Vis Comput (2011) 27: 519–529
DOI 10.1007/s00371-011-0561-3

ORIGINAL ARTICLE

Robust interactive cutting based on an adaptive octree simulation mesh

Martin Seiler · Denis Steinemann · Jonas Spillmann · Matthias Harders

Published online: 15 April 2011

Abstract We present an adaptive octree based approach for interactive cutting of deformable objects. Our technique relies on efficient *refine*- and *node split*-operations. These are sufficient to robustly represent cuts in the mechanical simulation mesh. A high-resolution surface embedded into the octree is employed to represent a cut visually. Model modification is performed in the rest state of the object, which is accomplished by back-transformation of the blade geometry. This results in an improved robustness of our approach. Further, an efficient update of the correspondences between simulation elements and surface vertices is proposed. The robustness and efficiency of our approach is underlined in test examples as well as by integrating it into a prototype surgical simulator.

Keywords Physically-based modeling · Cutting · Adaptive simulation · Octree

1 Introduction

The ability to modify objects in virtual environments by cutting or fracturing their geometric representation is an important field of research in computer graphics. Applications rank from mechanical simulations to medical simulators or video games.

Cutting is notorious for its algorithmic and computational complexity. The process is akin to a localized adaptive refinement of the simulation mesh. In fact, all the pitfalls that come with adaptive simulations also occur in cutting simulations: Ill-shaped simulation elements hampering the solving of the underlying mechanical model, element count explosion after multiple cuts in the same place, or incompatible elements in the cut region, just to name a few.

In the past, techniques have been established that circumvent or at least partially solve these challenges. However, researchers agree that no approach exists that entirely solves the aforementioned problems while still being efficient. This is particularly true if interactive cutting in surgical training simulators is considered, where a failure of the simulation is not tolerable.

In this paper, we propose a novel approach for handling interactive cutting in cases where pieces of material are completely excised from an object. Various real-world procedures are of this type, e.g., punching out material or drilling holes. The focus of our development is on maximum stability and controllability of the simulation mesh. At the current stage, we limit ourselves to non-progressive cuts where the blade is represented by a closed manifold surface defining a cut volume in space. We show that this restriction allows for a simple, but elegant solution.

The main contribution of our work is the extension of an adaptive octree simulation mesh with a local *refine*- and *node split*-operation in order to realize cutting that is both robust and efficient. The former operation allows refining the simulation mesh in the cut region, while the latter introduces new degrees-of-freedom to model the material discontinuity.

M. Seiler (✉) · J. Spillmann · M. Harders
ETH Zürich, Sternwartstrasse 7, 8092, Zürich, Switzerland
e-mail: seiler@vision.ee.ethz.ch

J. Spillmann
e-mail: jonas.spillmann@vision.ee.ethz.ch

M. Harders
e-mail: mharders@vision.ee.ethz.ch

D. Steinemann
VirtaMed AG, Technoparkstrasse 1, 8005 Zürich, Switzerland
e-mail: steinemann@virtamed.com

In addition, we propose a strategy to update the correspondences between the simulation mesh elements and the vertices of the embedded surface geometry by employing an optimized data structure. Moreover, we discuss a novel approach to transform the blade geometry into the *material space*, i.e., into the resting state of the deformed object. This is beneficial since in doing so cuts of heavily deformed objects are not hampered by ill-shaped geometries.

The main benefit of our approach, compared to previous techniques, is its high stability and efficiency which make it particularly appealing for interactive simulation systems. We illustrate this by integrating the approach into a prototype endoscopic surgery simulator.

The remainder of the paper is organized as follows: In Sect. 2, we review related work. Thereafter, we give an overview on our approach in Sect. 3. In Sect. 4, we propose our novel octree data structure equipped with the two key operations, refine and node split. Examples in Sect. 5 illustrate the characteristics of our method and evaluate overall performance.

2 Related work

The simulation of cutting and fracture of deformable objects is a subtopic of physically-based simulation. When an object is being cut, its underlying mesh is topologically changed. Consequently, cutting can be regarded as an adaptive simulation method.

