designed to work with such large timesteps and some artifacts are visible in the resulting videos.

Linear vs. Quadratic vs. Adaptive Elements. To evaluate the trade-offs involved in choosing between linear and quadratic elements we set up several very simple tests, including a bar that is pinned at one end and allowed to swing under gravity (see Figure 3), a bar that is twisted (see Figure 4), and a "sculpture" that is dropped on the ground (see Figure 6). We found that for low-degree-of-freedom systems quadratic elements perform very favorably, but that as the number of degrees of freedom increases, differences become difficult to discern. In retrospect this outcome is not particularly surprising—at the time-scales we care about in computer graphics, volumetric elastic effects are fairly low resolution. This fact has been exploited by numerous researchers who work with model reduction techniques (see, e.g., James and Pai [2002], Hauser et al. [2003], and Barbič and James [2005]) or embed

Table I. : *Timing Results and Mesh Statistics for the Examples in this Article*

simulation	nodes	tets	edges	non-0s	total	force	degree	build matrix	solve	collision
Hanging Bar										
linear (low res)	41	59	137	1602	0.00141	0.000154	—	0.000157	0.000467	—
linear (mid res)	154	313	608	6858	0.00751	0.00080	—	0.00089	0.00355	—
linear (refined)	178	472	801	8811	0.0102	0.00124	—	0.00126	0.00516	—
quadratic (lowres)	41	59	137	15318	0.00854	0.000252	—	0.00087	0.00588	—
adaptive A	41	59	137	9915	0.00726	0.000286	5.78e-05	0.00106	0.00407	—
adaptive B	41	59	137	6560	0.00589	0.000297	6.36e-05	0.00097	0.00272	—
adaptive C	41	59	137	3787	0.00384	0.000266	5.35e-05	0.00054	0.00143	—
overhead	41	59	137	1602	0.00283	0.00023	4.54e-05	0.00016	0.00084	—
quadratic (refined)	178	472	801	102366	0.101	0.00197	—	0.00679	0.0847	—
very hires	6341	30208	38980	407889	1.5	0.0759	—	0.0816	1.25	—
Twisting Bar										
linear (low res)	41	59	137	1602	0.0262	0.000553	—	0.000452	0.000634	0.0236
linear (mid res)	123	242	477	5400	0.0398	0.00263	—	0.00226	0.00361	0.0288
linear (refined)	178	472	801	8811	0.0483	0.00382	—	0.00335	0.00608	0.0321
quadratic (lowres)	41	59	137	15318	0.0205	0.000821	—	0.00242	0.00704	0.00847
Fin	2740	8705	13645	85736	1.66	0.0713	0.00697	0.212	0.804	0.428
Spirals	4848	7116	16776	155727	8.14	0.0682	0.00736	0.171	6.12	1.38
Chickens (adaptive)	4850	13125	22325	78356	3.45	0.2088	0.02511	0.597	1.716	—
Chickens (quadratic)	4850	13125	22325	114219	4.08	0.1992	—	0.585	2.454	—
Chickens (linear)	4850	13125	22325	9783	0.744	0.132	—	0.1188	0.1452	—

These are as measured on a late 2012 iMac with a 3.4 GHz *Core i7 processor and 32GB of memory. From left to right: total nodes, total tetrahedra, total edges, average nonzero entries in the global system matrix*[2]*, total time in seconds per frame, time spent computing elastic and damping forces (including polar decomposition), time spent choosing degree and adjusting simulation variables to deal with constrained edges, time spent assembling the global system matrix, time spent solving the linear system, time spent performing collision detection. The adaptive simulations grow increasingly aggressive in their linearization. The overhead row corresponds to an adaptive simulation with an infinite threshold for raising the degree (thus, the motion is identical to a linear simulation and the cost compared to the* linear (low res) *row gives the additional cost of running an adaptive simulation)*

high-resolution geometry in low-resolution simulation meshes (see, e.g., Capell et al. [2002], Sifakis et al. [2007], and Wojtan and Turk [2008]).

