Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2014)
J. Bender, C. Duriez, F. Jaillet, and G. Zachmann (Editors)

A p-Multigrid Algorithm using Cubic Finite Elements for Efficient Deformation Simulation

Daniel Weber[†][1,2], Johannes Mueller-Roemer[1,2], Christian Altenhofen[1,2], André Stork[1,2] and Dieter Fellner[1,2,3]

[1]Fraunhofer IGD, Darmstadt, Germany, [2]GRIS, TU Darmstadt, Germany, [3]CGV, TU Graz, Austria

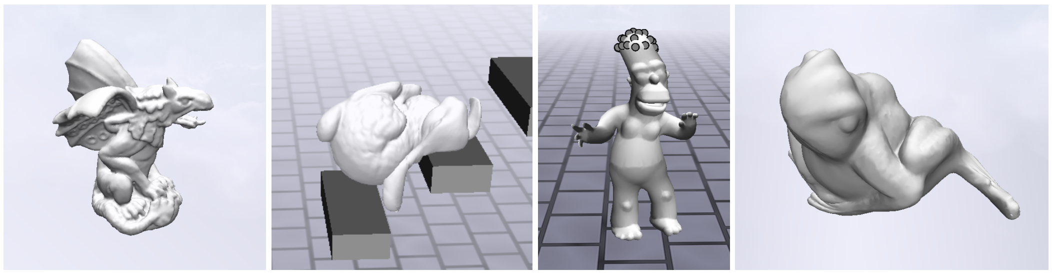

Figure 1: *The pictures show simulations using higher order finite elements (FE) with our p-multigrid algorithm. From left to right: Animation of a gargoyle using quadratic FE. A bunny with quadratic FE colliding with obstacles. A toy is animated by changing boundary conditions (cubic FE). A frog model with twisted deformation (cubic FE).*

Abstract

We present a novel p-multigrid method for efficient simulation of co-rotational elasticity with higher-order finite elements. In contrast to other multigrid methods proposed for volumetric deformation, the resolution hierarchy is realized by varying polynomial degrees on a tetrahedral mesh. We demonstrate the efficiency of our approach and compare it to commonly used direct sparse solvers and preconditioned conjugate gradient methods. As the polynomial representation is defined w.r.t. the same mesh, the update of the matrix hierarchy necessary for co-rotational elasticity can be computed efficiently. We introduce the use of cubic finite elements for volumetric deformation and investigate different combinations of polynomial degrees for the hierarchy. We analyze the applicability of cubic finite elements for deformation simulation by comparing analytical results in a static scenario and demonstrate our algorithm in dynamic simulations with quadratic and cubic elements. Applying our method to quadratic and cubic finite elements results in speed up of up to a factor of 7 for solving the linear system.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

In the realm of computer graphics several classes of algorithms have been proposed for simulating volumetric deformation. These include, among others, position-based dynamics [BMOT13], mass-spring systems and finite element methods [MG04] (FEM). FEM is a popular choice when intuitive material parameters and accuracy are important, as these methods are based on continuum mechanics and do not require parameter tuning to achieve physically plausible results. Instead of the commonly used linear basis functions, some authors propose to simulate volumetric deformation on

[†] e-mail: daniel.weber@igd.fraunhofer.de

DOI: 10.2312/vriphys.20141223

tetrahedral meshes using quadratic finite elements, i.e., using polynomials of degree two (see, e.g., the work of Mezger et al. [MS06] or recently Bargteil et al. [BC14]). This leads to improved accuracy as higher polynomial degrees can better approximate the solution of the partial differential equation. Even though the number of degrees of freedom per element increases, the computational time can be reduced as the desired deformation can be represented with significantly fewer elements.

However, simulating deformation with finite elements is computationally expensive, as for a stable simulation, large, sparse linear systems must be solved in every time step. Typically, direct solvers like sparse Cholesky factorization or the method of conjugate gradients (CG) with preconditioning are chosen to solve these linear systems. These are usually the bottleneck of the simulation algorithm and become even more critical with increasing model sizes. A frequent compromise is to use a fixed number of CG iterations, but this in turn increases the numerical damping of the simulation and dissipates energy as we will demonstrate. In this paper we address this issue and propose a geometric p-multigrid method to efficiently and accurately solve sparse linear systems arising from higher order finite elements.

In our approach, we construct a hierarchy of varying degree polynomials to represent the field of unknowns. These are defined on the same tetrahedral discretization to iteratively improve the solution of the linear system. In contrast to other multigrid approaches for deformation simulation these levels vary in polynomial degree instead of mesh resolution. Furthermore, we introduce volumetric deformation simulation using cubic finite elements on tetrahedral meshes, which have not been proposed so far in the computer graphics community to the best of our knowledge. We represent the shape functions as polynomials in Bernstein-Bézier form (B-form) and show how restriction and prolongation of shape functions can be incorporated into the polynomial hierarchy. Our contributions are as follows:

- We introduce a novel multigrid solver for volumetric deformation with higher order finite elements
- We present deformation simulations with cubic finite elements in B-form
- We show how polynomial representations in B-form of different degrees can be efficiently transformed into each other
- We demonstrate a speed up for higher order simulations up to a factor of 7 for solving the linear system in comparison to a preconditioned CG method.

2. Related Work

In the realm of computer graphics many methods for volumetric deformation simulation have been proposed. Most of the earlier work is summarized in the survey of Nealen et al. [NMK*06]. Besides methods based on solving the partial differential equations of linear, co-rotational or non-linear elasticity there are a number of other techniques for physically plausible volumetric deformation. In the early work of Baraff and Witkin [BW98] a numerical integrator for mass-spring systems and constraints was introduced allowing for arbitrarily large time steps. Although mass-spring systems have not been the focus of research for a long time, recently a block coordinate descent method was proposed by Liu et al. [LBOK13] where a constant stiffness matrix allows for efficient simulation. In the context of position-based dynamics a good summary of the relevant work is described in the survey of Bender et al. [BMOT13]. In contrast to these approaches we rely on continuum mechanical modeling of elasticity. This has the advantage that the simulation parameters such as, e.g., the material parameters have an intuitive meaning instead of parameters which require tuning depending on models and time step sizes.