2.1 Cutting conforming meshes

In computer graphics, conforming tetrahedral meshes are commonly used to simulate deformable objects. However, in the context of cutting, the employment of tetrahedral meshes reveals some problems, especially since tetrahedrons cannot easily be subdivided without generating ill-shaped elements that cause numerical problems. Many cutting approaches are a combination of the following four key concepts: *element removal*, *face-splits*, *node snapping*, and *element splits*.

The *element removal* technique basically removes the tetrahedrons that are intersected by the blade. An early approach has been presented by Terzopoulos and Fleisher [33] where the springs intersecting the blade are removed. Similar in spirit is the work of Delingette et al. [3]. Later, Forest et al. [7] proposed a strategy to solve the topological singularities that lead to non-manifoldness of the mesh when elements are blindly removed. However, by solely removing elements, the mesh becomes non-conforming, and the cut surfaces look unnaturally jagged.

This problem can be circumvented by *node snapping*, i.e., aligning the vertices with the cut surface. In doing so, the shape of the elements is changed, and ill-shaped elements could have been created. This issue has been addressed by Steinemann et al. [30] that handle arbitrary triangular blade surfaces and present a node snapping algorithm to avoid slivers.

Face split-approaches take a different way in that they handle the material discontinuity introduced by the cut by duplicating the degrees-of-freedom along the faces in the cut region. Nienhuys et al. [23] report that selecting appropriate split faces is difficult and often ambiguous. In addition, this technique does not prevent jagged cut surfaces.

Approaches that *split elements* along the cut surface are able to reproduce the cut surface exactly, as described in Bielser et al. [1]. Still, their approach suffers from *element count explosion* when multiple cuts are performed. This issue has been addressed by Mor and Kanade [19] who present an algorithm that minimizes the amount of newly generated elements. However, these approaches cannot completely prevent ill-shaped elements.

One solution of the problem is that instead of splitting existing simulation elements along the cut surface, the mesh can also be regenerated locally, thereby avoiding ill-shaped elements. Wojtan et al. [35, 36] remesh the material as soon as the element quality falls below a certain threshold. However, remeshing is inherently expensive, despite recent advantages in efficient mesh generation [15]. This is particularly true in our context where cuts are placed interactively.

The extended finite element method (XFEM) is related to element splitting since conceptually, the elements are split according to the intersection with the blade. Still, instead of topologically splitting the element, the basis functions are enriched to model the material discontinuities. Jeřábková and Kuhlen [12] showed that XFEM can be employed in the context of interactive surgery simulation. Kaufmann et al. [13] proposed to model the discontinuities on a texture-basis.

Recently, a different line of research has been opened by approaches that allow polyhedral simulation elements. In doing so, the elements along the cut surface are intrinsically conforming, as shown by Martin et al. [17]. Their approach is based on the earlier work by Wicke et al. [34] who only support convex element shapes. However, these approaches have the drawback that the update of the elemental stiffness matrices becomes very expensive.

2.2 Cutting non-conforming meshes

Non-conforming meshes are based on the observation that the visual representation of an object can easily be separated from the mechanical representation. The non-conforming mechanical representation can then keep well-shaped elements, while an embedded surface mesh is employed for rendering.

The *virtual node* technique that duplicates the elements along the cut is probably the most popular approach in this context. Introduced by Molino et al. [18], this approach has later been improved by Sifakis et al. [28] to handle arbitrarily many cuts within one single element. For each material component, one instance of the element is created. However, the required per-element connectivity analysis is quadratic in the number of cuts.

In contrast to the previously cited works that employ tetrahedral meshes, we opt for a non-conforming adaptive hexahedral simulation mesh. This is because the topological changes that come with the cutting are intrinsically simpler to realize if a regular mesh is employed.

Adaptive hexahedral simulation meshes based on octree-refinement have been widely employed in the past, for example, by Capell et al. [2], Dequidt et al. [4], Nesme et al. [21], and Seiler et al. [27]. Based on these works, Steinemann et al. [31] have proposed an adaptive geometric deformation model that includes topological changes. They employ the centers of the cubic simulation elements as simulation nodes, which requires a costly extrapolation of the surface vertices. Since we use the corners of the elements as simulation points, we are left with a relatively cheap surface interpolation. In addition to their work, we show how the correspondences between the surface vertices and the simulation elements can be updated efficiently.

