A Hexahedral Multigrid Approach for Simulating Cuts in Deformable Objects

Christian Dick, Joachim Georgii, and Rüdiger Westermann

Abstract—We present a hexahedral finite element method for simulating cuts in deformable bodies using the corotational formulation of strain at high computational efficiency. Key to our approach is a novel embedding of adaptive element refinements and topological changes of the simulation grid into a geometric multigrid solver. Starting with a coarse hexahedral simulation grid, this grid is adaptively refined at the surface of a cutting tool until a finest resolution level, and the cut is modeled by separating elements along the cell faces at this level. To represent the induced discontinuities on successive multigrid levels, the affected coarse grid cells are duplicated and the resulting connectivity components are distributed to either side of the cut. Drawing upon recent work on octree and multigrid schemes for the numerical solution of partial differential equations, we develop efficient algorithms for updating the systems of equations of the adaptive finite element discretization and the multigrid hierarchy. To construct a surface that accurately aligns with the cuts, we adapt the splitting cubes algorithm to the specific linked voxel representation of the simulation domain we use. The paper is completed by a convergence analysis of the finite element solver and a performance comparison to alternative numerical solution methods. These investigations show that our approach offers high computational efficiency and physical accuracy, and that it enables cutting of deformable bodies at very high resolutions.

Index Terms—Deformable objects, cutting, finite elements, multigrid, octree meshes.

1 Introduction

In this work we propose a method for realistically simulating complicated cuts in linear elastic deformable objects. Our approach is different from previous approaches in that it does not treat the cutting procedure and the numerical solution scheme independently, but intertwines both procedures in a way that enables high computational efficiency. We achieve this by developing a novel embedding of adaptive finite element refinements and topological changes of the simulation grid into a geometric multigrid method [1], [2], [3]. Adaptivity enables representing complicated cuts at very high resolution, and the multigrid method achieves optimal computational complexity that is linear in the number of simulation elements. Figure 1 shows some cuts that have been performed using our approach.

Underlying the basic multigrid idea is the principle of coupling multiple scales, for instance, by using a geometric model hierarchy equipped with transfer operators to propagate quantities across the scales. The use of such a hierarchy, in general, imposes performance limitations when using multigrid schemes in combination with cutting schemes based on tetrahedral [4], [5], [6] or polyhedral [7] finite elements. Since element subdivision generates unstructured meshes in general, there are no canonical coarse versions of the mesh and the construction of a geometric model hierarchy becomes very complicated. Due to this reason, incorporating subdivision based cutting into geometric multigrid schemes has not yet been considered.

Instead of explicitly modeling the boundary induced by a cut in the finite element discretization, this boundary can also be incorporated into the basis functions of the finite dimensional solution spaces [8], [9]. This enables using a coarse simulation grid that does not depend on the shape of the object. The same principle is underlying the extended finite element method (XFEM) [10], which enriches the finite element spaces by employing additional step functions to represent material discontinuities. XFEM has mainly been used to accurately simulate material interfaces and crack propagation [11], [12], and just recently its potential for cutting and fracturing deformable objects in graphics applications has been recognized [13], [14], [15].

Since the XFEM method uses a static simulation grid for which a hierarchy can be constructed in a pre-process, its embedding into a multigrid solver is possible in principle. However, the modeling of high resolution cuts, for instance via enrichment textures [15], requires large systems of equations to be solved, and it comes at the expense of increasing the computational cost of element integration.

2 Our Contribution

We propose a novel algorithm for physics based cutting of linear elastic deformable bodies using hexahedral finite elements. Simulation elements that are

• *C. Dick, J. Georgii, and R. Westermann are with Computer Graphics and Visualization Group, Technische Universität München, Germany. E-mail: {dick, georgii, westermann}@tum.de*

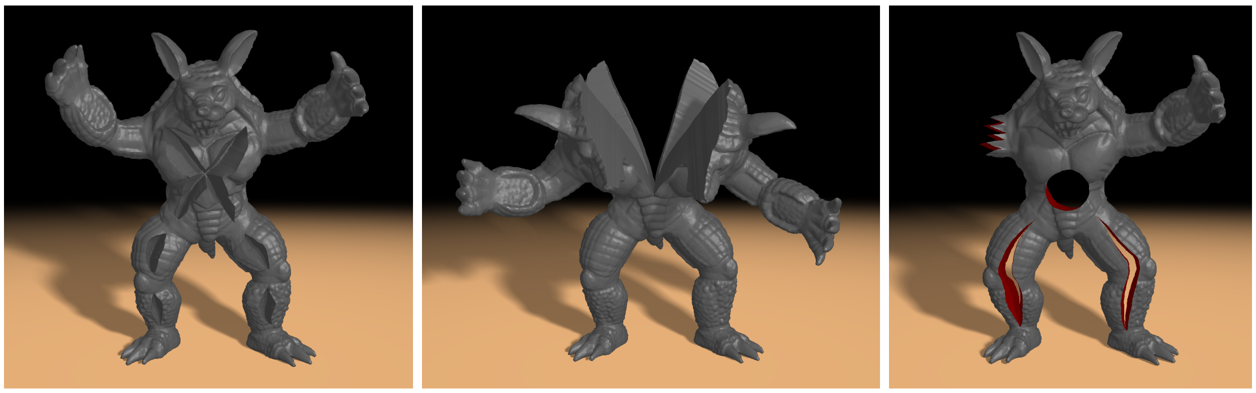

Fig. 1. Cuts in the Stanford Armadillo model. An adaptive finite hexahedra model consisting of 493K simulation elements is used. Adaptive refinements of the simulation mesh along the cuts result in 76K, 150K, and 153K additional elements, from left to right. Cutting and simulation is performed at 9 to 11 seconds per time step.

touched by the cutting tool are recursively subdivided using a regular octree refinement. This results in an adaptive finite hexahedron approximation of the cut object. An example is given in Figure 2.

The octree is refined until a sufficient approximation is reached, and on this subdivision level cutting is performed along the element faces. The adaptive refinement allows arbitrarily thin and complicated structures to be sliced, and it can be employed to adapt the octree to material jumps in heterogeneous materials. Examples demonstrating these possibilities are shown in Figures 13 and 14, respectively. Since all elements have the same shape the method only requires scaled instances of one local stiffness matrix to accommodate whatever deformation is applied. Thus, it has small storage requirements and only needs to consider a few cases in the construction and update of the multigrid hierarchy.

The deformable object has to be discretized on the adaptive octree that is generated by the cutting algorithm. Since none of the previous multigrid approaches considers cuts of the simulation grid, we propose a novel algorithm to embed the induced discontinuities in the geometric multigrid hierarchy, including fast algorithms for updating the resulting systems of equations. On the coarse resolution levels the algorithm duplicates respective cells and distributes the per-cell equations to the duplicates according to the separated material components. To represent the cuts on the coarse grid levels where the structure of the object boundary is not resolved adequately, we present a new boundary treatment method which is based on the principle of potential energy minimization underlying the variational formulation of elasticity problems. To reflect that cells on the coarser grids might be only partially filled with material, this method computes the potential energy always with respect to the finest grid. We present a detailed convergence analysis of our solver and a thorough comparison to alternative solution methods.

Rendering a smooth polygon surface that aligns with the cuts is very difficult since it can undergo complicated topological changes. In particular, a render surface that is bound to the initial simulation grid as proposed in [16], [17], [18], [19] cannot easily be employed for this purpose. Due to this reason we use the splitting cubes algorithm [20] on a dual grid to compute a watertight boundary surface directly from the 3D simulation grid. By using for each triangle vertex either a vertex or a face normal, high quality rendering is achieved.

To clearly focus our work on the efficient cutting and simulation of a finite element model, throughout this paper collision detection and response is not considered. Please note that this is *not* a limitation of the presented approach. In fact, any state of the art collision handling algorithm which uses surface meshes for collision detection and external forces for collision response could be directly integrated.

The remainder of this paper is as follows: In the next section we review work that is related to ours. Then we describe the cutting algorithm from a purely geometric perspective. In Section 5 we review the fundamentals of the corotated finite element method which we use in our work. Section 6 discusses the embedding of adaptive finite element refinements and topological changes of the simulation grid into the multigrid scheme. Section 7 presents the specific adaptations to the splitting cubes algorithm to reconstruct an accurate boundary surface from the simulation grid. A detailed analysis of the proposed algorithm, both with respect to performance and convergence is given in Section 8. We conclude the paper with a comparison to alternative approaches and some ideas on future challenges in the field.

3 Related Work

Starting with the seminal work of Terzopoulos and co-workers [21], [22], physics based methods for simulating deformable models have been researched exten-

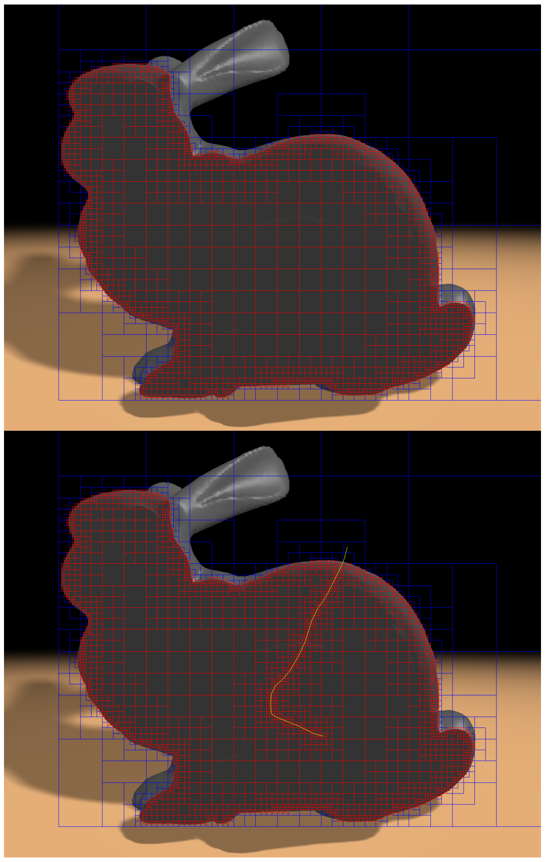
Fig. 2. Cross-section through an adaptive octree grid without (top) and with (bottom) a cut.

sively in computer graphics for the last two decades. A good overview of the multitude of methods for realistically simulating deformable bodies can be found in [23]. For example, boundary element models [24], adaptive and multiresolution approaches [25], [26], [27], grid-less techniques [28], [29], and finite element methods [30], [31] have been proposed. The simulation of brittle fracture based on finite elements was described by O'Brien and Hodgins [32] and later extended to ductile fracture [33]. [34] proposed a composite element formulation that considers varying material properties within a coarse element.