Physically-based simulation of deformation with FEM was first adopted by O'Brien and Hodgins [OH99]. They modeled and simulated brittle fracture using an explicit time stepping method. Mueller and Gross [MG04] used implicit time integration together with a co-rotational formulation. This allowed for stable simulations and avoided the artifacts of linear elasticity by recomputing the reference coordinate system. Irving et al. [ITF04] presented a method to cope with inverted tetrahedral elements that occur when large forces are present. Parker and O'Brien [PO09] demonstrated how the co-rotational formulation for simulating deformation and fracture can be applied with strict computation time constraints in a gaming environment. Kaufmann et al. [KMBG08] used a discontinuous Galerkin method to simulate volumetric deformation. An extension of the co-rotational method that takes the rotational derivatives into account has been presented by Chao et al. [CPSS10], achieving energy conservation. In order to allow for larger time steps without numerical damping Michels at al. [MSW14] introduced exponential integrators for long-term stability due to energy conservation.

Simulation with higher-order finite elements is widespread in engineering applications (see e.g. [ZT00]) where linear shape functions do not provide sufficient accuracy. In the computer graphics community quadratic finite elements were used for deformation simulation [MS06] and interactive shape editing [MTPS08]. In a previous work [WKS*11] we used quadratic finite elements in B-form for the simulation of volumetric deformation. Later we developed a GPU implementation for linear and quadratic finite elements [WBS*13]. Recently, Bargteil and Cohen [BC14] presented a framework for adaptive finite elements with linear and quadratic B-form polynomials. Our method builds upon the work of Bargteil and Cohen and extends it by additionally introducing cubic finite elements and employing an efficient method for solving the governing linear systems.

Multigrid methods in general have been the subject of extensive research. Standard textbooks like [TS01] and [BHM00] provide a good overview of the basic method and its theory. Geometric multigrid methods are especially suited for discretizations on regular grids. In the context of deformation simulation Zhu et al. [ZSTB10] propose a multigrid framework based on finite differences. Dick et al. [DGW11b] use hexahedral finite elements discretization on a regular grid and solve the linear systems using a GPU-based multigrid method. In [DGW11a] they extend this approach for simulating cuts. In contrast, our method employs a discretization on tetrahedral meshes, which allow for an adaptive approximation of the simulated geometry potentially requiring less elements.

Based on tetrahedral meshes Georgii et al. [GW05] proposed a geometric multigrid algorithm based on linear finite elements using nested and non-nested mesh hierarchies. In general this geometric concept cannot be easily adapted for higher order finite elements. In their work the computational bottleneck is the matrix update, where sparse matrix-matrix products (SpMM) are required on every level to update the multigrid hierarchy. Later in [GW08] they specifically developed an optimized SpMM to address this bottleneck and report a speed-up of one order of magnitude. However, the matrix update is still as expensive as the time for applying the multigrid algorithm itself. In contrast, our p-multigrid method employs polynomial hierarchies on a common tetrahedral mesh. As the problem is directly discretized the expensive SpMM operations are avoided.

For two-dimensional analysis of elliptic boundary value problems, Shu et al. [SSX06] introduced a p-multigrid for finite elements using a higher-order Lagrangian basis. To the best of our knowledge we are the first to introduce this concept in three dimensions, to employ polynomials in B-form and to solve equations of elasticity with this algorithm.

3. Higher-Order Finite Element Discretization of Elasticity

In this section we briefly outline the general approach for simulating volumetric deformation using higher order finite elements. First, we describe the general process of higher order discretization and then outline the steps necessary for co-rotational elasticity.

3.1. Finite Element Discretization

Given a tetrahedral mesh $\triangle := (\mathcal{T},\mathcal{F},\mathcal{E},\mathcal{V})$ with a set of tetrahedra $\mathcal{T}$, faces $\mathcal{F}$, edges $\mathcal{E}$ and vertices $\mathcal{V}$ approximating the simulation domain Ω, we employ a finite element discretization by constructing a piecewise polynomial representation. A degree p polynomial w.r.t. a tetrahedron $\mathbb{T} = [\mathbf{v}_0,\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3]$ with vertices $\mathbf{v}_i$ is defined by a fixed number of values at nodes. Depending on the order p there are $\binom{p+3}{3}$ degrees of freedom associated with one tetrahedral element at the nodes:

$$\xi_I = \xi^{\mathbb{T}}_{ijkl} = \frac{i\mathbf{v}_0 + j\mathbf{v}_1 + k\mathbf{v}_2 + l\mathbf{v}_3}{p}, \quad |I| = i+j+k+l = p.$$

Here, we use capital letters for a multi-index to simplify the notation, e.g., I represents the set of non-negative indices i,j,k,l. Figure 2 shows an example for the node numbering of one tetrahedral element.

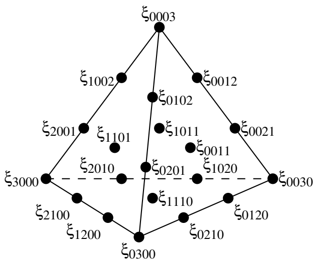

Figure 2: *Node numbering for a quadratic Bézier tetrahedron. The nodes at the corners of the element are associated with the vertices, e.g., ξ_{3000} corresponds to $\mathbf{v}_0$.*

We represent the field of unknowns as polynomial in B-form, i.e., the degree p basis functions are the $\binom{p+3}{3}$ Bernstein polynomials

$$B^p_I(\mathbf{x}) = B^p_{ijkl}(\mathbf{x}) = \frac{p!}{i!j!k!l!}\lambda^i_0(\mathbf{x})\lambda^j_1(\mathbf{x})\lambda^k_2(\mathbf{x})\lambda^l_3(\mathbf{x})$$

where $\lambda_i(\mathbf{x})$ are the barycentric coordinates. Just as for the nodes, the indices sum up to the polynomial degree $|I| = i + j+k+l = p$ reflecting that one basis function is associated to the node with the same index. A polynomial in B-form per element $\mathbb{T}$ is then defined by

$$q(\mathbf{x}) = \sum_{|I|=p} b_I B^p_I(\mathbf{x})$$

where in the general form the unknowns b_I can be scalars, vectors or tensors. A general advantage of modeling the field of displacements with this type of shape function is that the computation of integrals reduces to a sum of coefficients with binomial weights [WKS*11]. For finite element simulations, typically integrals of products of basis functions need to be computed. As polynomials in B-form are defined w.r.t. barycentric coordinates the chain rule applies when differentiating:

$$\frac{\partial B^p_I}{\partial x_a} = \sum_{i=0}^{3} \frac{\partial B^p}{\partial \lambda_i}\frac{\partial \lambda_i}{\partial x_a} \tag{1}$$

More details regarding polynomials in B-form and their properties regarding differentiation, multiplication and integration can be found in Lai and Schumaker's textbook [Sch07].