Dick et al. [5] recently presented a cutting approach where the surface is interpolated into an adaptive simulation mesh. They intersect the dual graph of the octree mesh with the blade in order to generate the cut surface. As a consequence, the visual representation of the cut surface depends on the resolution of the octree mesh. In contrast, we strictly separate the visual representation from the mechanical representation. That is, by employing our approach, the cut surface can be arbitrarily detailed, while the underlying simulation mesh is automatically refined and split in order to represent the cut mechanically. In addition, we show that by employing unrestricted octree refinement, the adaptation can be realized locally. This is particularly favorable in the interactive context where the number of simulation elements has to be controlled.

The work of Kaufmann et al. [14] bases on adaptive rectangular simulation elements that allow for a more flexible refinement, which is in contrast to octrees. Further, instead of using tetrahedral or hexahedral background meshes, there exist a couple of approaches that do not require any mesh structure to compute the dynamics. Pauly et al. [25] present a complete point-based simulation framework that can handle cutting and fracturing. Subsequently, the approaches of Steinemann et al. [32] and Pietroni et al. [26] improved the performance of topology changing operations significantly. Summarizing, although these approaches perform very well for challenging scenarios, they are less appropriate in the user-interactive context.

3 Overview

When an object is being cut or fractured, the topology of the underlying simulation mesh is changed. Instead of equipping a non-adaptive simulation mesh with element splitting operations, we propose employing an inherently adaptive but non-conforming *octree simulation mesh* $\mathcal{T}$. Then, a high-resolution *surface geometry* $\mathcal{S}$ is embedded into the simulation mesh. While the surface geometry is only employed for the rendering, the mechanical behavior is governed by the simulation mesh [22]. As preprocessing, the adaptive simulation mesh is refined up to a user-defined base resolution in order to reproduce the deformations faithfully. Following the cited work, we employ the finite element method to compute the deformation forces, and an implicit Euler scheme to obtain the dynamic evolution of the simulation nodes. The deformation of the surface geometry is then obtained by barycentrically interpolating the surface vertices from the cubic simulation elements. At this point, it is important to notice that this requires an explicit correspondence between the simulation elements and the embedded surface vertices. This correspondence must be updated upon cutting the surface geometry, as described later.

In addition, we employ a *material graph* $\mathcal{M}$, which can be understood as a spatial approximation of the domain occupied by the high-resolution surface. We rely on a standard tetrahedral mesh generation approach [15] along with quadric surface simplification [8] to obtain $\mathcal{M}$ from $\mathcal{S}$. The material graph allows quickly determining which cells of the octree are covered by material, and which cells are empty. This threefold representation is similar in spirit to the previous work of Seiler et al. [27] that did, however, not consider topological changes. As discussed later, $\mathcal{M}$ is non-adaptive, and cutting is accomplished by removing elements. Consequently, the resolution of $\mathcal{M}$ must be large enough to represent the blade geometry.

In this work, we limit ourselves to volumetric non-progressive cuts. That is, we assume a triangulated volumetric manifold blade geometry $\mathcal{B}$ which is intersected with $\mathcal{S}$. The goal is then to represent the cut both visually and mechanically, i.e., to compute water-tight cut surfaces of $\mathcal{S}$ along the blade, and to re-arrange the degrees-of-freedom (DOFs) of $\mathcal{T}$ in order to model the resulting material discontinuities. The cutting procedure consists of five basic steps:

1. The blade geometry $\mathcal{B}$ is transformed into the material space in order to perform the cutting on the undeformed object geometries. In doing so, the problems related to heavily distorted surface geometries are avoided. We propose a novel approach to compute this back-transformation such that the transformed blade geometry $\mathcal{B}'$ retains a minimum quality, and cuts of strongly deformed objects are enabled.