Tetrahedral subdivision methods for cutting deformable objects were introduced in [4], [6]. To reduce the number of ill-shaped elements, Nienhuys and van der Stappen proposed cutting along the element faces [35]. Cotin et al. and Forest et al. deleted elements that were cut [36], [37]. Smooth cuts that also reduce the number of ill-shaped elements were achieved by adaptively aligning mesh edges and faces with the cutting surface [38], [39], [40]. By restricting subdivisions to a few refinement patterns [5], [41] the number of additional simulation elements caused by a cut can be reduced. A multi-resolution approach for this method was presented by Ganovelli et al. [42]. The virtual node algorithm [43] avoids ill-shaped elements by duplicating simulation elements and re-assigning material components on both sides of a cut.

Wicke et al. and Kaufmann et al. introduced polyhedral subdivision [7], [44], which splits initial tetrahedra into polyhedra and then subdivides these elements further. Extended finite element methods [10] enrich a finite element model with specific basis functions to capture discontinuities in the simulation elements. The use of XFEM for virtual surgery simulation [13], [14] and cutting in 2D thin shells [15] has been demonstrated. Sifakis et al. clipped a high-resolution material boundary surface mesh against a coarse simulation mesh to consider fine material components in a coarse elasticity simulation [45].

Octree-based physical simulation of fluids and gases was shown in [46], [47], [48]. Both restricted and unrestricted octrees were used. To achieve high resolution of small scale details, one focus was on deriving adaptive finite difference discretizations of the governing equations. Finite element discretizations for the numerical solution of partial differential equations on restricted octrees were introduced in [49], [50].

Multigrid approaches have recently gained much attention in the computer graphics community due to their optimal computational complexity. The applications range from fluid simulation [51] and deformable body simulation [52], [53], [54] to image processing [55] and texture synthesis [56]. Interactive multigrid approaches for simulating linear elastic materials on hexahedral grids were presented in [57], [58]. Since it is well known that the multigrid convergence is lowered in case of complicated material boundaries, [58] proposed a numerical boundary smoother for finite difference schemes. An adaptation of the basis functions on the coarser levels to more accurately represent the covered boundary was proposed in [59], [60], [61].

4 Cutting Algorithm

To enable the efficient embedding of the cutting algorithm into a geometric multigrid scheme, we avoid any unstructured grid refinement and instead cut along the element faces in a hexahedral simulation grid that adaptively refines along the cut. This results in an adaptive octree grid. The octree's leaf cells represent the simulation elements and store the corresponding vertices. At the cells on the finest level we store links that are marked as connected or disconnected depending on whether the corresponding elements have been detached by the cut. The links that are cut by the object's boundary are also marked as disconnected. Each octree cell is equipped with memory references to its child cells, its parent cell, and the neighboring cells on the same level.

For the sake of simplicity, we first describe the cutting algorithm in an uniform grid before we introduce the extensions that are necessary to perform a cut in an adaptive octree grid.

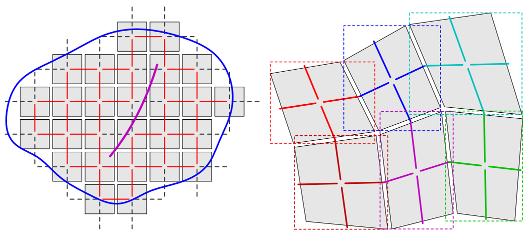
Fig. 3. Left: 2D illustration of the linked volume representation, consisting of a set of finite elements (gray) which are connected via links (red). These links are disconnected (dashed, gray) at the object boundary (blue) and when cut by a cutting surface (violet). Right: Bounding boxes (dashed) of deformed finite elements used to accelerate the link/blade intersection test. The links are deformed according to the (tri-linear) deformation of the elements. Each element's bounding box covers half of each link emanating from this element.

4.1 Cuts - Uniform Grid

In the following we assume that the object to be cut has been discretized into a uniform hexahedral (voxel) grid. Discretization means building a binary representation, where every voxel is classified as inside or outside depending on whether its center is in the interior of the object or not. Inside voxels represent the finite elements that are used in the physics based simulation.

We leverage a linked volume representation [62] in which the centers of face adjacent elements are connected via links. Initially, each 3D element has six links, and the links which are intersected by the surface of the object are marked as disconnected. Cutting the FE-model is performed by disconnecting the links that are cut by the cutting blade (see Figure 3 (left) for an illustration).

The blade is realized as a triangle mesh, which is built from consecutive cut-lines that are induced via a cutting tool. To find the links that are cut, all links in the vicinity of the cutting front, i.e., the surface spanned by the current and the last cut-line, are tested for an intersection with the triangles representing this front.

Since the intersection test has to be performed in the deformed object state, in general the cutting blade needs to be tested against all connected links. To prune as many links as possible before testing against the blade, the blade's axis-aligned bounding box is tested first. In addition, every finite element stores its axis-aligned bounding box (see Figure 3 (right)), and a bounding box hierarchy is created to prune needless tests. By using this information the broad phase intersection test reduces to simple bounding box tests, and only a few link-triangle tests are required in the narrow phase.

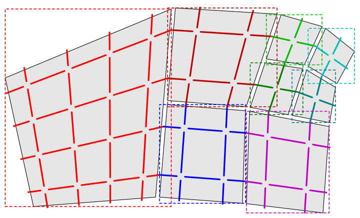
Fig. 4. Illustration of the bounding boxes (dashed) of the octree finite elements (gray) used in the link/blade intersection test.

4.2 Cuts - Octree Grid

In order to avoid using a uniform finite element representation at the finest resolution level, we employ an adaptive object discretization, i.e., an octree grid, where the octree's leaf cells represent the finite elements. An adaptive representation allows using the fine level simulation elements at the locations where they are needed to resolve the object boundary accurately. Thus, it can reduce the number of simulation elements significantly, and therefore improves the performance of the finite element simulation.

The octree grid is built from an initial uniform object discretization in a coarse-to-fine procedure. The resolution of this discretization is chosen such that it can adequately model the physical behavior of the object's inner part. In an interactive application, it can be set to a resolution that allows simulating the deforming body at reasonable speed.

Starting with this discretization, at each level the cells containing at least one link that would be cut by the object boundary are refined regularly into 2^3 smaller cells. Finally, all cells that do not contain at least one finest level cell center that is in the object's interior are deleted from the octree structure. Note that the cell centers at the finest level, and thus also the link positions, can be computed without that a cell has to be refined explicitly down to the finest resolution.

The octree refines adaptively along the object's boundary while it models the object's interior away from the boundary at the selected coarser resolution. It is restricted to not have level differences between adjacent elements—sharing a vertex, an edge, or a face—of more than one (see Figure 2). In this way abrupt changes in the structural material representation are avoided.

Cutting is performed by traversing the octree and performing a regular 1:8 split of the leaf cells to be refined. The refinement criterion is the same as is used to initially refine the octree grid along the object boundary. On the finest level no split is performed but the links that are cut are disconnected. To accelerate the intersection test between the blade and the elements, each element stores its axis-aligned bounding box. Figure 4 illustrates the bounding boxes for adjacent elements at different resolution levels.

The refinement procedure creates additional elements, which number increases proportional to the object's surface increase. Upon refining an element, the octree is updated to ensure only level transitions of at most one. The bounding boxes of new elements are computed on-the-fly.

5 Finite Element Discretization

We follow the variational formulation of linear elasticity, which is based on the principle of potential energy minimization. By using a finite element discretization of the displacement field—we use hexahedral finite elements and tri-linear shape functions—the static behavior of the object is described by a linear system of equations $Ku = f$. Here, K denotes the stiffness matrix, and u and f are the linearized vertex displacements and external forces exerting at these vertices, respectively.

The linear system is assembled from the element equations $K_e u_e = f_e$. The element stiffness matrix K_e is computed by $K_e = \int_{\Omega_e} B^{\mathrm{T}} D B \; \mathrm{d}x$. Ω_e is the domain of the finite element, B is the Cauchy strain matrix, and D is the linear material law. A detailed explanation of the numerical finite element scheme to solve linear elasticity problems can be found in [63].

We employ a rotational invariant formulation of the Cauchy strain tensor using the corotated strain of linear elasticity [64]. In this formulation finite elements are first rotated into their reference configuration before the strain is computed. Although the strain is still approximated linearly, artificial forces as they occur when using the Cauchy strain are reduced significantly.