In contrast to discontinuous Galerkin methods

[KMBG08], finite elements are C^0-continuous across element boundaries. There, fewer degrees of freedom are required, as adjacent elements share the nodes. Therefore, the number of nodes n not only depends on the number of tetrahedra and the polynomial order but also on the topology of the mesh. The number of degrees of freedom can be computed by [Sch07]

$$n = |\mathcal{V}| + (p-1)|\mathcal{E}| + \binom{p-1}{2}|\mathcal{F}| + \binom{p-1}{3}|\mathcal{T}|, \quad (2)$$

which is $|\mathcal{V}|$ for $p=1$, $|\mathcal{V}|+|\mathcal{E}|$ for $p=2$, $|\mathcal{V}|+2|\mathcal{E}|+|\mathcal{F}|$ for $p=3$.

3.2. Co-rotational elasticity with higher order FEM

For linear and co-rotational elasticity we model the field of displacements $\mathbf{u}(\mathbf{x})$ as polynomials in B-form. By differentiating $\mathbf{u}(\mathbf{x})$ w.r.t. to space the spatially varying deformation gradient $\mathbf{F}$ is obtained. For co-rotational elasticity a polar decomposition $\mathbf{F} = \mathbf{RS}$ of the deformation gradient must be computed to obtain the rotation $\mathbf{R}$ of each element. For isotropic materials the Youngs modulus E and the poisson ratio ν or the Lamé coefficients λ_L and μ_L are required to set up the stiffness matrix. For more details on discretizing co-rotational elasticity on tetrahedral meshes we refer to the course notes of Sifakis and Barbic [SB12].

For a degree-p discretization the initial element stiffness matrices ${}^0\mathbf{K}^p_{\mathbb{T}}$ must be computed for each element $\mathbb{T}$. A total of $\binom{p+3}{3}$ 3×3 block matrices are computed, one for each of the elements $\binom{p+3}{3}$ node pairs. As described by Weber et al. [WKS*11] one block of the stiffness matrix for a pair of nodes I,J is computed by

$$\left[\mathbf{K}^p_{\mathbb{T}_{(I,J)}}\right]_{ab} = \int_{\mathbb{T}} \lambda_L \frac{\partial B^p_I}{\partial x_a}\frac{\partial B^p_J}{\partial x_b} + \mu_L \frac{\partial B^p_I}{\partial x_b}\frac{\partial B^p_J}{\partial x_a} + \mu_L\left(\sum_{c=0}^{2}\frac{\partial B^p_I}{\partial x_c}\frac{\partial B^p_J}{\partial x_c}\right)\delta_{ab}\, dV. \quad (3)$$

Here, λ_L and μ_L are the Lamé coefficients. For example, differentiation of cubic Bernstein polynomials B^3_I and B^3_J results in quadratic polynomials B^2. Therefore, following Weber et al. one entry without considering the chain rule of Eq.(1) is

$$\int_{\mathbb{T}} \frac{\partial B^3_I}{\partial\lambda_c}\frac{\partial B^3_J}{\partial\lambda_d} dV = 9G(I,J)\int_{\mathbb{T}} B^4\, dV = \frac{1}{35}G(I,J)V_{\mathbb{T}}.$$

Here, $G(I,J)$ are binomial coefficients that are computed as

$$G(I,J) = \frac{\binom{i+\alpha}{i}\binom{j+\beta}{j}\binom{k+\gamma}{k}\binom{l+\delta}{l}}{\binom{2p}{p}}$$

where I and J are multi-indices for i,j,k,l and $\alpha,\beta,\gamma,\delta$, respectively and $p=3$ for cubic polynomials.

The element matrices $\mathbf{K}^p_{\mathbb{T}}$ then need to be assembled into the global stiffness matrix $\mathbf{K}^p$, where a 3×3 entry is non-zero if the corresponding two degrees of freedom share a tetrahedron. In case of co-rotational elasticity the global and element stiffness matrices must be updated using

$$\mathbf{K}^p_{\mathbb{T}} = \mathbf{R}\,{}^0\mathbf{K}^p_{\mathbb{T}}\,\mathbf{R}^T \quad (4)$$

in every time step by determining per element rotations $\mathbf{R}$. Similar to Bargteil et al. [BC14] we only use the linear part of the deformation gradient to compute the polar decomposition. For the polynomial hierarchy, we only have to compute the rotation matrix once per element and can reuse the rotation matrices on the other levels.

For a dynamic simulation the lumped mass matrix $\mathbf{M}^p$ is required. Omitting damping terms, a system of ordinary equations

$$\mathbf{M}^p\ddot{\mathbf{u}}^p + \mathbf{K}^p\mathbf{u}^p = \mathbf{f}^p, \quad (5)$$

is set up. Here, $\mathbf{f}^p$ are the forces and the dots are derivatives w.r.t. time, i.e., $\ddot{\mathbf{u}}^p$ is the acceleration of the displacements or equivalently the acceleration of positions. We added the superscript p to denote that this system is additionally parametrized by the polynomial degree. The matrices depend on the degree p and the vectors $\ddot{\mathbf{u}}^p$, $\mathbf{u}^p$ and $\mathbf{f}^p$ are the combined degrees of freedom and therefore represent the time dependent fields in polynomial B-form. Employing implicit time-integration [BW98], a sparse linear system

$$(\mathbf{M}^p + \Delta^2\mathbf{K}^p)\Delta\mathbf{v}^p = \Delta t(\mathbf{f} + \Delta t\mathbf{K}\mathbf{v}) \quad (6)$$

is set up. Solving this, the change of $\dot{\mathbf{u}}^p$, $\Delta\mathbf{v}^p$ is computed. Then the new velocities are determined by $\mathbf{v}^p_{n+1} = \mathbf{v}^p_n + \Delta\mathbf{v}^p$ and the new positions $\mathbf{p}^p_{n+1} = \mathbf{p}^p_n + \Delta t\mathbf{v}^p$ at all finite element nodes are updated. For brevity we rewrite Eq. (6) as

$$\mathbf{A}^p\mathbf{x}^p = \mathbf{b}^p. \quad (7)$$

Solving this system is computationally expensive and takes a large part of the time required for a simulation step. However, it is important to solve this accurately as otherwise additional numerical damping is generated (see Section 6).