2. The surface geometry $\mathcal{S}$ in the material space is intersected with the blade geometry $\mathcal{B}'$ in order to obtain the intersection contours, and to fill the resulting cut surfaces. The result is an updated surface geometry $\mathcal{S}'$. We employ a standard surface cutting approach to perform this task.
3. To represent the cut mechanically, the material graph $\mathcal{M}$ is intersected with $\mathcal{B}$ in order to reflect the material connectivity $\mathcal{M}'$ after the cut.
4. To re-arrange the degrees-of-freedom (DOFs) of $\mathcal{T}$ after the cut, a series of octree *refinement operations* is applied until sufficient accuracy is obtained to model the material discontinuities. Based on $\mathcal{M}'$, the faces of the octree simulation mesh are split, and new DOFs are created. The corresponding *connected component analysis* is detailed in Sect. 4.4.
5. As a last step, the correspondence between the vertices of $\mathcal{S}'$, and the elements of $\mathcal{T}'$ has to be re-computed. This can be accomplished efficiently by employing a Z-curve, as shown in Sect. 4.5.

4 Method

In this section, we describe the distinct steps of the cutting procedure in detail. The focus is on the mechanical representation of the cut, i.e., the adaptation of the simulation mesh, which is a main contribution of our work. In addition, we show how to efficiently re-compute the correspondences between $\mathcal{S}'$ and $\mathcal{T}'$. We start by describing the back-transformation of the blade geometry.

4.1 Robust back-transformation

It is a common strategy to perform the cutting in the material space where the cut object is in its resting-state. The reason is that the object geometry might be heavily deformed immediately before the cutting, with elements being inverted or compressed to zero. As a consequence, the triangles of $\mathcal{S}$ might have degenerated, making the computation of intersection contours and the subsequent Delaunay triangulation extremely unstable. By cutting in the material space, these problems can be circumvented elegantly, given that the back-transformation of $\mathcal{B}$ into the object's material space is robust. An additional reason is that a bounding volume hierarchy employed to detect the interference between $\mathcal{B}'$ and $\mathcal{S}$ does not need to be updated in the material space.

We propose an approximative, yet robust back-transformation approach that is able to reproduce cuts of distorted and even folded object geometries, as depicted in Fig. 7. As a first step, we intersect $\mathcal{T}$ in the physical space with $\mathcal{B}$ in order to obtain a set of intersecting simulation elements. Since the intersection is performed on a coarse base resolution of the simulation mesh, this can be done efficiently. This process is done to determine which elements are intersected; consequently, degenerated elements do not cause problems at this stage. We then group the elements into connected components S_i, i.e., we group all elements that are connected by a face. For each S_i, we approximate its deformation tensor $\mathbf{A}_i \in \mathbb{R}^{3\times3}$ in its centroid by employing the shape matching technique [20]. By considering a group S_i of elements instead of single elements in isolation, the approach becomes insensitive to degenerated elements. Then, we back-transform $\mathcal{B}$ into the material space of the connected component by multiplying the vertex positions of $\mathcal{B}$ by $\mathbf{A}_i^{-1}$, resulting in an affinely transformed blade geometry $\mathcal{B}'_i$ per connected component S_i. Since $\mathbf{A}_i$ only captures linear deformations, $\mathcal{B}'_i$ will never be self-intersecting, making the cutting in the material space robust. In addition, $\mathbf{A}_i$ is always invertible, since we blend in the purely rotational part if the condition number increases [11]. The cutting is now performed for each S_i, as described subsequently. Figure 1 illustrates the process. For clarity, we drop the index i in the following text and assume that we have only one connected component for each object.

4.2 Surface mesh cutting

Having obtained the blade geometry $\mathcal{B}'$ in the material space, the next step is to compute the visual representation of the cut, i.e., to intersect the surface mesh $\mathcal{S}$ with $\mathcal{B}'$ to obtain a new water-tight cut surface geometry $\mathcal{S}'$.

Since we employ a standard approach to fulfill this task, we keep it short here and refer the reader to [30]. The surface cutting algorithm starts by computing a set of intersection contours. A constrained Delaunay triangulation fills each contour to form new surface patches. Some bookkeeping is necessary to guarantee compactness of the index space.