The current element rotations R_e are determined from the current element deformations. We follow the approach proposed by Georgii and Westermann [65], which seeks to minimize the distance between the rotated deformed and the undeformed configuration. The 'energy' E_e measures this distance as

$$\begin{aligned} E_e &= \int_{\Omega_e} d^{\mathrm{T}} d \, \mathrm{d}x, \\ d &= R_e^{\mathrm{T}}(x + u - c) - (x - c_0), \end{aligned}$$

where c and c_0 are the centers of mass of the deformed and undeformed element, respectively. The rotation that results in the minimum energy is computed using a quaternion formulation [66]. Once the element rotations have been determined, the element equations become

$$[R_e] K_e \left([R_e]^{\mathrm{T}} (x_e + u_e) - x_e \right) = f_e .$$

$[R_e]$ is a 24×24 matrix with 8 instances of R_e on the diagonal, and x_e is a vector containing the element's vertex positions in the undeformed state. Rewriting the equations for u_e allows assembling the element contributions into one linear equation system $\hat{A}u = \hat{b}$.

The dynamic behavior of a deformable object is described by the Lagrangian equation of motion

$$M\ddot{u} + C\dot{u} + \hat{A}u = \hat{b},$$

where M and C denote the mass and damping matrix, respectively. Time discretization using the classical Newmark integration method [63] yields the linear equation system $Au = b$. This system needs to be solved for the unknown displacements u in every time integration step.

6 Discontinuous Multigrid Solver

In the following we describe the design and implementation of the geometric multigrid approach for solving the equations of deformable body motions. The solver is specifically tailored to an equation system that changes in every time step due to topological changes of the simulation grid and the use of the corotated strain formulation.

6.1 Geometric Multigrid

Basic iterative methods like Jacobi or Gauss-Seidel relaxation effectively reduce high-frequency (oscillatory) error components, but they are ineffective in reducing low-frequency (smooth) components, which typically causes the error reduction to stall after a few iteration steps.

Consider the linear equation system $Au = b$ with exact solution u. Let $\tilde{u}$ be the current approximate solution and $r = b - A\tilde{u}$ the current residual. When the error reduction stalls, the current error e determined by the residual equation $Ae = r$ is smooth. The basic idea of multigrid is to solve the residual equation on a coarser grid, since there the error appears more oscillatory and the basic relaxation methods are more effective. This leads to the so-called *two-grid correction scheme*.

In the following, superscripts h and $2h$ denote the respective grid spacings and are used to refer to the original and the coarser grid. The main components of the two-grid correction scheme are a coarse grid version of A^h, denoted A^{2h}, a transfer operator R_h^{2h} to restrict the residual to the coarser grid, and a transfer operator I_{2h}^h to interpolate the error from the coarser grid. Let $\tilde{u}^h$ be the current approximate solution. The two-grid correction scheme then consists of the following steps:

1) Relax $A^h \tilde{u}^h \approx b^h$ (n_1 times).
2) Compute residual $r^h = b^h - A^h \tilde{u}^h$.
3) Restrict residual: $r^{2h} = R_h^{2h} r^h$.
4) Solve residual equation $A^{2h} e^{2h} = r^{2h}$.
5) Interpolate error: $\tilde{e}^h = I_{2h}^h e^{2h}$.
6) Apply coarse grid correction: $\tilde{u}^h \leftarrow \tilde{u}^h + \tilde{e}^h$.
7) Relax $A^h \tilde{u}^h \approx b^h$ (n_2 times).

Typical values for the number n_1 and n_2 of pre- and post-smoothing steps are 1 or 2. In our application, we employ 1 pre- and 1 post-smoothing Gauss-Seidel relaxation step with an overrelaxation parameter of 1.7. Recursive application of the two-grid correction scheme on a hierarchy of successively coarser grids to solve the residual equation (step 4) leads to a *multigrid V-cycle scheme*. On the coarsest level, where the number of unknowns is small, a direct solver—we employ a Cholesky solver—is typically used to solve the residual equation. The resulting multigrid solver exhibits linear time complexity in the number of unknowns.

Coarse grid hierarchy. A nested hierarchy of ever coarser grids is constructed in a bottom-up process. Contiguous blocks of 2^3 cells are successively grouped into one cell on the next coarser level. A coarse grid cell is created if it contains *at least* one cell at the finer level. This basic construction principle, which has already been used in the context of composite finite elements [61], enables to automatically create a coarse grid hierarchy independent of the complexity of the shape of the object. However, since in this approach cells are merged solely based on their spatial location, for objects with complicated (concave) boundaries, physically disconnected parts might be merged into the same coarse grid cell, leading to a reduced coarse grid approximation quality and thus to reduced convergence rates. Especially in the current scenario, the cutting of the finite element model leads to highly complex boundaries, with disconnected parts of the object lying closely together. To avoid a reduction of convergence rates, we propose a grid hierarchy that reflects the cuts on the coarser grids to correctly approximate the physical separation of the material parts. The construction of this hierarchy will be explained in Section 6.2.

Coarse grid operators and transfer operators. We use Galerkin-based coarsening to construct the coarse grid versions of A^h, i.e., these operators are successively constructed via $A^{2h} = R_h^{2h} A^h I_{2h}^h$. Furthermore, we employ tri-linear interpolation operators I_{2h}^h, and the restriction operators are obtained by transposition of the interpolation operators, i.e., $R_h^{2h} = \left(I_{2h}^h\right)^{\mathrm{T}}$.

Using a finite element discretization with tri-linear shape functions, these choices of the coarse grid and transfer operators naturally arise from the principle of potential energy minimization underlying the variational formulation of elasticity problems: The total potential energy, consisting of the potential energy of the deformable body and the potential energy of the external forces is

$$E\left(u^h\right) = \frac{1}{2}\left(u^h\right)^{\mathrm{T}} K^h u^h - \left(f^h\right)^{\mathrm{T}} u^h. \tag{1}$$

Solving the elasticity problem is equivalent to minimizing the potential energy, i.e., equivalent to solving $\frac{\partial}{\partial u^h} E\left(u^h\right) = K^h u^h - f^h = 0$.

Given an approximate solution $\tilde{u}^h$, its error e^h is solved for on a coarser grid. By using a tri-linear interpolation operator I_{2h}^h, i.e., $\tilde{e}^h = I_{2h}^h e^{2h}$, the error e^h is approximated on the coarser grid by tri-linear shape functions with *twice* the support (in each dimension) as on the fine grid. Applying the coarse grid correction, according to Equation 1 the potential energy is $E\left(\tilde{u}^h + I_{2h}^h e^{2h}\right) = E\left(\tilde{u}^h\right) + \frac{1}{2}\left(e^{2h}\right)^{\mathrm{T}}\left(I_{2h}^h\right)^{\mathrm{T}} K^h I_{2h}^h e^{2h} - \left(\left(f^h\right)^{\mathrm{T}} - \left(u^h\right)^{\mathrm{T}} K^h\right) I_{2h}^h e^{2h}$. e^{2h} is determined by minimizing the potential energy, i.e., by solving

$$\begin{aligned}\frac{\partial}{\partial e^{2h}} & E\left(\tilde{u}^h + I_{2h}^h e^{2h}\right) \\ &= \underbrace{\left(I_{2h}^h\right)^{\mathrm{T}} K^h I_{2h}^h}_{=K^{2h}} e^{2h} - \underbrace{\left(I_{2h}^h\right)^{\mathrm{T}}}_{=R_h^{2h}} \underbrace{\left(f^h - K^h \tilde{u}^h\right)}_{=r^h} = 0,\end{aligned}$$

which directly yields the Galerkin coarse grid operator and the restriction operator. Note that the error e^{2h} resides on the coarser grid, but that the potential energy E is always calculated with respect to the discretization of the object on the finest grid. It is thus considered that on the coarser grids cells at the object's boundary might be only partially filled with material.

We use a matrix-free implementation of all multigrid components. Details on the construction of the coarse grid equations will be given in Section 6.4.

6.2 Hierarchy Construction - Uniform Grid

The challenge in developing a multigrid approach that supports topological changes of the simulation grid lies in constructing the coarse grid hierarchy. Since cuts on the finest level also have to be modeled on the coarser levels, cells cannot simply be merged based on their spatial relationship. This would correspond to merging physically and mechanically disconnected parts, and it would result in low approximation quality on the coarse grids and very slow convergence rate of the solver. In the following we describe the novel principle underlying our hierarchy construction. For the sake of clarity we first restrict the discussion to a uniform hexahedral grid with cuts.

The common approach to build a hexahedral multigrid hierarchy is to merge blocks of 2^3 cells into one coarse grid cell. To respect physically disconnected parts of the simulation domain, a similar merging strategy is performed but the connectivity between the merged cells is considered to possibly generate more than one coarse grid cell at the same position. A similar strategy has been pursued by Aftosmis et al. [67] to handle complex embedded object boundaries in a multigrid solver for computational fluid dynamics.

The basic idea underlying our construction is to interpret each grid of the hierarchy as an undirected graph $(\mathcal{C}^\ell, \mathcal{E}^\ell)$. The level number $\ell = 0$ denotes the finest level of the hierarchy, and ascending numbers denote successively coarser levels. The nodes $\mathcal{C}^\ell$ of each graph represent the cells of the respective grid,

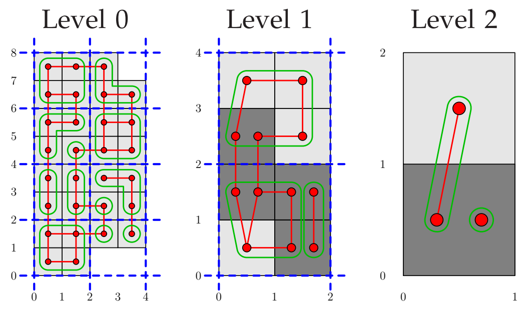

Fig. 5. Construction of the coarse multigrid levels based on the undirected graph representation (red) in a uniform grid. The blocks of double cell size, in which connected components (green) are searched, are marked in blue. Duplicated cells are indicated in dark gray. The numbers denote integer coordinates in the underlying lattice.