In Section 4 we explain how to efficiently solve this equation by making use of a polynomial hierarchy with varying degree p on the same tetrahedral mesh. In our approach the matrices in Eq. (7) are computed independently by simply discretizing with a different polynomial degree.

4. Multigrid method

The basic, non-recursive multigrid V-cycle given in Listing 1 consists of five operations: restriction ($\mathbf{I}^{l+1}_l$), prolongation ($\mathbf{I}^l_{l+1}$), sparse matrix-vector multiplication ($\mathbf{A}_l\mathbf{x}_l$), smoothing and the exact solution on the lowest level ($\mathbf{A}^{-1}_L\mathbf{x}_L$).

Typically, level 0 corresponds to the finest grid and level L to the coarsest grid. However, in the p-multigrid method on constant grids presented here, the levels correspond to varying polynomial degrees. The algorithm for computing the

```
def mgvcycle($\mathbf{x}_0, \mathbf{b}_0$):
    for $l$ in $[0 \dots L)$:
        $\mathbf{x}_l := \text{smooth}(\mathbf{A}_l, \mathbf{x}_l, b_l)$
        $\mathbf{r}_l := \mathbf{b}_l - \mathbf{A}_l \mathbf{x}_l$
        $\mathbf{b}_{l+1} := \mathbf{I}_l^{l+1} \mathbf{r}_l$; $\mathbf{x}_{l+1} := \mathbf{0}$
    $\mathbf{x}_L := \mathbf{A}_L^{-1} \mathbf{b}_L$
    for $l$ in $(L \dots 0]$:
        $\mathbf{x}_l := \mathbf{x}_l + \mathbf{I}_{l+1}^l \mathbf{x}_{l+1}$
        $\mathbf{x}_l := \text{smooth}(\mathbf{A}_l, \mathbf{x}_l, b_l)$
    return $\mathbf{x}_0$
```

Listing 1: *Multigrid V-cycle*

restriction and prolongation between the levels is outlined in section 4.1 and is a central part of our p-multigrid algorithm that is distinct to other multigrid approaches. The other procedures are similar to standard multigrid implementations and are briefly described here. Further details of standard multigrid theory can be found in standard textbooks (see, e.g., [TS01, BHM00]).

Intergrid Transfer: The intergrid transfer operators $\mathbf{I}_l^{l+1}$ and $\mathbf{I}_{l+1}^l$ perform the mapping between different multigrid levels. In the case of p-multigrid on constant grids they correspond to transformations between the respective polynomial representations, as described in Section 4.1.

Multigrid algorithms require the computation of matrices on all levels. In algorithmic multigrid procedures one determines the intergrid transfer operators and then computes the matrices using the Galerkin coarse grid operator, i.e., the matrix for level $l+1$ is computed as $\mathbf{A}_{l+1} = \mathbf{I}_l^{l+1} \mathbf{A}_l \mathbf{I}_{l+1}^l$. This requires multiple expensive SpMM operations. In contrast, we discretize different resolutions directly, i.e., with different polynomial degrees, which significantly accelerates the overall solution process.

Smoothing: The result of the prolongated solution on the lower level is a correction term $\mathbf{I}_{l+1}^l \mathbf{x}_{l+1}$ which is added to the current level. Although this correction term improves the solution for the low-frequency part of the solution, a high-frequency error term remains which is removed by smoothing. Analogously high-frequency components of the residual must be removed before restriction by a similar smoothing operator. In our case low-frequency components correspond to lower-order polynomial terms and high-frequency components to higher-order polynomial terms. The simplest form of smoothing is so-called point smoothing using a fixed number of Gauss-Seidel or weighted Jacobi iterations. We used 5 Gauss-Seidel iterations as a smoothing operator in our experiments.

Exact Solution: For the exact solution on the coarsest level L with the smallest polynomial degree, we use a standard preconditioned conjugate gradient solver (PCG) with an relative residual threshold of 10^{-3}.

4.1. Polynomial intergrid transfer

As described above, our restriction and prolongation operators perform transformations between different polynomial representations. Therefore, each level l in our multigrid hierarchy corresponds to a polynomial of degree p_l. On the lowest level $p_L = 1$ which corresponds to linear finite elements. On the highest level $p_0 = p_s$, where p_s is the polynomial degree of the simulation. In our examples p_s is either 2 or 3.

To apply the prolongation operator, we note that a polynomial of degree p can be exactly represented by a polynomial of degree $p+1$. As we model the degrees of freedom using polynomials in B-form (see Section 3), a standard degree elevation algorithm can be used. The polynomial transformation can be computed by constructing values at each of the $\binom{(p+1)+3}{3}$ nodes [Sch07]:

$$b_{ijkl}^{p+1} = \frac{ib_{i-1jkl}^p + jb_{ij-1kl}^p + kb_{ijk-1l}^p + lb_{ijkl-1}^p}{p+1} \tag{8}$$

Here, the superscripts denote the polynomial degree of the respective discretization.

For restriction, we perform a degree reduction which is lossy by definition. We follow the approach given by Farin [Far02] for univariate Bézier curves and extend it to the tetrahedral case. Equation (8) can be rewritten as a system of linear equations

$$\mathbf{b}^{p+1} = \mathbf{P}\mathbf{b}^p. \tag{9}$$

The resulting matrix $\mathbf{P}$ is rectangular with $\binom{(p+1)+3}{3}$ rows and $\binom{p+3}{3}$ columns. The rows corresponding to b_{ijkl}^{p+1} where one of the indices equals $p+1$ (i.e, rows associated to vertices) contain only a single non-zero unit entry at the column corresponding to b_{ijkl}^p where the same index equals p. As the system has more equations than unknowns it must be solved in the least squares sense as given by

$$\mathbf{b}^p = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{b}^{p+1} = \mathbf{P}^+\mathbf{b}^{p+1} \tag{10}$$