4.3 Material graph cutting

Since the material graph is exclusively employed to determine which elements of the simulation mesh are covered by

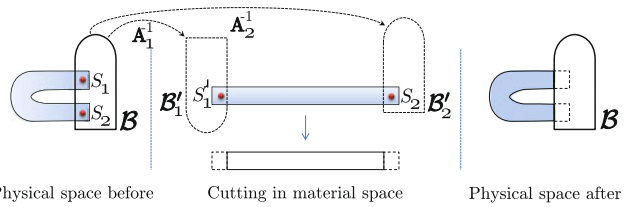


Fig. 1 To enable cutting of folded objects, we intersect the blade $\mathcal{B}$ with the deformed simulation mesh to determine the connected components S_1 and S_2 (*left*). Then, for each component, we compute the deformation tensor $\mathbf{A}_1$ and $\mathbf{A}_2$ and the corresponding deformed blades $\mathcal{B}'_1$ and $\mathcal{B}'_2$. Cutting is then performed in the material space (*middle*). This process results in the correctly cut deformed object (*right*)

material, it is sufficient to simply remove the tetrahedral elements in $\mathcal{M}$ that intersect with $\mathcal{B}'$. Since $\mathcal{M}$ is non-adaptive, the resolution of $\mathcal{M}$ directly relates to the level of detail that can be represented mechanically. The updated material graph that excludes the cut elements is denoted as $\mathcal{M}'$, and subsequently used to determine the connectivity of the simulation mesh.

4.4 Simulation mesh cutting

In this section, we describe the core of our approach, which is the cutting, i.e., the re-arrangement of the DOFs, of the simulation mesh $\mathcal{T}$. Here, the key benefits of employing an adaptive non-conforming octree mesh are revealed.

We employ the octree data structure earlier presented in [27]. This data structure has the advantage that it does not impose a maximum difference between different levels of resolution, i.e., the refinement operation is strictly local. This is particularly beneficial in our case since it allows refining the mesh in the cut region on the fly in order to represent the cut faithfully. This property makes our approach particularly well-suited for interactive applications without a priori knowledge of the cut region. The resulting interface nodes (also denoted as T-junctions) are handled with the hard binding concept [29].

The cutting of $\mathcal{T}$ consists of two sub-steps: First, the resolution of $\mathcal{T}$ is adjusted such that the material discontinuities can be represented. Second, a series of node splits introduces new DOFs and models the material discontinuities.

4.4.1 Mesh refinement

The goal of the first phase is to refine $\mathcal{T}$ such that the required level of detail imposed by the blade geometry $\mathcal{B}'$ can be represented mechanically. Following Grinspun et al.'s terminology, we are looking for an *oracle* [9] indicating which region of the simulation domain requires which resolution. As a rule of thumb, we found that enforcing a simulation element edge length of $l_e = \frac{r}{\sqrt{3}}$, with r being the radius of the inscribed sphere of $\mathcal{B}$, results in sufficient accuracy (see Fig. 2). That is, an octree simulation mesh element is split into eight sub-elements if its length is larger than l_e *and* if it is intersected by $\mathcal{B}'$. The result is an octree mesh with additional DOFs in the cut region in order to represent the cut mechanically.

4.4.2 Material analysis

In the second phase, we analyze the updated material graph $\mathcal{M}'$ in order to determine which simulation elements are empty. In addition, we determine whether a pair of face-adjacent elements is connected; this criterion is given if and only if their common face is intersected by a tetrahedral element of $\mathcal{M}'$. The unrestricted adaptivity of our octree implies that there is a $1:n$ relation between faces of adjacent elements. This has to be considered when processing the faces of the octree.

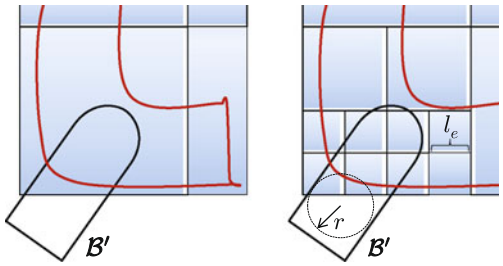


Fig. 2 *Left*: the back-transformed blade surface intersects the *red surface* embedded into the octree grid. *Right*: the elements are refined to meet the criterion $l_e < \frac{r}{\sqrt{2}}$. Then the surface and the material graph are cut