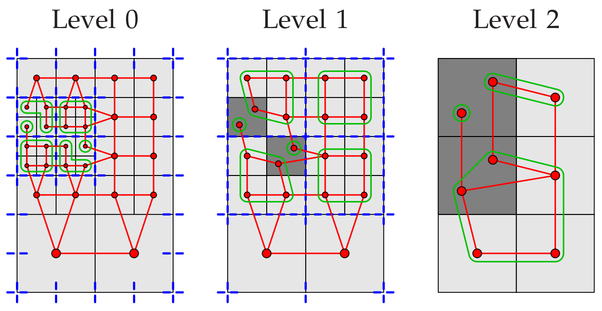

Fig. 6. Construction of the coarse multigrid levels based on the undirected graph representation in an octree grid. The notation is analogous to Figure 5.

and the edges $\mathcal{E}^\ell \subseteq \mathcal{C}^\ell \times \mathcal{C}^\ell$ describe the connectivity between face-adjacent cells. Connected cells share their vertices. In the following, a cell $c \in \mathcal{C}^\ell$ with domain $[x_c, x_c+1] \times [y_c, y_c+1] \times [z_c, z_c+1]$ in the underlying lattice (with a spacing of 2^ℓ relative to the spacing of the finest level) is associated with position (x_c, y_c, z_c).

At the finest level, the nodes $\mathcal{C}^0$ correspond to the finite elements, and the edges $\mathcal{E}^0$ correspond to the links introduced in Section 4.1. Starting on the finest level and proceeding to the coarsest level, the following two steps are performed subsequently at each level $\ell = 0, 1, \ldots$ to build the coarse grid hierarchy (see also the 2D example in Figure 5):

Step 1: Analogous to the common approach, blocks of double grid cell size are considered. Let $\mathcal{C}^\ell_{(x,y,z)} \subseteq \mathcal{C}^\ell$, $x, y, z \in \mathbb{Z}$ denote the set of nodes corresponding to the cells of such a block, defined by $\mathcal{C}^\ell_{(x,y,z)} = \{c \in \mathcal{C}^\ell \mid \lfloor x_c/2 \rfloor = x, \lfloor y_c/2 \rfloor = y, \lfloor z_c/2 \rfloor = z\}$. By performing a depth first search on the subgraph of $(\mathcal{C}^\ell, \mathcal{E}^\ell)$ induced by the nodes $\mathcal{C}^\ell_{(x,y,z)}$, the connected components of this subgraph are determined. For each connected component, a coarse grid cell $C \in \mathcal{C}^{\ell+1}$ at position $(x_C, y_C, z_C) = (x, y, z)$ is created, which subsumes the finer grid cells belonging to this component.

Step 2: The connectivity between cells on the coarse grid level is determined from the connectivity on the finer level. Two coarse grid cells C_1 and C_2 are connected iff there exist two connected fine grid cells c_1 and c_2 which are merged into C_1 and C_2, respectively. I.e., it is $(C_1, C_2) \in \mathcal{E}^{\ell+1}$ iff there exist c_1 in C_1 and c_2 in C_2 such that $(c_1, c_2) \in \mathcal{E}^\ell$. The notation c in C denotes that the finer grid cell c is belonging to the connected component corresponding to the coarse grid cell C.

The hierarchy that is constructed in this way has the property that small cuts disappear at coarser resolution levels. This is in-line with the multigrid idea that the coarse grid levels describe the macroscopic behavior of the body. Small cuts do not strongly affect this behavior and can therefore be neglected at one of the coarser levels.

6.3 Hierarchy Construction - Octree Grid

Building the hierarchy on the octree grid proceeds similar to the uniform case, with the exception that on each multigrid level it must be considered that cells with different size exist. At the transition from multigrid level ℓ to level $\ell+1$, only cells of size 2^ℓ are merged. All other cells are passed to the next coarser level. In Figure 6 a 2D example of the construction process is given.

6.4 Construction of Coarse Grid Equations

Once the cell hierarchy has been constructed, the coarse grid equations have to be built correspondingly. Since the corotated strain formulation is used, the equations have to be rebuilt in every time step to account for the element rotations.

Since all simulation elements have the same shape and only differ in size, we need to pre-compute only one element stiffness matrix. If we denote by K^e this instance for a uniform hexahedron with side length 1, then a uniform hexahedron with side length s has the element stiffness matrix sK^e [68].

The restricted octree discretization leads to hanging vertices lying in the interior of other cells' edges or faces. To obtain a continuous discretization of the displacement (finest grid) or the error (coarse grids) using tri-linear shape functions, a hanging vertex does *not* represent a degree of freedom but the value at this vertex is determined by linear (along edges) or bilinear (on faces) interpolation. Thus, we substitute unknowns at hanging vertices by interpolation from unknowns at non-hanging vertices. Each cell finally depends on eight non-hanging vertices (which are not necessarily the geometric vertices of the cell), and we associate the cell with these vertices. This approach corresponds to the special finite elements introduced by Wang [69] and extended to 3D by Sampath and Biros [50].

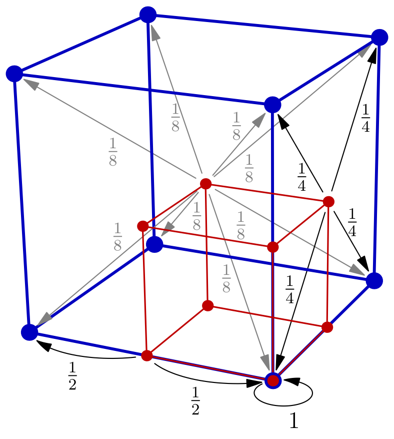

Fig. 7. Tri-linear weights used to distribute the per-cell equations of a fine grid cell (red) to its corresponding coarse grid cell (blue) (restriction). For the interpolation of the unknowns from the coarse grid, the same weights are used. For simplicity reasons, the distribution is only illustrated for selected vertices.

Computing the Galerkin coarse grid operators $A^{2h} = R_h^{2h} A^h I_{2h}^h$ means distributing the equations at the fine grid vertices to the respective coarse grid vertices (restriction R_h^{2h}), at the same time substituting the unknowns at the fine grid vertices by interpolation from the unknowns at the coarse grid vertices (interpolation I_{2h}^h). The respective weights of the tri-linear interpolation operator I_{2h}^h and the restriction operator $R_h^{2h} = \left(I_{2h}^h\right)^{\mathrm{T}}$ are illustrated in Figure 7.

We compute the equations on each level per multigrid *cell*, since this leads to constant memory requirements per cell to store the equations and to constant memory access patterns during equation construction. In contrast, constructing the equations per shared *vertex* would yield varying memory requirements and access patterns, since vertices can have a varying number of adjacent vertices due to the irregular topology induced by cutting.

At a specific level, the left-hand sides of the equations for each cell c are stored as rows of an 8×8 matrix A^c, with each entry being itself a 3×3 matrix. The matrix A^C for a coarse grid cell C on the next coarser level is then computed from the matrices A^c of those cells c which are merged into cell C as follows:

$$A^C_{mn} = \sum_{c \text{ in } C} \sum_{i=1}^{8} \left(\underbrace{w^{c \to C}_{i \to m}}_{\text{Restr.}} \sum_{j=1}^{8} \underbrace{w^{C \to c}_{n \to j}}_{\text{Interp.}} A^c_{ij} \right), \qquad m,n = 1,\ldots,8. \tag{2}$$

In this equation, i, j, m, n are cell-local vertex indices, and $w^{c \to C}_{i \to m}$ and $w^{C \to c}_{n \to j}$ are the corresponding weights for restriction and interpolation between the cells c on the finer grid and the coarse grid cell C, respectively (see Figure 7). Since all element matrices are symmetric (considering 24×24 matrices with scalar entries, i.e., $A^c_{pq} = {A^c_{qp}}^{\mathrm{T}}$, $p, q = 1, \ldots, 24$), we only have to compute and store the matrices' lower triangular parts. To simulate very large models consisting of millions of elements, memory requirements can be reduced significantly by not storing the equations on the finest level, since they can be built on-the-fly from the generic element stiffness matrix and the element rotations.

To finally solve the system of equations, in the relaxation step and the residual computation step of the multigrid V-cycle equations per shared vertex have to be considered. These per-vertex equations are assembled from the per-cell equations as follows: At a shared vertex V, described by the set of cell vertices $V \subset \mathcal{C}^\ell \times \{1, \ldots, 8\}$ that are coalesced into this vertex, we accumulate the per-cell equations at these cell vertices. The resulting per-vertex equation at the shared vertex V then is

$$\sum_{(c,i) \in V} \sum_{j=1}^{8} A^c_{ij} u^{V(c,j)} = b^V.$$

Here, $c \in \mathcal{C}^\ell$ denotes a cell incident to the shared vertex and $i, j \in \{1, \ldots, 8\}$ denote cell-local vertex indices. $V(c, j)$ is the shared vertex corresponding to vertex j of cell c, and $u^{V(c,j)}$ is the displacement/error at this vertex. b^V is the right-hand side/residual at vertex V.

7 Surface Construction

To reconstruct a smooth polygonal surface that is aligned with the separated object parts we use the splitting cubes algorithm [20]. In a hexahedral simulation grid the splitting cubes algorithm constructs the surface topology in each cell depending on the patterns of the edges that are cut. The algorithm introduces additional points in the interior of the cells to construct a smooth surface representation. The placement of these points is driven by the normals of the cut surface at the intersection points (see Figure 8).