However, this introduces dependencies of the vertex values on the other control points. Due to the C^0 continuity requirement in the finite element method, vertices, edges and triangles of neighboring elements must share degrees of freedom. One option would be to combine the matrices P for each tetrahedron into a large matrix and computing the pseudoinverse of that matrix. Although this matrix is only dependent on the topology and can therefore be precomputed, doing so would adversely affect memory consumption and processing power requirements as sparse but large matrices for the restriction operation need to be stored. Therefore we choose local, mesh-based scheme, i.e., we process subsimplices in order, i.e., first the vertices which are transferred directly, then the edges, the triangles and finally the tetrahedra (the

```
def precompute():
    for all $\mathbb{T}$:
        $\frac{\partial \lambda_i^{\mathbb{T}}}{\partial \mathbf{x}}$ := calculate_derivatives($\mathbb{T}$)
        for $p$ in $[1, p_s]$:
            ${}^0\mathbf{K}_{\mathbb{T}}^p$ = calculate_stiffness_matrix($\mathbb{T}$)
    for $p$ in $[1, p_s]$:
        $\mathbf{M}^p$ := calculate_mass_matrix()

def simulation_step():
    $\mathbf{R}, \mathbf{S}$ := decompose($\mathbf{F}$)
    for $p$ in $[1, p_s]$:
        $\mathbf{K}_{\mathbb{T}}^p := \mathbf{R}\,{}^0\mathbf{K}_{\mathbb{T}}^p\,\mathbf{R}^T$
        $\mathbf{K}^p$ := assemble_matrix($\mathbf{K}_{\mathbb{T}}^p$ for all $\mathbb{T}$)
        $\mathbf{A}^p := \mathbf{M}^p + \Delta^2 \mathbf{K}^p$
    $\mathbf{f}^{p_s}$ := compute_forces()
    $\mathbf{b}^{p_s} := \Delta t(\mathbf{f}^{p_s} + \Delta t \mathbf{K}^{p_s} \mathbf{v}^{p_s})$
    $\Delta \mathbf{v}^{p_s} := (\mathbf{A}^{p_s})^{-1} \mathbf{b}^{p_s}$ # $p-multigrid$
    $v^{p_s}$ := $v^{p_s}$ + $\Delta \mathbf{v}^{p_s}$
    $\mathbf{x}^{p_s} = \mathbf{x}^{p_s} + \Delta t \mathbf{v}^p$
```

Listing 2: *Simulation algorithm for p_s levels. Solving the linear system with the p-multigrid algorithm launches several v-cycles (see Listing 1) of depth p_s.*

latter two are only relevant for simulation degrees higher than 3 which we did not evaluate). After computing the vertices, the rows and columns associated with these elements are removed and their contributions are added to the right hand side of the equation resulting in a modified system

$$\mathbf{b'}^{p+1} = \mathbf{P'}\mathbf{b'}^{p}. \qquad (11)$$

In the cubic to quadratic and the quadratic to linear cases, an analytical solution can easily be found and is used as described in Section 5.

5. Algorithm

Co-rotational elasticity: In this section, we outline the general algorithm for computing co-rotational elasticity with higher-order finite elements and provide implementation and optimization details. Listing 2 outlines the algorithm for a degree p_s finite element simulation using our p-multigrid approach. In the precomputation phase the initial stiffness matrices ${}^0\mathbf{K}_{\mathbb{T}}^p$ are determined for all polynomial degrees p (see Section 3). This requires the derivatives of the barycentric coordinates $\frac{\partial \lambda_i^{\mathbb{T}}}{\partial \mathbf{x}}$ that carry the geometric information of each tetrahedron. These can be reused for all polynomial levels as the discretization is performed on the same tetrahedral mesh. Additionally, the global mass matrix is set up for all polynomial degrees during precomputation, as it remains constant throughout the simulation.

To set up the linear system all stiffness matrices on the hierarchy must be updated. Therefore, each element's rotation is determined by polar decomposition, the element stiffness matrices are updated and the global stiffness matrix is assembled. Similar to Bargteil et al. [BC14], we only use the linear part of the deformation gradient to compute an element's rotation. Therefore, the rotation matrices are independent of the polynomial degree and can be used on all levels of the hierarchy. In comparison to other approaches that directly solve the system on the highest level, our p-multigrid method additionally requires an update of the system and stiffness matrices on all lower levels of the hierarchy. We directly discretize the governing partial differential equations with the corresponding polynomial degrees for the respective level. This avoids the computation of expensive SpMM operations in every time step, which would be required for computing the Galerkin coarse grid operator. The system matrix is then constructed using the precomputed mass matrix and the assembled stiffness matrix. To complete the set up for the linear system, external forces must be determined to compute the right hand side. Note that this is only required for the highest level of the multigrid hierarchy.

Efficient prolongation: The prolongation operator $\mathbf{I}_{l+1}^{l}$ for transforming polynomials from degree p to $p+1$ can be computed per tetrahedral element using the degree elevation algorithm in Eq. (8). However, one can do this more efficiently by taking the C^0-continuity of finite elements into account, i.e., some degrees of freedom are shared with adjacent elements. We illustrate this using an example of a prolongation operator applied to a 2D quadratic finite element as depicted in Fig. 3, where the values b_{ijk}^p of the polynomial are associated to the nodes ξ_{ijk}. Additionally, we make use of the fact that the polynomials in B-form on a subsimplex, e.g., a vertex, an edge or triangle, can be considered in isolation. In this configuration some barycentric coordinates are zero and the others vary on the subsimplex and sum to one. For our case the degree elevation can be computed locally on one subsimplex and we can ignore the other values on the finite element. Specifically for cubic elements, we first compute the values at the vertices, then at the edges and finally on the triangles.