Conversely, we found that as resolution increases the advantages of adaptive elements over quadratic elements become more significant. Again this trend is not surprising; at higher resolutions deformation is better captured by linear elements. As seen in Table II, for very coarse models adaptivity saves about 10% of the running time, while in the "VeryFine" example adaptivity shaves nearly a quarter of the running time off the quadratic case, with much of the savings coming from the matrix solve. Computing forces takes longer in the adaptive examples due to slower convergence of the polar decompositions. In the included examples, we were unable to see differences between the adaptive and quadratic examples. More aggressive adaptivity produced high-quality results at lower cost, but the results were distinguishable from the fully quadratic simulations. Of course, as is evident in the hanging bar examples, over-aggressive adaptivity achieves impressive runtimes, but degrades animation quality.

Quadratic elements are clearly superior to linear elements in *cost per degree of freedom*. For linear elements, increasing the number of degrees of freedom requires increasing the number of elements, which leads to more polar decompositions and greater complexity in filling in the global system matrix. Hierarchically refining a tetrahedral mesh by splitting every edge yields the same number of degrees of freedom as using quadratic elements, but generates eight times as many elements. In addition to more polar decompositions, storing the element stiffness matrices for this refined mesh also requires more storage than the coarser quadratic mesh. At higher resolutions, our adaptive elements are able to focus computation on interesting regions of the simulation, at the cost of more complex force computations and global matrix multiplies, as can be seen by comparing the *overhead* row to the *linear (low-res)* row.

The example of the helices (see Figure 1) contains elements with nonlinear rest shapes obtained by setting the rest pose to the end result of a previous simulation using quadratic elements. The example with 25 rubber chickens (see Figure 7) demonstrates embedding high-resolution surface meshes in coarse adaptive-degree finite-element meshes. In this example the chickens collide with scene geometry, but self-collisions and inter-object collisions were disabled to maintain stability and a reasonable timestep.

Mass Lumping. We performed experiments with and without mass lumping. We found that mass lumping did result in different motion than the full mass matrix (integrals of the products of pairs of basis functions), and that these changes were most noticeable in coarse simulations. However, the full mass matrix resulted in substantially higher cost because the full mass matrix results in a

[2]Because each object stores its own global system matrices, this number is an average over all timesteps and objects in the scene.

UU IR Author Manuscript

Table II. : *Timing Results and Mesh Statistics for the Sculpture Examples*

simulation	nodes	tets	edges	non-0s	total	force	degree	build matrix	solve	collisions
Quadratic										
Coarse	124	268	519	62784	0.0281	0.00123	—	0.00396	0.00836	0.00932
Moderate	213	504	933	115155	0.0547	0.00235	—	0.00764	0.0196	0.016
Fine	505	1360	2354	299241	0.162	0.00657	—	0.0218	0.0684	0.0421
VeryFine	1851	6853	9979	1369854	0.97	0.0388	—	0.146	0.593	0.0988
Adaptive										
Coarse	124	268	519	45795	0.026	0.00125	0.00018	0.00386	0.00624	0.00921
Moderate	213	504	933	82452	0.05	0.0025	0.00033	0.0074	0.0143	0.0162
Fine	505	1360	2354	192812	0.142	0.00789	0.00107	0.0219	0.0449	0.0425
VeryFine	1851	6853	9979	688908	0.736	0.0467	0.00707	0.139	0.346	0.101
Linear										
Coarse	124	268	519	5787	0.0133	0.00078	—	0.00074	0.00064	0.00906
Moderate	213	504	933	10314	0.0233	0.00147	—	0.00138	0.00137	0.0161
Fine	505	1360	2354	25731	0.0557	0.00391	—	0.00371	0.00475	0.0361
VeryFine	1851	6853	9979	106470	0.204	0.0197	—	0.0205	0.0286	0.108

From left to right: nodes in the mesh, tetrahedra in the mesh, edges in the mesh, nonzero entries in the global system matrix (average over all timesteps), total time in seconds per frame, time spent computing elastic and damping forces (including polar decomposition), time spent choosing degree and adjusting simulation variables to deal with constrained edges, time spent assembling the global system matrix, time spent solving the linear system, time spent performing collision detection. The VeryFine *quadratic and adaptive simulations required a smaller timestep (20 steps per frame) than the others (10 steps per frame) to remain stable.*