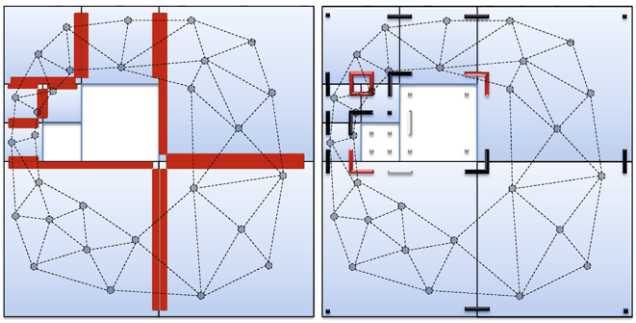


Fig. 3 Illustration of the node splitting approach. As a first step, we determine the intersection of the material graph with the faces of the octree elements, thereby considering the $1:n$ relation between faces in the adaptive case. The *red bars* in the *left* figure indicate faces intersected by material. Based on this information, we compute the actual DOFs of the simulation mesh (*dots*, *bars*, and *angles* in the *right* figure). The *bars* and *angles* indicate which octants share the same DOF. *Red DOFs* are constrained (i.e., T-junctions), while *black DOFs* indicate real DOFs

4.4.3 Node splitting

In the third phase of the simulation mesh cutting, the material discontinuities are introduced. This is accomplished by splitting topological nodes of the octree into multiple DOFs with associated physical properties (see Fig. 3). The resulting algorithm is a variant of the one presented in [23]. In contrast to them, we can draw on the regular structure of the octree, making the algorithm more efficient.

Conceptually, a node i has eight adjacent *octants* that are eventually shared by the same physical element in the adaptive case. We now think of an adjacency graph $\mathcal{G}_i$ per node i linking two octants if they share the same physical element, or if their common face is intersected by material. An edge $(j,k) \in E(\mathcal{G}_i)$ indicates that the octants j and k belong to

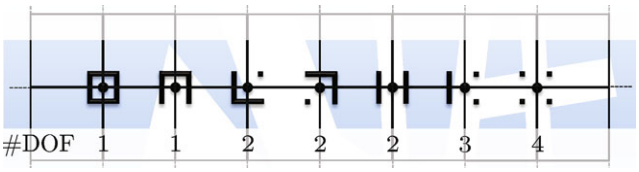


Fig. 4 2D-examples of different material configurations, resulting in different arrangements of the actual DOFs. The *blue regions* indicate the material distribution, and the *black dots*, *bars* and *angles* correspond to the resulting DOFs

the same material component, and thus share the same DOF. The goal of the subsequent *connected component analysis* is now to determine the number of material components per node, which corresponds to the required number of physical DOFs per node (see Fig. 4).

We represent $\mathcal{G}_i$ by its symmetric *adjacency matrix* $\mathbf{B}$, where an entry $[\mathbf{B}]_{jk}$ is 1 if $j = k$ or $(j,k) \in E(\mathcal{G}_i)$, otherwise 0. By recursively computing $\mathbf{B}^8 = ((\mathbf{B}^2)^2)^2$, we obtain the transitive closure of each octant of i. In turn, this allows finally determining which octants belong to the same material component, and thus share a physical DOF. The connected component analysis is performed for each logical node i, resulting in the updated octree simulation mesh $\mathcal{T}'$.

4.4.4 Discussion

We have shown that mechanical cutting can be surprisingly simply realized by employing an octree mesh equipped with a *refinement* and a *node splitting* operation. Notice that in theory the *refinement* operation alone is sufficient to achieve the same result, since a node split can be emulated by performing an additional refinement-step around the node, and marking the node-adjacent elements as empty. However, this approach usually results in more simulation elements and thus it is less efficient than our solution that includes node splitting.

A further benefit is that an octree allows for a structured refinement. By storing the material intersection information on all levels, the cut octree elements can therefore easily be re-merged when the deformation of the object reaches zero. This allows saving computation resources, and makes the method particularly well-suited for time-critical simulations.

At this point, we underline that our approach currently only considers volumetric blade geometries that excise pieces of material. Non-volumetric cuts would require to either extrapolate parts of the embedded surface geometry, or to employ a virtual node technique in the spirit of [18, 28]. We will examine an extension to general cuts in future work.