In order to use the splitting cubes algorithm in our approach we have to adapt the algorithm to the particular simulation data structure we use, i.e., the linked voxel representation. Therefore, we consider the dual grid representation that is built from the links between the simulation elements. These links define the connectivity between the simulation elements, and exactly these links are cut by the cutting tool. For each link that is cut we store two distances from the link end points to the respective nearest cut point on that link. In this way, the cubic cells of the dual grid are cut at their edges, which results in a representation that allows us to directly apply the splitting cubes algorithm at these cells.

To let the reconstructed surface move according to the object deformations we bind the surface to the simulation vertices. For vertices on the links that are cut, the corresponding simulation element is found

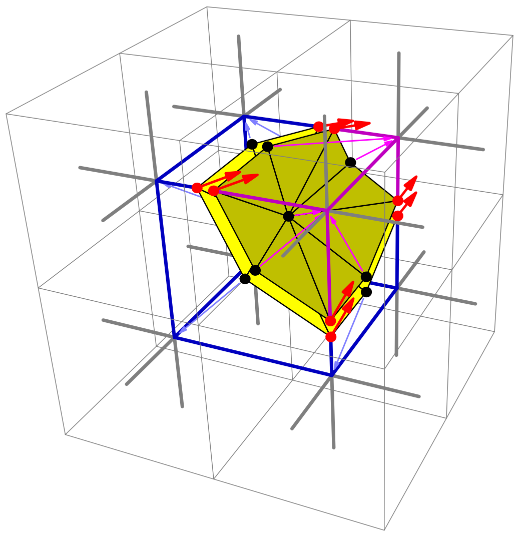

Fig. 8. Dual grid (bold lines) consisting of the links between the elements. For each cell of the dual grid a local render surface is built via the splitting cubes algorithm. The surface is spanned by the intersection points (red) between the cutting blade and the element links, and by the normal of the cutting tool at these points (red arrows). Vertices are bound to the nearest element of the respective connected component (thin blue and magenta arrows to the respective element centers).

by following the links to the left or to the right, respectively. To handle interior vertices we utilize the fact that each interior vertex is associated to a unique surface patch in the splitting cubes approach. Therefore, for an interior vertex the respective hexahedra can be found by following the links of the edges that are cut by the respective surface patch. From the determined set of hexahedra we choose the one that has the shortest distance to the surface vertex. The interior vertex is then bound to this hexahedron by means of tri-linear interpolation or extrapolation.

Using standard per-vertex normals, which are computed by averaging the face normals of all triangles incident to the vertex, leads to rendering artifacts at cutting edges as shown in Figure 9. To achieve a visually pleasant rendering of the splitting cubes surface, we render both sharp cutting edges as well as smooth cutting surfaces by employing per-face and per-vertex normals, respectively. To compute the respective normals, we make use of a cut surface ID, which is incremented for each new cut and then stored at the links intersected by this cut together with the distances. These IDs are distributed to the incident triangles of the splitting cubes surface, and the normals for each triangle vertex are computed by only averaging the face normals of triangles with the same ID. The cut surface IDs can also be used to color the arising cutting surfaces differently (see Figure 1 right).

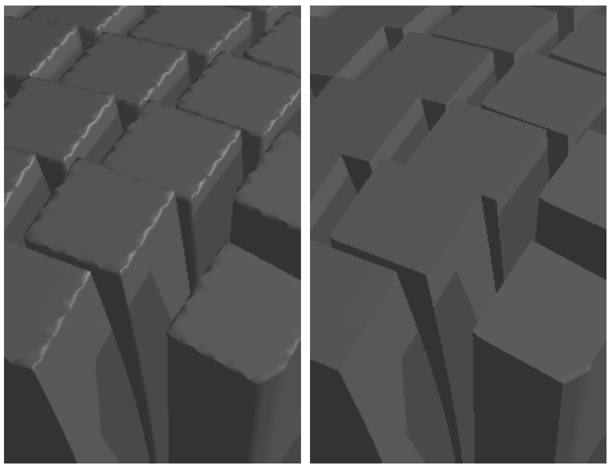

Fig. 9. Left: Standard per-vertex normals computed by averaging the face normals of all incident triangles lead to rendering artifacts at cutting edges. Right: Only face normals of triangles with the same cut ID are averaged, enabling smooth rendering of cutting surfaces while preserving sharp cutting edges.

8 Results and Analysis

In the following we analyze our method and show results that have been produced using this method. The analysis includes performance measures, a detailed evaluation of the convergence rates of the multigrid solver, and a comparison to alternative numerical solvers. All of our experiments were run on a desktop PC, equipped with an Intel Xeon X5560 2.8 GHz processor (we use a single core), 8 GB of RAM, and an NVIDIA GeForce GTX 280 graphics card.

8.1 Performance

Table 1 lists the deformable models we used and gives timings for cutting and simulation. The first three entries correspond to Figures 15 (left), 11 (right), and 1 (middle). The last two entries also correspond to the model shown in Figure 15 (left), but using an adaptive octree discretization reduced by one (Bunny 2) or two (Bunny 3) octree levels. The first two columns give the initial number of simulation elements and the number of elements due to adaptive refinements along the cuts. The next column shows the time spent on adaptively refining the octree and rebuilding the finite element model. Note that these steps are only required when the cutting tool is moved. Finally, the time required to perform one time step is given. It includes the times it takes to recompute the equations on the simulation level and all coarse grids (as shown in Equation 2) and to perform three multigrid V-cycles with one pre- and one post-smoothing Gauss-Seidel step. In a dynamic simulation this is a reasonable choice since the solution from the last time step typically gives a good initial guess to start with in the current time step. These times are measured for

	#Hexahedra		Time [sec]	
Model	Initial	Cut	Cutting	Simulation
Bunny	118K (755K)	197K	0.693	2.42 (9.31)
Cube	83.3K (512K)	230K	0.781	3.13 (7.05)
Armadillo	493K (3717K)	643K	2.24	8.24 (47.2)
Bunny 2	26.1K (94.3K)	42.4K	0.143	0.498 (1.12)
Bunny 3	5.48K (11.9K)	8.27K	0.0286	0.0928 (0.130)

TABLE 1
Timing statistics for cutting different deformable models using our adaptive finite element approach. Numbers in parentheses show respective measures for a uniform grid at the finest resolution level.

Model	MG Time [sec]	MG Error Reduction	CG Error Reduction	CG Time [sec]
Bunny	1.30	0.100	0.935	9.76
Cube	1.94	0.0145	0.870	31.9
Armadillo	4.35	0.0190	0.973	101
Bunny 2	0.270	0.107	0.864	1.22
Bunny 3	0.0532	0.0734	0.655	0.172

TABLE 2
Comparison of the adaptive MG solver to a CG solver with Jacobi preconditioner. Solver times are given for one time step. The error reduction is computed as $\|e_3\|_2 \,/\, \|e_0\|_2$, where e_0 and e_3 are the linearized error vectors before and after the time step, respectively. The last column shows the time that is required by CG to achieved the same error reduction as MG.

the adaptively refined models. Times in parentheses are for a uniform grid at the finest resolution level.

Table 1 demonstrates the advantages of an adaptive octree discretization over a uniform discretization. It shows that a considerable amount of elements can be saved, yet modeling the induced discontinuities at equal resolution. For the higher resolution models only $1/6$ of the number of elements of the uniform grid are required. In general, this allows resolving the boundaries at a significantly higher resolution than would be possible in a uniform setting, giving the possibility to apply very complicated and thin cuts. Remarkably, even though the numerical simulation on an octree discretization is much more complicated than on a uniform discretization, the reduced number of elements also results in a significant performance gain over the uniform setting. In our scenarios, speed-ups between a factor of 4 and 5 are achieved. Furthermore, the statistics show that the computation time per simulation step depends linearly on the number of finite elements.

To demonstrate the computational efficiency and accuracy of the adaptive MG approach, in Table 2 we show a comparison of the experiments in Table 1 to the very same setup using a CG solver with Jacobi preconditioner. In this table only the solver times, i.e., the times required for solving the equations on the finest level, are considered. For the MG solver this means that the given times (2nd column) also include the construction of the equations on the coarser multigrid levels. The 3rd column gives the error reduction that was achieved by the adaptive MG solver. In the 4th column we show the error reduction by the CG solver within the same period of time as given in the 2nd column. Finally, the 5th column shows the times required by the CG solver to achieve the same error reduction as the MG solver.

8.2 Multigrid Convergence Analysis

To demonstrate the effectiveness of the multigrid solver for simulating objects with complicated boundaries as induced by a cut, in Figure 10 (left) we show the solver's convergence for the scenario in Figure 11 (right). The curves show the total error reduction $\|e_k\|_2 \,/\, \|e_0\|_2$ dependent on the number k of V-cycles, with e_k denoting the error after the k-th V-cycle. The error is determined by using the solution obtained by a direct solver (Cholesky) as ground truth.

Three different tests were performed, all of them using a cube model aligned with the axes of the simulation grid: a) no cuts were performed, b) a grid-like cutting tool consisting of square blades aligned with the simulation grid was moved into the cube for a distance of half the cube's edge length, c) the cutting tool was rotated against the axes of the simulation grid to create cuts that are not aligned with the simulation grid. In each of the experiments the cuts were performed at a single point in time, and the solver's performance was measured for simulating the next time step right after the model was cut.