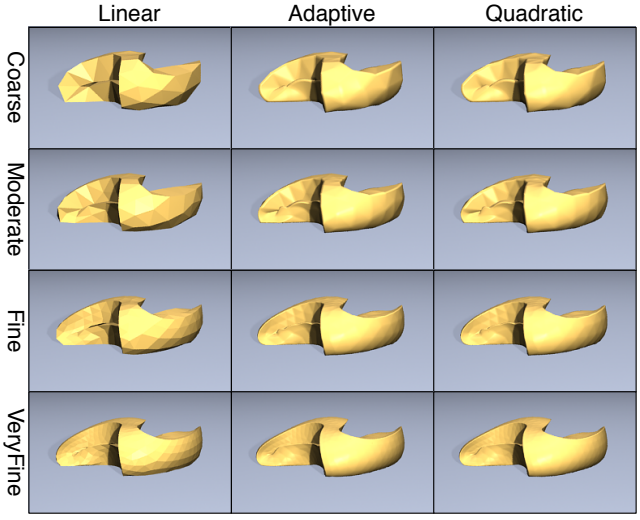

Fig. 6: *A sculpture model dropped on the ground with various mesh resolutions and both linear and quadratic elements. Sculpture model from the NETGEN distribution.*

significantly worse conditioning. This effect was more pronounced for the quadratic elements, where the mass was spread to more off-diagonal entries, than for linear elements. Consequently, all of our results used the lumped mass formulation.

Limitations and future work. There are many interesting directions for future work. One obvious question we would like to answer is whether there is any benefit to using cubic or higher-order elements. Such elements are significantly more complex. First, cubic tetrahedra have 20 control points and 60 degrees of freedom. The resulting stiffness matrices have 1830 unique elements. Such

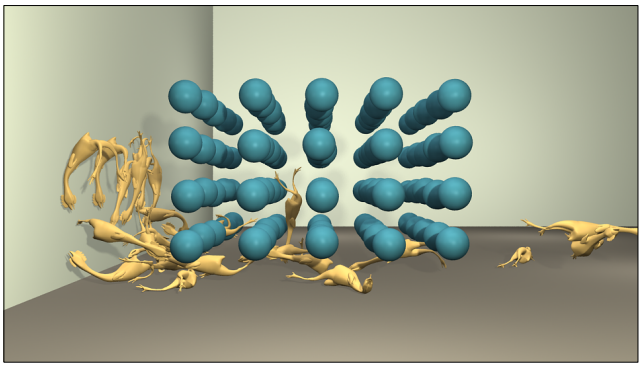

Fig. 7: *Twenty-five rubber chickens navigate obstacles. Rubber chicken model courtesy of Yong Wan.*

matrices would require nearly 20KB per element, not a prohibitive cost on modern computers, but not trivial either. Additionally, while our additional control points are easily constrained by edge endpoints, constraining the control points of cubic elements is more involved. This fact, in part, explains the popularity of the hierarchical Lagrange polynomials in high-order methods. An additional direction that may prove fruitful is the combination of high-order finite element with discontinuous Galerkin methods [Kaufmann et al. 2009]. Such methods are quite popular in other fields of engineering [Hesthaven and Warburton 2007]

We would also like to explore using our adaptive framework for other types of deformable body simulation, such as fracture and elastoplasticity. In the case of fracture, the need to create new elements would not allow us to preallocate the system matrix at the start of the simulation and may also void some of the advantage we gain from precomputing the per-element stiffness matrices. In the case of elastoplasticity, plastic deformation changes the entries in the β matrices and also makes our precomputation less of an advan-

tage. However, the integrals of the Bernstein polynomials are taken over the canonical tetrahedron and are the same for all elements. Therefore, we could avoid storing the per-element stiffness matrices and instead compute them on-the-fly. While this will surely increase the number of flops per timestep, it will also dramatically reduce storage.

One limitation of our implementation is the lack of adaptive timestepping, which we consciously decided not to include because we felt the inherent complexity of varying timestep size would obscure timing results in our experiments. Another limitation, mentioned earlier, is that streaming of element stiffness matrices during force computation results in a significant number of cache misses, though the rest of our computations demonstrate excellent cache behavior. It is possible that using a more structured mesh, such as those that result from isosurface stuffing [Labelle and Shewchuk 2007], would improve performance by limiting the number of distinct element shapes and unique element stiffness matrices.