In the example shown in Fig. 3 we first transfer the values associated with the vertices of the tetrahedral mesh directly, e.g., $b_{300}^3 = b_{200}^2$, and only once per vertex. Note that the superscripts denote polynomial degrees instead of an exponent in this context. We then locally perform a degree elevation for every edge in the mesh over a 1-dimensional parametric space. In the example the values along the edge $(\mathbf{v}_0, \mathbf{v}_2)$, i.e., $b_{200}^2, b_{101}^2, b_{002}^2$ can be interpreted as an one-dimensional quadratic Bézier polynomial by simply omitting the second index: $q = b_{20}^2 B_{20}^2(\lambda) + b_{11}^2 B_{11}(\lambda) + b_{02}^2 B_{02}^2(\lambda)$. Then the standard degree elevation for Bézier curves [Far02] can be applied

$$b_{12}^3 = \frac{1}{3}(b_{20}^2 + 2b_{11}^2), \quad b_{21}^3 = \frac{1}{3}(2b_{11}^2 + b_{02}^2), \qquad (12)$$

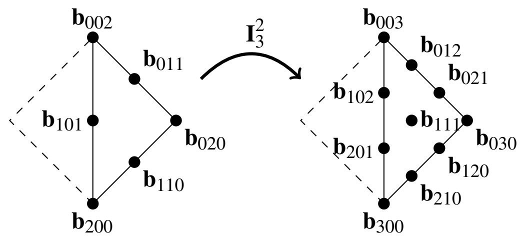

Figure 3: *Prolongation operator transferring a field from a quadratic finite element to to a cubic finite element by degree elevation. Note that the degree elevation for edge* $(\mathbf{v}_0, \mathbf{v}_2)$ *can be reused for both adjacent triangles.*

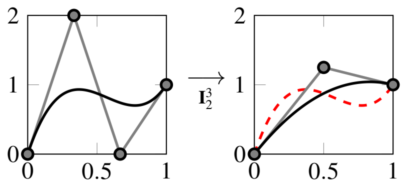

Figure 4: *Restriction from cubic to quadratic form uses a cubic polynomial which interpolates the value of the quadratic polynomial at* $t = 0.5$.

where we explicitly omit the computation of b^3_{30} and b^3_{03}. Finally, the remaining value b^3_{111} at the triangle center is obtained by applying the degree elevation algorithm for Bézier triangles:

$$b^3_{111} = \frac{1}{3}(b^2_{110} + b^2_{101} + b^2_{011})$$

For volumetric meshes, the degree-elevated values on the edges and the triangles, respectively, can be reused for adjacent tetrahedra.

Efficient restriction: In the case of quadratic and linear polynomials the restriction operator is rather simple. One can simply omit the additional degrees of freedom on the edges, as the vertices are specified directly and no other degrees of freedom exist. For cubic and quadratic polynomials, vertices and edges can be treated separately as done for prolongation. Taking Eq.(12) and substituting the vertex values b^2_{20} and b^2_{02} by b^3_{30} and b^3_{03} respectively, one obtains two equations for b^2_{11}. Taking the average of these values results in

$$b^2_{11} = \frac{b^3_{30} + 3b^3_{21} + 3b^3_{12} + b^3_{03}}{4}$$

and is equivalent to computing $\mathbf{P}'\mathbf{b}'^{p+1}$ and also to computing a quadratic form interpolating the cubic form at $\lambda_0 = \lambda_1 = 0.5$ as shown in Fig. 4.

6. Results

We now show some examples and analyze our p-multigrid method in terms of performance and accuracy. We used an Intel Core i7 with 3.4 Ghz for our tests and implemented all building blocks utilizing only one of the processor's cores. Table 1 lists the models for our tests together with the characteristic numbers, i.e., the polynomial degree, the number of degrees of freedom and number of non-zero entries of the sparse matrix.

Examples: The accompanying video and the snapshots in Fig. 1 show different scenarios with quadratic and cubic finite elements using our p-multigrid algorithm to solve the linear system. In all cases the simulation mesh is coupled with a high-resolution surface mesh for rendering. Note that we have not implemented self collision which may be apparent in the videos.

Validation: To validate our cubic finite elements we performed a static analysis of a beam fixed on one side deforming under an applied load. We chose a beam with dimensions $(l \times w \times h) = (1.0\text{m} \times 0.2\text{m} \times 0.2\text{m})$ and an elastic modulus of $E = 500$ kPa. By applying a force of $F = 10$ N at the position in the middle of the free end of the beam, it deforms and we measure the vertical displacement. For the numerical computation a model consisting of five cubes subdivided in 24 tetrahedra each was constructed that simulated with linear, quadratic and cubic finite elements. With a deflection of approximately 0.03 m for linear, 0.048 m for quadratic and 0.05 m for cubic finite elements one can observe convergence when raising the polynomial degree. We compared the computed deflections with an analytical result from beam theory according to the textbook [GHSW07], which is only valid for small deformations. For this model the moment of inertia for the cross-sectional area is $I = \frac{1}{7500}\text{m}^4$. The analytical maximum deflection in vertical direction is then determined by $w_{\max} = \frac{Fl^3}{3EI} = 0.05$ m, which is very close to our computation and only varies at the fifth decimal place. This shows that cubic finite elements can better approximate the solution of the partial differential equation. However, we expect that a further degree raising of the basis functions will not pay off in terms of accuracy.

Effect of approximate solve: In order to speed up the simulation and achieve constant computation times, many authors (see, e.g., [NPF05, ACF11, WBS*13]) propose using a small, fixed number of 20 CG iterations to solve the linear system. This choice leads to artificial damping, as some residual forces remain that are not converted into momentum. The following experiment demonstrates this effect. A bar is fixed on one side and initialized to a deflected state. As the deflected side is not fixed, the simulation then causes the bar to oscillate. As no explicit damping terms are applied, the bar should oscillate without energy loss and the amplitude of the oscillation should remain constant. Fig. 7 shows the amplitude over time for a CG algorithm with a fixed number

Model	degree	# tets	# vertices	# nodes	# dofs	# non-zeros	acceleration
Bunny	2	1691	471	2957	8871	652527	1.72
Gargoyle	2	17040	4490	29218	87654	6544566	2.48
Homer	3	5151	1513	28945	86835	11005101	3.15
Frog	3	7746	2099	42199	126597	16359903	2.74

Table 1: *Name, number of tetrahedral elements, number of vertices, number of finite element nodes of the models used for the deformation simulation.*

of 20 iterations in comparison to a p-multigrid solver with a prescribed residual reduction of 10^{-3}. One can clearly see that the simulation with accurate solves exhibits significantly reduced damping. The remaining damping is caused by the implicit integration scheme used and can be reduced by decreasing the time or further increasing the iterations. However, a future extension of our work could adopt an energy-preserving variational integrator [KYT*06] to further reduce damping. Much like Mullen *et al.* [MCP*09] argument for numerical dissipation in fluid simulations, reducing artificial numerical damping to a minimum is very important to put desired damping under artist control.