In experiment a) the grids of all multigrid levels exactly cover the domain of the cube, giving rise to an optimal multigrid hierarchy. A convergence rate $\rho = 0.66$ is achieved. The convergence rate is computed as $\rho = \left(\|e_{k_2}\|_2 \,/\, \|e_{k_1}\|_2\right)^{1/(k_2-k_1)}$ and denotes the average error reduction per V-cycle for those cycles k_1 through k_2 which exhibit a stable convergence rate. Experiments b) and c) demonstrate the influence of complicated boundaries on the convergence. In b) and c) the same spacing between the individual blades is used, but in contrast to b) a jagged object boundary is generated in c) due to the cut orientation.

In both scenarios the convergence rate decreases to 0.75. This decrease is in line with previous find-

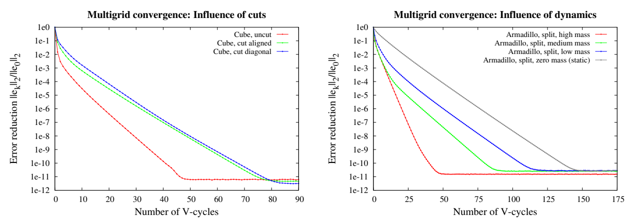

Fig. 10. Multigrid convergence. Left: Differently oriented cuts through a cube (Figure 11 right). Right: The Armadillo model (Figure 1 middle) having different mass.

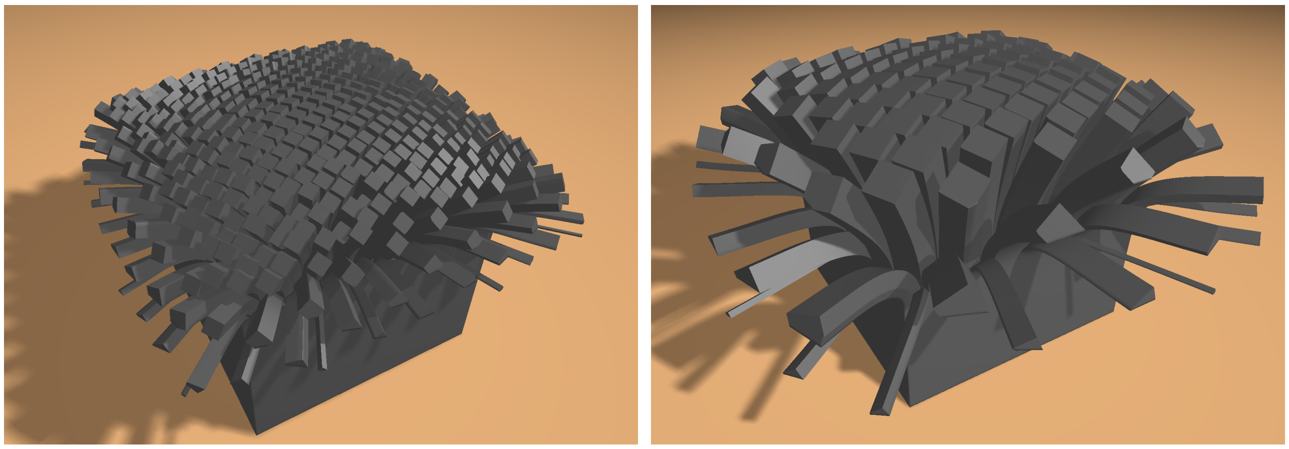

Fig. 11. In one time step a cube model (83K hexahedral elements) is cut using a grid-like cutting tool consisting of square blades, resulting in 204K (left) and 230K (right) elements. A radial force field is applied to fan out the resulting rods. Note that rods with different size bend differently. Cutting and simulation takes less than 3.5 seconds (left) and 3.9 seconds (right) per time step.

ings in the context of geometric multigrid schemes, which have indicated decreasing convergence in case the coarse grid cells can no longer approximate the object's domain accurately, which is the case if the material is cut into disconnected parts. Notably, however, the convergence behavior does not depend on the smoothness of the boundary of the object.

In another example we analyze the influence of the model dynamics on the solver's convergence. Since the values on the main diagonal of our system matrix depend on the mass of the object, the system's numerical stiffness is decreased proportionally to the mass. As a result, higher element masses yield better convergence rates, as indicated in Figure 10 (right) for the model shown in Figure 1 (middle). In comparison we have also simulated the static problem where the dynamics is not considered (gray curve). This curve indicates that independent of the object's mass a convergence rate of at least 0.85 can be achieved for this model by the multigrid solver.

8.3 Solver Comparison

In the following we compare the performance of the multigrid solver to alternative solvers that are widely used for simulating deformable objects. One important aspect in this analysis is the potential of the solver to reduce the error at most in a given period of time. This is required, for instance, to guarantee a given response time in interactive scenarios.

For the models shown in Figures 11 (right), and 1 (middle) with medium and zero mass, Figure 12 shows the error reduction of different numerical solvers over time for the solution of the first simulation time step, i.e., the cuts are performed to the object in its initial position, and an initial guess of 0 is used for the displacements. Here, $e(t)$ is the error after the respective solver has run for time t. In this comparison the respective initialization times of the solvers are included, i.e., each solver starts solely from the finest level equations and all computations specific to the solver—in case of the multigrid solver the construction of the coarse grids, the assembly of the coarse grid equations, and the initialization of the direct solver on the coarsest level—are included in the timings.

We analyze the Cholesky solver of the TAUCS library [70], a CG solver with Jacobi preconditioner, and the proposed multigrid solver. The multigrid solver is either used directly (MG) or as a preconditioner for the conjugate gradient method (CGMG). For each solver a cross on the respective curve highlights the structural initialization time that is only required in case of topological changes. For the Cholesky solver, structural initialization consists of the symbolical factorization of the system matrix. For the multigrid solver, structural initialization refers to the construction of the coarse grid hierarchy. (For MG and CGMG the crosses fall onto the same position.) CG does not have a structural initialization time.

Besides needing a vast amount of time to solve the system of linear equations, Cholesky turns out to be impractical for applications requiring a first approximation of the solution within a short time interval. CG-Jacobi shows a significantly slower convergence rate than MG and CGMG. Both multigrid solvers converge much faster towards the solution than their competitors and, in particular, they are able to provide good approximations in a significantly shorter time. CGMG increases the error reduction per time slightly, but this benefit only pays off if a large number of cycles is performed.

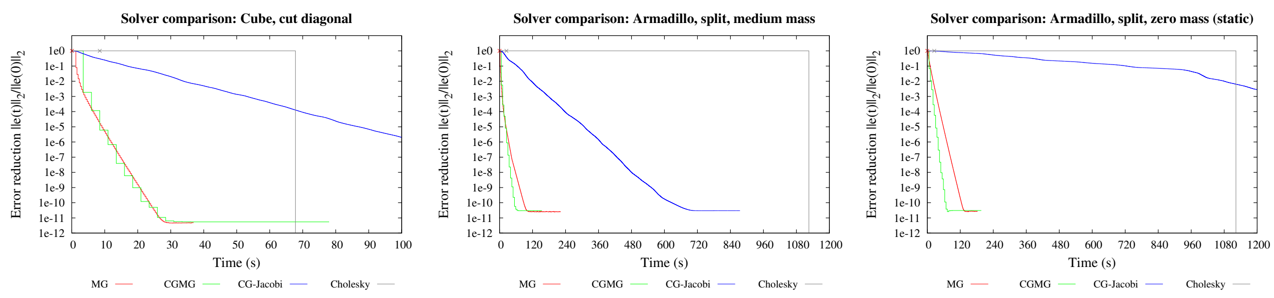

Fig. 12. Computational efficiency of different numerical solvers for deformable object simulation.

9 Conclusion and Future Work

We have proposed an efficient approach for physics based cutting of deformable objects. This approach employs an adaptive octree grid to represent cuts at very fine scales. The cutting algorithm is incorporated into a multigrid scheme, giving rise to a numerical solver that can handle topological changes in the simulation grid at high computational efficiency. To reconstruct a smooth surface from the disconnected object parts, an extension of the splitting cubes approach has been proposed. This extension uses the dual simulation grid to build a boundary surface that is consistent with the cuts in the simulation grid.

It is worth noting that in a particular scenario, the unrestricted refinement along a cut may result in more elements than would actually be required to solve accurately. For instance, homogenization approaches [34], [71] could possibly reduce the number of elements by identifying the material properties at coarser scales from those of their constituents and using only the respective coarse grid cells in the simulation. Thus, even with less efficient numerical solvers homogenization approaches can often simulate very fast. In the general case, however, such approaches yield an approximation to the numerical solution on a finer discretization using 'non-homogenized' finite elements, since they reduce the number of degrees of freedom to solve for. The principle underlying our approach is to simulate on a finite element model that has as many degrees of freedom as given by the initial discretization and to achieve high speed by employing a computationally efficient numerical solver. Thus, our approach always simulates accurately at the possible expense of a higher number of simulation elements.

The proposed method opens a number of future research directions. Since in the current approach we fix the resolution of the simulation grid in the interior of the deforming object, the resulting number of degrees of freedom might not be sufficient to accurately represent the induced deformations. Therefore, a dynamic adaption of the finite element discretization in the object's interior is desirable. This can directly be integrated into our approach, but it first requires to develop an a priori oracle to decide where to refine.