In summary, we have explored the use of quadratic Bézier tetrahedra for computer animation of deformable bodies. We have integrated quadratic and linear elements into an adaptive simulation framework and we have demonstrated the first computer graphics animation of elastic bodies with nonlinear rest shapes. Our quadratic elements have clear advantages over linear elements, especially when working with very coarse simulation meshes. We expect our quadratic elements to find widespread usage when combined with high-resolution embedded surfaces and our adaptive elements to be popular for mid-scale simulation.

Acknowledgements

The authors wish to thank the anonymous reviewers for their time and helpful comments.

REFERENCES

BABUŠKA, I. AND SURI, M. 1990. The p- and h-p version of the finite element method, an overview. *Comput. Methods Appl. Mech. Eng. 80,* 1-3, 5–26.

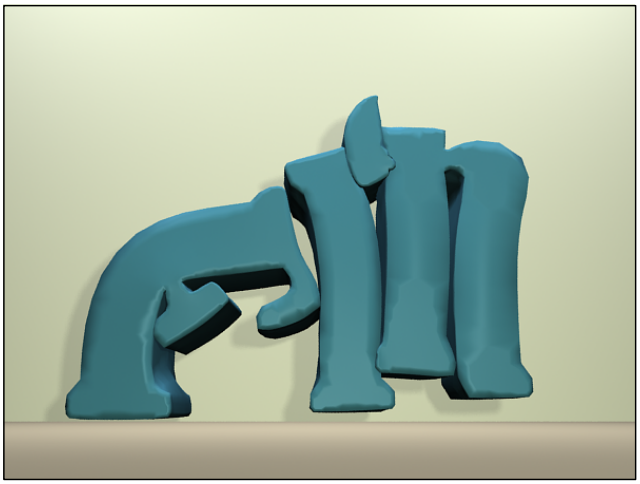

Fig. 8: *Another adaptive example.*

BABUŠKA, I., SZABO, B. A., AND KATZ, I. N. 1981. The p-version of the finite element method. *SIAM Journal on Numerical Analysis 18,* 3, 515–545.

BARAFF, D. AND WITKIN, A. 1998. Large steps in cloth simulation. In *The Proceedings of ACM SIGGRAPH*. ACM, New York, NY, USA, 43–54.

BARBIČ, J. AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. Graph. 24,* 3, 982–990.

BICKEL, B., BÄCHER, M., OTADUY, M. A., MATUSIK, W., PFISTER, H., AND GROSS, M. 2009. Capture and modeling of non-linear heterogeneous soft tissue. *ACM Trans. Graph. 28,* 3 (July), 89:1–89:9.

BONET, J. AND WOOD, R. 2008. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press.

BORDEN, M. J., SCOTT, M. A., EVANS, J. A., AND HUGHES, T. J. R. 2011. Isogeometric finite element data structures based on bŁzier extraction of nurbs. *Int. J. Numer. Meth. Engng. 87*, 15–47.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 28–36.

BROCHU, T. AND BRIDSON, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing 31,* 4, 2472–2493.

BURKARDT, J. 2007. Quadrature rules for tetrahedrons. `http://people.sc.fsu.edu/~jburkardt`.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21*, 586–593.

CHAO, I., PINKALL, U., SANAN, P., AND SCHRÖDER, P. 2010. A simple geometric model for elastic deformations. *ACM Trans. Graph. 29,* 4 (July), 38:1–38:6.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. H. 2001. Dynamic real-time deformations using space & time adaptive sampling. In *The Proceedings of ACM SIGGRAPH*. ACM, New York, NY, USA, 31–36.

GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. 2002. CHARMS: a simple framework for adaptive simulation. *ACM Trans. Graph. 21,* 3 (July), 281–290.

HAUSER, K. K., SHEN, C., AND O'BRIEN, J. F. 2003. Interactive deformation using modal analysis with constraints. In *The Proceedings of Graphics Interface*. 247–256.

HESTHAVEN, J. S. AND WARBURTON, T. 2007. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, 1st ed. Springer Publishing Company, Incorporated.

HUGHES, T. J. R. 1987. *The finite element method : linear static and dynamic finite element analysis*. Englewood Cliffs, N.J. Prentice-Hall International.

IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible finite elements for robust simulation of large deformation. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 131–140.