Performance: In order to analyze the performance of our p-multigrid solver we monitor the convergence, i.e., the residuals $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ for the i-th iteration. The relative residual reduction ($\|\mathbf{r}_i\|_2 / \|\mathbf{r}_0\|_2$) w.r.t. time of our p-multigrid method is then compared with a preconditioned CG algorithm and a direct sparse solver (a Cholesky factorization method [GJ*10]). For the tests the models in Table 1 are compressed along the y-axis and released afterwards. The diagrams in Figure 5 show the relative residual reduction for the model *Gargoyle*. As the direct method finishes without any intermediate results, the graph shows a vertical line. In contrast to the CG algorithm, the residuals of the multigrid algorithm are smooth in the logarithmic plot making it more predictive. The last column in Table 1 denotes acceleration when using our p-multigrid instead of a CG solver. We additionally performed the tests for even larger model sizes. Fig. 6 shows a plot of the acceleration over the of number of nodes for quadratic finite elements. This is the expected behavior as for large models the multigrid algorithm starts to pay off.

The *frog* scenario in Fig. 1 with cubic finite elements (see Table 1 for the mesh complexity) needs two to three V-cycles to reduce the residuum by three order of magnitude. A single V-cycle in the p-multigrid algorithm takes approximately 235ms on a single core implementation for this example. This splits in approximately 130ms and 72ms for smoothing (each with five Gauss-Seidel iterations) on the cubic and quadratic level, respectively, where the remaining time is needed for restriction, prolongation and the exact solve. Note that the exact solution takes only 3ms.

In addition to the computation for solving the linear system, the hierarchy of matrices must be updated. However,

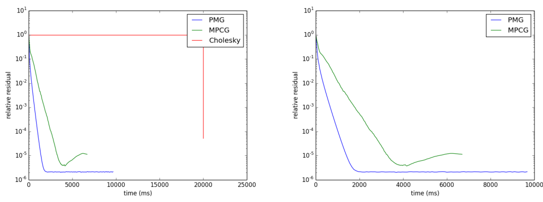

Figure 5: *The diagrams shows a logarithmic plot of the residual over time for our multigrid approach, a preconditioned conjugate gradient solver and a direct solver with Cholesky factorization. In the right diagram the Cholesky factorization is omitted to better see the convergence behavior. The multigrid solver reaches a residual of* 10^{-3} *after* 7 *iterations. The CG solver requires* 135 *iterations.*

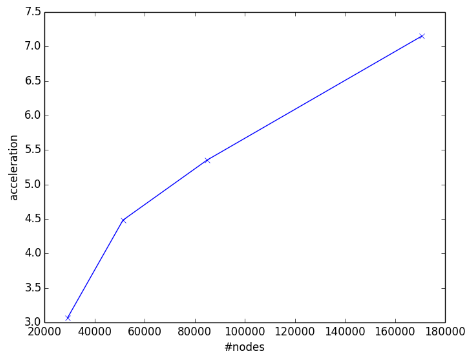

Figure 6: *This diagram shows the acceleration of using our p-multigrid solver instead of a CG method with preconditioning for quadratic finite elements . The x-axis denotes the number of nodes of the simulation. This demonstrates that multigrid algorithms pay off at large matrix sizes.*

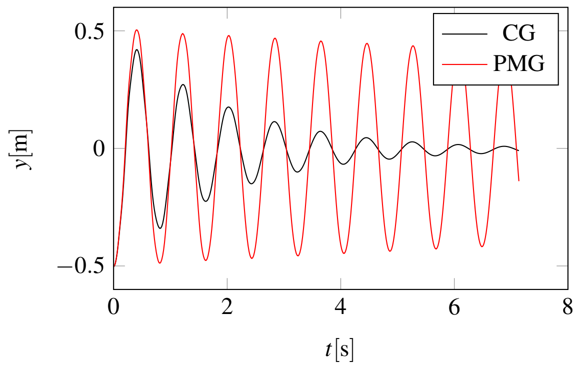

Figure 7: *Comparison of the deflection over time of an oscillating bar using 20 CG iterations or solving accurately using our method.*

this additional overhead compared to non-multigrid methods is compensated due to the improved convergence rate. A fundamental difference besides our polynomial hierarchy and an algorithmic multigrid computing the matrices on linear shape functions (e.g., Georgii et al. [GW05]) is the update of the matrix hierarchy. Whereas their approach requires recomputation of the matrices on each level by two expensive sparse matrix products mutliplying the restriction and prolongation operators with the system matrix, our approach can directly use the matrices for different polynomial degrees. Even in an optimized version, Georgii et al. [GW08] report that the matrix update takes more time than the multigrid algorithm itself. In our case the matrix construction only takes approximately 1% for quadratic finite elements and up to 15% for cubic finite elements of the time required for the solve. The higher percentage for cubic elements is due to the fact that additionally a linear system for quadratic finite elements must be set up. Furthermore, it is worth to note that a geometric multigrid approach as proposed by Georgii et al. has not yet been developed for higher order finite elements.

7. Conclusion

In this paper we have presented a p-multigrid approach for efficiently solving sparse linear systems arising from higher-order finite element discretizations. Due to the direct discretization of the problem with different polynomial degrees on the same tetrahedral mesh, updates of the matrix hierarchy do not have to be computed by expensive SpMM operations. Furthermore, we demonstrated the use of cubic finite elements for simulating volumetric deformation.

As future work it would be very interesting to investigate the exact co-rotational method for quadratic and cubic finite elements. In this approach the dependency of the rotation matrix on the vertex positions is considered when deriving the stiffness matrix from the elastic forces. Together with a variational integrator, we would expect a better energy conservation as reported by Chao et al. [CPSS10]. Furthermore, for large models it would be interesting to investigate if a geometric multigrid approach on our lowest level would be beneficial for even higher convergence. Parallelizing all of the building blocks of our multigrid algorithm on GPUs and comparing the speedup to the one achieved in our previous work [WBS*13] could make our algorithm viable for interactive use. We also expect GPU performance benefits comparable to the multigrid work by Dick et al. [DGW11a] and [DGW11b].

Acknowledgments

The work of the authors has been supported by the EU-projects VISTRA (FP7-2011-NMP-ICT-FoF-285176) and CloudFlow (FP7-2013-NMP-ICT-FoF-609100). The geometry models used in this work are provided by Stanford Shape Repository (bunny) Aim@Shape. The tetrahedral meshes were generated with CGAL library.