Another interesting question is how to efficiently adapt a high resolution render surface to the topological changes of the simulation mesh. In the current approach the resolution of the finite element model and the render surface are coupled due to the use

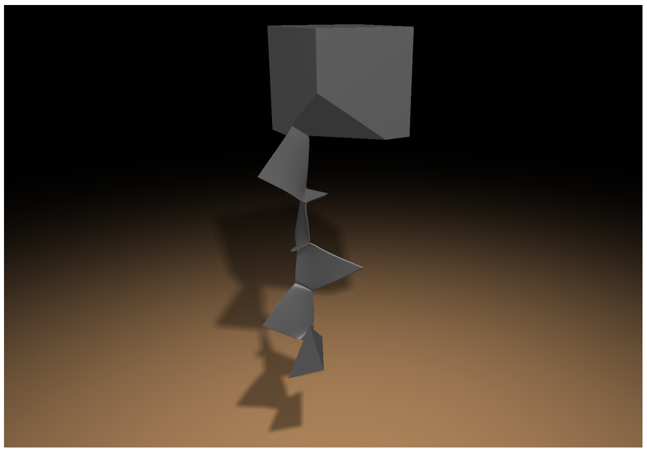

Fig. 13. Cutting of thin slices (initially 345,000 hexahedra, 429,000 hexahedra after cutting). Cutting and simulation take 7.3 seconds per time step.

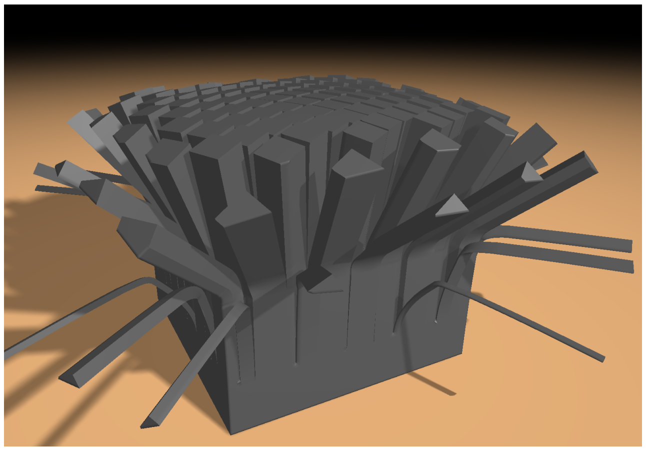

Fig. 14. Cutting of a heterogeneous model (initially 83,000 hexahedra, 321,000 hexahedra after cutting). The cube consists of very stiff material, except of a thin horizontal layer at half height, leading to buckling of the rods. Cutting and simulation are performed at 5.5 seconds per time step.

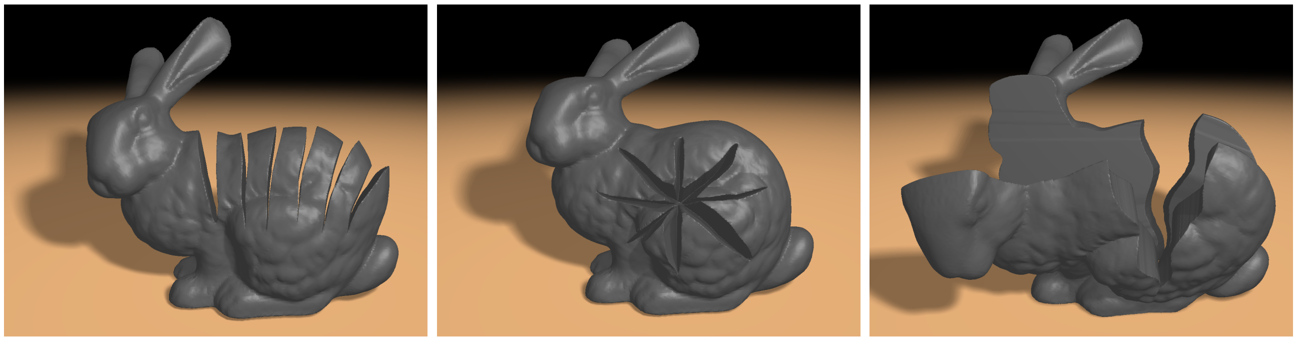

Fig. 15. A potpourri of different cuts is applied to the Stanford bunny model to demonstrate the flexibility of our approach. Multiple overlapping cuts (middle) can be handled accurately. From left to right, the cut models consist of 197,000, 166,000, and 177,000 hexahedral elements. Cutting and simulation are performed at approximately 3.1, 2.6, and 2.8 seconds per time step, respectively.

of the splitting cubes algorithm. Even though it is possible to bind a higher resolution render surface to the vertices of the initial simulation grid, it is unclear how to adapt this surface to the induced topological changes at the same speed than the simulation.

Furthermore, we will also integrate collision handling into our approach. In particular we plan to use the deforming splitting cubes surface to detect self-collisions [72] and to propagate the collision response to the respective simulation elements.

Acknowledgments

The first author is funded by the International Graduate School of Science and Engineering (IGSSE) of the Technische Universität München. The authors thank Peter Schröder for helpful discussions and hints.

References

[1] A. Brandt, "Multi-level adaptive solutions to boundary-value problems," *Mathematics of Computation*, vol. 31, no. 138, pp. 333–390, 1977.

[2] W. Hackbusch, *Multi-Grid Methods and Applications*, ser. Springer Series in Computational Mathematics. Springer, 1985.

[3] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed. SIAM, 2000.

[4] D. Bielser, V. A. Maiwald, and M. H. Gross, "Interactive cuts through 3-dimensional soft tissue," *Computer Graphics Forum*, vol. 18, no. 3, pp. 31–38, 1999.

[5] D. Bielser and M. H. Gross, "Interactive simulation of surgical cuts," in *Proc. Pacific Graphics*, 2000, pp. 116–125.

[6] A. B. Mor and T. Kanade, "Modifying soft tissue models: Progressive cutting with minimal new element creation," in *Proc. MICCAI*, ser. Lecture Notes in Computer Science, vol. 1935, 2000, pp. 598–608.

[7] M. Wicke, M. Botsch, and M. Gross, "A finite element method on convex polyhedra," *Computer Graphics Forum*, vol. 26, no. 3, pp. 355–364, 2007.

[8] I. Babuška and J. M. Melenk, "The partition of unity method," *International Journal for Numerical Methods in Engineering*, vol. 40, no. 4, pp. 727–758, 1997.

[9] T. Strouboulis, K. Copps, and I. Babuška, "The generalized finite element method," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, no. 32-33, pp. 4081–4193, 2001.

[10] T. Belytschko and T. Black, "Elastic crack growth in finite elements with minimal remeshing," *International Journal for Numerical Methods in Engineering*, vol. 45, no. 5, pp. 601–620, 1999.

[11] N. Moës, J. Dolbow, and T. Belytschko, "A finite element method for crack growth without remeshing," *International Journal for Numerical Methods in Engineering*, vol. 46, no. 1, pp. 131–150, 1999.

[12] N. Sukumar, N. Moës, B. Moran, and T. Belytschko, "Extended finite element method for three-dimensional crack modelling," *International Journal for Numerical Methods in Engineering*, vol. 48, no. 11, pp. 1549–1570, 2000.

[13] Y. Abdelaziz and A. Hamouine, "A survey of the extended finite element," *Computers & Structures*, vol. 86, no. 11-12, pp. 1141–1151, 2008.

[14] L. Jeřábková and T. Kuhlen, "Stable cutting of deformable objects in virtual environments using XFEM," *IEEE Computer Graphics and Applications*, vol. 29, no. 2, pp. 61–71, 2009.

[15] P. Kaufmann, S. Martin, M. Botsch, E. Grinspun, and M. Gross, "Enrichment textures for detailed cutting of shells," *ACM TOG*, vol. 28, no. 3, pp. 50:1–50:10, 2009.

[16] G. Debunne, M. Desbrun, A. H. Barr, and M.-P. Cani, "Interactive multiresolution animation of deformable models," in *Proc. Eurographics Workshop on Computer Animation and Simulation*, 1999, pp. 133–144.

[17] M. Müller and M. Gross, "Interactive virtual materials," in *Proc. Graphics Interface*, 2004, pp. 239–246.

[18] J. Georgii and R. Westermann, "Interactive simulation and rendering of heterogeneous deformable bodies," in *Proc. Vision, Modeling and Visualization*, 2005, pp. 383–390.

[19] M. Botsch, M. Pauly, M. Wicke, and M. Gross, "Adaptive space deformations based on rigid cells," *Computer Graphics Forum*, vol. 26, no. 3, pp. 339–347, 2007.

[20] N. Pietroni, F. Ganovelli, P. Cignoni, and R. Scopigno, "Splitting cubes: a fast and robust technique for virtual cutting," *The Visual Computer*, vol. 25, no. 3, pp. 227–239, 2009.

[21] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically deformable models," in *Proc. ACM SIGGRAPH*, 1987, pp. 205–214.

[22] D. Terzopoulos and K. Fleischer, "Modeling inelastic deformation: Viscoelasticity, plasticity, fracture," in *Proc. ACM SIGGRAPH*, 1988, pp. 269–278.

[23] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," *Computer Graphics Forum*, vol. 25, no. 4, pp. 809–836, 2006.

[24] D. L. James and D. K. Pai, "ArtDefo: Accurate real time deformable objects," in *Proc. ACM SIGGRAPH*, 1999, pp. 65–72.

[25] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr, "Dynamic real-time deformations using space & time adaptive sampling," in *Proc. ACM SIGGRAPH*, 2001, pp. 31–36.

[26] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović, "A multiresolution framework for dynamic deformations," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002, pp. 41–47.

[27] E. Grinspun, P. Krysl, and P. Schröder, "CHARMS: A simple framework for adaptive simulation," *ACM TOG*, vol. 21, no. 3, pp. 281–290, 2002.

[28] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Mesh-

less deformations based on shape matching," *ACM TOG*, vol. 24, no. 3, pp. 471–478, 2005.

[29] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw, "Hybrid simulation of deformable solids," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007, pp. 81–90.

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

[31] X. Wu, M. S. Downes, T. Goktekin, and F. Tendick, "Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes," *Computer Graphics Forum*, vol. 20, no. 3, pp. 349–358, 2001.