JAMES, D. L. AND PAI, D. K. 2002. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. *ACM Trans. Graph. 21,* 3, 582–585.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26,* 3 (July).

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24,* 3 (July), 561–566.

UU IR Author Manuscript

KALDOR, J. M., JAMES, D. L., AND MARSCHNER, S. 2008. Simulating knitted cloth at the yarn level. *ACM Trans. Graph. 27,* 3 (Aug.), 65:1–65:9.

KAUFMANN, P., MARTIN, S., BOTSCH, M., GRINSPUN, E., AND GROSS, M. 2009. Enrichment textures for detailed cutting of shells. *ACM Trans. Graph. 28,* 3 (July), 50:1–50:10.

KAUFMANN, P., MARTIN, S., BOTSCH, M., AND GROSS, M. 2009. Flexible simulation of deformable models using discontinuous galerkin fem. *Graph. Models 71,* 4, 153–167.

KHAREVYCH, L., YANG, W., TONG, Y., KANSO, E., MARSDEN, J. E., SCHRÖDER, P., AND DESBRUN, M. 2006. Geometric, variational integrators for computer animation. In *The Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 43–51.

LABELLE, F. AND SHEWCHUK, J. R. 2007. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26,* 3 (July).

MARTIN, S., KAUFMANN, P., BOTSCH, M., WICKE, M., AND GROSS, M. 2008. Polyhedral finite elements using harmonic basis functions. *Computer Graphics Forum 27,* 5, 1521–1529.

MEZGER, J., THOMASZEWSKI, B., PABST, S., AND STRASSER, W. 2009. Interactive physically-based shape editing. *Comput. Aided Geom. Des. 26,* 6 (Aug.), 680–694.

MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 49–54.

MÜLLER, M. AND GROSS, M. 2004. Interactive virtual materials. In *The Proccedings of Graphics Interface*. 239–246.

O'BRIEN, J. F. AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *The Proceedings of ACM SIGGRAPH*. 137–146.

ORSZAG, S. 1969. Numerical methods for the simulation of turbulence. *Phys. Fluids Suppl. II. 12*, 250–257.

PARKER, E. G. AND O'BRIEN, J. F. 2009. Real-time deformation and fracture in a game environment. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 156–166.

PATERA, A. 1984. A spectral element method for fluid dynamics: Laminar flow in a chennel expansion. *Journal of Computational Physics 54*, 468–488.

REMION, Y., NOURRIT, J.-M., AND GILLARD, D. 1999. Dynamic animation of spline like objects. In *The Proceedings of WSCG'99*, V. Skala, Ed.

ROTH, S. H. M., GROSS, M. H., TURELLO, S., AND CARLS, F. R. 1998. A bernstein-bzier based approach to soft tissue simulation. *Computer Graphics Forum 17,* 3, 285–294.

SCHÖBERL, J. 1997. NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science 1,* 1, 41–52.

SIFAKIS, E., SHINAR, T., IRVING, G., AND FEDKIW, R. 2007. Hybrid simulation of deformable solids. In *The Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*. 81–90.

STOMAKHIN, A., HOWES, R., SCHROEDER, C., AND TERAN, J. M. 2012. Energetically consistent invertible elasticity. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 25–32.

TERAN, J., BLEMKER, S., HING, V. N. T., AND FEDKIW, R. 2003. Finite volume methods for the simulation of skeletal muscle. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 68–74.

WEBER, D., BENDER, J., SCHNOES, M., STORK, A., AND FELLNER, D. 2013. Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32,* 1, 16–26.

WEBER, D., KALBE, T., STORK, A., FELLNER, D., AND GOESELE, M. 2011. Interactive deformable models with quadratic bases in Bernstein-Bézier-form. *TVC 27*, 473–483.

WICKE, M., BOTSCH, M., AND GROSS, M. 2007. A finite element method on convex polyhedra. *Computer Graphics Forum 26,* 3, 355–364.

WOJTAN, C. AND TURK, G. 2008. Fast viscoelastic behavior with thin features. *ACM Trans. Graph. 27*, 47:1–47:8.

ZHANG, L., CUI, T., AND LIU, H. 2009. A set of symmetric quadrature rules on triangles and tetrahedra. *J. Comp. Math. 27,* 1, 89–96.

UU IR Author Manuscript