References

[ACF11] Allard J., Courtecuisse H., Faure F.: Implicit FEM solver on GPU for interactive deformation simulation. In *GPU Computing Gems Jade Edition*. NVIDIA/Elsevier, Sept. 2011, ch. 21. 7

[BC14] Bargteil A. W., Cohen E.: Animation of deformable bodies with quadratic bézier finite elements. *ACM Trans. Graph. 33*, 3 (June 2014), 27:1–27:10. 2, 4, 6

[BHM00] Briggs W. L., Henson V. E., McCormick S. F.: *A multigrid tutorial (2nd ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 3, 5

[BMOT13] Bender J., Müller M., Otaduy M. A., Teschner M.: Position-based methods for the simulation of solid objects in computer graphics. In *EUROGRAPHICS 2013 State of the Art Reports* (2013), Eurographics Association. 1, 2

[BW98] Baraff D., Witkin A.: Large steps in cloth simulation. *Computer Graphics 32* (1998), 43–54. 2, 4

[CPSS10] Chao I., Pinkall U., Sanan P., Schröder P.: A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (July 2010), 38:1–38:6. 2, 9

[DGW11a] Dick C., Georgii J., Westermann R.: A hexahedral multigrid approach for simulating cuts in deformable objects. *Visualization and Computer Graphics, IEEE Transactions on 17*, 11 (2011), 1663–1675. 3, 9

[DGW11b] Dick C., Georgii J., Westermann R.: A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice and Theory 19*, 2 (2011), 801–816. 3, 9

[Far02] Farin G.: *Curves and surfaces for CAGD: A practical guide*. MK Publishers Inc., 2002. 5, 6

[GHSW07] Gross D., Hauger W., Schröder J., Wall W.: *Technische Mechanik 2 – Elastostatik*. Springer–Verlag, 2007. 7

[GJ*10] Guennebaud G., Jacob B., et al.: Eigen v3. http://eigen.tuxfamily.org, 2010. 8

[GW05] Georgii J., Westermann R.: A multigrid framework for real-time simulation of deformable volumes. In *Proceedings of the 2nd Workshop On Virtual Reality Interaction and Physical Simulation* (2005), pp. 50–57. 3, 9

[GW08] GEORGII J., WESTERMANN R.: Corotated finite elements made fast and stable. In *VRIPHYS* (2008), pp. 11–19. 3, 9

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), SCA '04, Eurographics Association, pp. 131–140. 2

[KMBG08] KAUFMANN P., MARTIN S., BOTSCH M., GROSS M.: Flexible simulation of deformable models using discontinuous galerkin fem. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2008), SCA '08, Eurographics Association, pp. 105–115. 2, 4

[KYT*06] KHAREVYCH L., YANG W., TONG Y., KANSO E., MARSDEN J. E., SCHRÖDER P., DESBRUN M.: Geometric, variational integrators for computer animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), SCA '06, Eurographics Association, pp. 43–51. 8

[LBOK13] LIU T., BARGTEIL A. W., O'BRIEN J. F., KAVAN L.: Fast simulation of mass-spring systems. *ACM Transactions on Graphics 32*, 6 (Nov. 2013), 209:1–7. Proceedings of ACM SIGGRAPH Asia 2013, Hong Kong. 2

[MCP*09] MULLEN P., CRANE K., PAVLOV D., TONG Y., DESBRUN M.: Energy-preserving integrators for fluid animation. *ACM Trans. Graph. 28*, 3 (July 2009), 38:1–38:8. 8

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proceedings of Graphics Interface 2004* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004), GI '04, Canadian Human-Computer Communications Society, pp. 239–246. 1, 2

[MS06] MEZGER J., STRASSER W.: Interactive soft object simulation with quadratic finite elements. In *Proceedings of the 4th international conference on Articulated Motion and Deformable Objects* (Berlin, Heidelberg, 2006), AMDO'06, Springer-Verlag, pp. 434–443. 2

[MSW14] MICHELS D. L., SOBOTTKA G. A., WEBER A. G.: Exponential integrators for stiff elastodynamic problems. *ACM Trans. Graph. 33*, 1 (Feb. 2014), 7:1–7:20. 2

[MTPS08] MEZGER J., THOMASZEWSKI B., PABST S., STRASSER W.: Interactive physically-based shape editing. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling* (New York, NY, USA, 2008), SPM '08, ACM, pp. 79–89. 2

[NMK*06] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4 (2006), 809–836. 2

[NPF05] NESME M., PAYAN Y., FAURE F.: Efficient, Physically Plausible Finite Elements. In *Eurographics* (Dublin, Irlande, 2005). 7

[OH99] O'BRIEN J. F., HODGINS J. K.: Graphical modeling and animation of brittle fracture. In *SIGGRAPH 1999* (1999), pp. 137–146. 2

[PO09] PARKER E. G., O'BRIEN J. F.: Real-time deformation and fracture in a game environment. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 165–175. 2

[SB12] SIFAKIS E., BARBIC J.: FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses* (2012), ACM, pp. 20:1–20:50. 4

[Sch07] SCHUMAKER L.: *Spline Functions on Triangulations*. Cambridge University Press, 2007. 3, 4, 5

[SSX06] SHU S., SUN D., XU J.: An algebraic multigrid method for higher-order finite element discretizations. *Computing 77*, 4 (2006), 347–377. 3

[TS01] TROTTENBERG U., SCHULLER A.: *Multigrid*. Academic Press, Inc., Orlando, FL, USA, 2001. 3, 5

[WBS*13] WEBER D., BENDER J., SCHNOES M., STORK A., FELLNER D.: Efficient gpu data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1 (2013), 16–26. 2, 7, 9

[WKS*11] WEBER D., KALBE T., STORK A., FELLNER D., GOESELE M.: Interactive deformable models with quadratic bases in Bernstein-Bézier-form. *TVC 27* (2011), 473–483. 2, 3, 4

[ZSTB10] ZHU Y., SIFAKIS E., TERAN J., BRANDT A.: An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph. 29*, 2 (Apr. 2010), 16:1–16:18. 3

[ZT00] ZIENCKIEWICZ O. C., TAYLOR R. L.: *The Finite Element Method*. Butterworth-Heinemann, 2000. 2