[32] J. F. O'Brien and J. K. Hodgins, "Graphical modeling and animation of brittle fracture," in *Proc. ACM SIGGRAPH*, 1999, pp. 137–146.

[33] J. F. O'Brien, A. W. Bargteil, and J. K. Hodgins, "Graphical modeling and animation of ductile fracture," *ACM TOG*, vol. 21, no. 3, pp. 291–294, 2002.

[34] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure, "Preserving topology and elasticity for embedded deformable models," *ACM TOG*, vol. 28, no. 3, pp. 52:1–52:9, 2009.

[35] H.-W. Nienhuys and A. F. van der Stappen, "Combining finite element deformation with cutting for surgery simulations," in *Proc. Eurographics - Short Presentations*, 2000, pp. 43–52.

[36] S. Cotin, H. Delingette, and N. Ayache, "A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation," *The Visual Computer*, vol. 16, no. 8, pp. 437–452, 2000.

[37] C. Forest, H. Delingette, and N. Ayache, "Removing tetrahedra from a manifold mesh," in *Proc. Computer Animation*, 2002, pp. 225–229.

[38] H.-W. Nienhuys and A. F. van der Stappen, "A surgery simulation supporting cuts and finite element deformation," in *Proc. MICCAI*, ser. Lecture Notes in Computer Science, vol. 2208, 2001, pp. 145–152.

[39] D. Serby, M. Harders, and G. Székely, "A new approach to cutting into finite element models," in *Proc. MICCAI*, ser. Lecture Notes in Computer Science, vol. 2208, 2001, pp. 425–433.

[40] D. Steinemann, M. A. Otaduy, and M. Gross, "Fast arbitrary splitting of deforming objects," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006, pp. 63–72.

[41] D. Bielser, P. Glardon, M. Teschner, and M. Gross, "A state machine for real-time cutting of tetrahedral meshes," in *Proc. Pacific Graphics*, 2003, pp. 377–386.

[42] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno, "A multiresolution model for soft objects supporting interactive cuts and lacerations," *Computer Graphics Forum*, vol. 19, no. 3, pp. 271–281, 2000.

[43] N. Molino, Z. Bao, and R. Fedkiw, "A virtual node algorithm for changing mesh topology during simulation," *ACM TOG*, vol. 23, no. 3, pp. 385–392, 2004.

[44] S. Martin, P. Kaufmann, M. Botsch, M. Wicke, and M. Gross, "Polyhedral finite elements using harmonic basis functions," *Computer Graphics Forum*, vol. 27, no. 5, pp. 1521–1529, 2008.

[45] E. Sifakis, K. G. Der, and R. Fedkiw, "Arbitrary cutting of deformable tetrahedralized objects," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007, pp. 73–80.

[46] S. Popinet, "Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries," *Journal of Computational Physics*, vol. 190, no. 2, pp. 572–600, 2003.

[47] L. Shi and Y. Yu, "Visual smoke simulation with adaptive octree refinement," in *Proc. Computer Graphics and Imaging*, 2004, pp. 13–19.

[48] F. Losasso, F. Gibou, and R. Fedkiw, "Simulating water and smoke with an octree data structure," *ACM TOG*, vol. 23, no. 3, pp. 457–462, 2004.

[49] E. Haber and S. Heldmann, "An octree multigrid method for quasi-static Maxwell's equations with highly discontinuous coefficients," *Journal of Computational Physics*, vol. 223, no. 2, pp. 783–796, 2007.

[50] R. S. Sampath and G. Biros, "A parallel geometric multigrid method for finite elements on octree meshes," *SIAM Journal on Scientific Computing*, vol. 32, no. 3, pp. 1361–1392, 2010.

[51] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," *ACM TOG*, vol. 22, no. 3, pp. 917–924, 2003.

[52] X. Wu and F. Tendick, "Multigrid integration for interactive deformable body simulation," in *Proc. International Symposium on Medical Simulation*, ser. Lecture Notes in Computer Science, vol. 3078, 2004, pp. 92–104.

[53] J. Georgii and R. Westermann, "A multigrid framework for real-time simulation of deformable bodies," *Computer & Graphics*, vol. 30, no. 3, pp. 408–415, 2006.

[54] L. Shi, Y. Yu, N. Bell, and W.-W. Feng, "A fast multigrid algorithm for mesh deformation," *ACM TOG*, vol. 25, no. 3, pp. 1108–1117, 2006.

[55] M. Kazhdan and H. Hoppe, "Streaming multigrid for gradient-domain operations on large images," *ACM TOG*, vol. 27, no. 3, pp. 21:1–21:10, 2008.

[56] X. Jin, S. Chen, and X. Mao, "Computer-generated marbling textures: A GPU-based design system," *IEEE Computer Graphics and Applications*, vol. 27, no. 2, pp. 78–84, 2007.

[57] C. Dick, J. Georgii, R. Burgkart, and R. Westermann, "Computational steering for patient-specific implant planning in orthopedics," in *Proc. Eurographics Workshop on Visual Computing for Biomedicine*, 2008, pp. 83–92.

[58] Y. Zhu, E. Sifakis, J. Teran, and A. Brandt, "An efficient multigrid method for the simulation of high-resolution elastic solids," *ACM TOG*, vol. 29, no. 2, pp. 16:1–16:18, 2010.

[59] S. A. Sauter and R. Warnke, "Composite finite elements for elliptic boundary value problems with discontinuous coefficients," *Computing*, vol. 77, no. 1, pp. 29–55, 2006.

[60] T. Preusser, M. Rumpf, and L. O. Schwen, "Finite element simulation of bone microstructures," in *Proc. 14th Workshop on the Finite Element Method in Biomedical Engineering, Biomechanics and Related Fields*, 2007, pp. 52–66.

[61] F. Liehr, T. Preusser, M. Rumpf, S. Sauter, and L. O. Schwen, "Composite finite elements for 3D image based computing," *Computing and Visualization in Science*, vol. 12, no. 4, pp. 171–188, 2009.

[62] S. F. Frisken-Gibson, "Using linked volumes to model object collisions, deformation, cutting, carving, and joining," *IEEE TVCG*, vol. 5, no. 4, pp. 333–348, 1999.

[63] K.-J. Bathe, *Finite Element Procedures*. Prentice Hall, 2002.

[64] C. C. Rankin and F. A. Brogan, "An element independent corotational procedure for the treatment of large rotations," *ASME Journal of Pressure Vessel Technology*, vol. 108, no. 2, pp. 165–174, 1986.

[65] J. Georgii and R. Westermann, "Corotated finite elements made fast and stable," in *Proc. Workshop in Virtual Reality Interactions and Physical Simulation*, 2008, pp. 11–19.

[66] A. Lorusso, D. W. Eggert, and R. B. Fisher, "A comparison of four algorithms for estimating 3-D rigid transformations," in *Proc. British Conference on Machine Vision*, 1995, pp. 237–246.

[67] M. J. Aftosmis, M. J. Berger, and G. Adomavicius, "A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries, AIAA 2000-0808," in *Proc. 38th AIAA Aerospace Sciences Meeting and Exhibit*, 2000.

[68] L. Jeřábková, T. Kuhlen, T. P. Wolter, and N. Pallua, "A voxel based multiresolution technique for soft tissue deformation," in *Proc. ACM Symposium on Virtual Reality Software and Technology*, 2004, pp. 158–161.

[69] W. Wang, "Special bilinear quadrilateral elements for locally refined finite element grids," *SIAM Journal on Scientific Computing*, vol. 22, no. 6, pp. 2029–2050, 2001.

[70] S. Toledo, D. Chen, V. Rotkin, and O. Meshar, "TAUCS: A library of sparse linear solvers," 2003, http://www.tau.ac.il/∼stoledo/taucs.

[71] L. Kharevych, P. Mullen, H. Owhadi, and M. Desbrun, "Numerical coarsening of inhomogeneous elastic materials," *ACM TOG*, vol. 28, no. 3, pp. 51:1–51:8, 2009.

[72] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino, "Collision detection for deformable objects," *Computer Graphics Forum*, vol. 24, no. 1, pp. 61–81, 2005.

Christian Dick received the diploma degree in computer science from the Technische Universität München, Germany in July 2007. Since August 2007 he is a member of the Computer Graphics and Visualization Group at the Technische Universität München and working towards the PhD degree. His current research interests include interactive, physics-based simulation of deformable objects, multigrid methods, GPU computing, medical simulation and visualization, computational steering, as well as the visualization of very large scientific data sets such as terrain data.

Joachim Georgii is a PostDoc in the Simulation/Modelling Group as well as the Registration Group at Fraunhofer MEVIS, Bremen. From 2007 to 2010, he was a PostDoc at the Computer Graphics and Visualization Group headed by Professor Rüdiger Westermann at the Technische Universität München. In 2007, Joachim Georgii received a PhD in computer science at the Technische Universität München. His research interests are physics-based simulation techniques as well as their efficient implementations thereby focusing on medical applications such as image registration or surgical planning.

Rüdiger Westermann studied computer science at the Technical University Darmstadt, Germany. He pursued his doctoral thesis on multiresolution techniques in volume rendering, and he received a PhD in computer science from the University of Dortmund, Germany. In 1999, he was a visiting professor at the University of Utah in Salt Lake City, and he became an assistant professor at the University of Stuttgart, Germany. In 2000, he was appointed an associate professor at the Technical University Aachen, Germany, where he was head of the Scientific Visualization and Imaging Group. In 2002, he was appointed the chair of Computer Graphics and Visualization at the Technische Universität München. His research interests include real-time physical simulation, general purpose computing on GPUs, interactive data visualization, and real-time rendering.