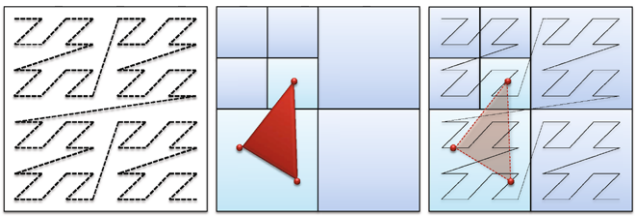


Fig. 5 2D-illustration of the Z-curve employed to encode surface vertex positions. *Left*: Conceptionally, the Z-curve covers the whole domain of the octree, while preserving spatial coherence. *Middle*: A surface triangle embedded in the octree. *Right*: Each vertex of the triangle maps to a unique location within the sorted list of 1D hash values obtained from the Z-curve. All vertices within an octree element can be quickly queried by determining the 1D hash values of its lower left and upper right corner

4.5 Surface embedding

In order to cut the object, we have proposed first processing the surface geometry $\mathcal{S}$, and then adapting the underlying simulation mesh $\mathcal{T}$. Consequently, both the number of vertices of $\mathcal{S}'$ and the number of simulation elements of $\mathcal{T}'$ will have changed, requiring to recompute the correspondences between the vertices of $\mathcal{S}'$, and the cubic elements of $\mathcal{T}'$. Since this recomputation has to be done after each cut, and since the surface geometry has in general a high-resolution, it is crucial that this task can be realized efficiently. In this section, we propose to use a *Z-curve* to store the vertex correspondences per simulation element.

4.5.1 Z-curves

A Z-curve (or space-filling curve) $f : \mathbb{R}^n \to \mathbb{N}$ is a mapping from a multi-dimensional data domain to a one-dimensional *hash value*. Its main advantage is that spatial locality is well preserved, resulting in reduced cache misses and thus a more efficient access. Z-curves are widely used in computer graphics, e.g., to visualize scalar fields [24], to speed-up fluid dynamics [10], or to detect collisions [16].

We employ the Z-curve to map the undeformed vertex positions $\mathbf{x}_i$ of $\mathcal{S}$ into the octree simulation mesh elements (see Fig. 5). More precisely, we keep a sorted list of 1D hash values $f(\mathbf{x}_i)$ of the vertices $\mathbf{x}_i$. For a given octree element, we can easily obtain the range of vertices inside the cubic element by computing the hash values of its lower left and upper right corner. Then, the range of vertices inside the cell is obtained from two $\mathcal{O}(\log n)$ look-ups, where n is the number of surface vertices.

4.5.2 Correspondence construction

In order to initially obtain the sorted list of vertices, we compute for each vertex $\mathbf{x}_i \in \mathcal{S}$ its 1D hash value $f(\mathbf{x}_i)$. This is accomplished by a coordinate-wise bit interleaving of $\mathbf{x}_i$

into a single bit-string. We omit the details here and refer the reader to [24]. By employing a fixed-point representation of the coordinates, this process can be realized very efficiently. Finally, the list of hash values is sorted in $\mathcal{O}(n \log n)$ in order to enable the subsequent range queries.

4.5.3 Correspondence update

Once the sorted list of initial surface vertices has been computed, the insertion of new surface vertices into the list can be done efficiently. If the number n' of newly inserted vertices is smaller than $\frac{n}{\log n - 1}$, we directly insert the new vertices into the sorted list, resulting in a cost of $\mathcal{O}(n'(\log n + \log n'))$. Otherwise, we generate a sorted list of the newly inserted vertices, which is in $\mathcal{O}(n' \log n')$. Then, merging with the sorted list of the vertices of $\mathcal{S}$ is at most $\mathcal{O}(n)$. The benefit of employing a Z-curve reveals if we consider a traditional octree embedding approach where the correspondences are stored in each level of the tree. In that case, updating the correspondences would take $\mathcal{O}(n \log n)$.

5 Evaluation and application

The evaluation of our method is done with a set of illustrative synthetic scenes that convey the characteristics of our method. In addition, we integrate our method into a prototype endoscopic surgery simulator in order to show its benefits in an interactive setting. All experiments have been staged on an Intel Core i7 with 2.66 GHz.

5.1 Evaluation

In this section, we evaluate the performance of our approach under various challenging settings.

5.1.1 Embedding update performance

Our Z-curve based range queries allow for an efficient recomputation of the correspondences between the surface vertices and the simulation elements. To illustrate this, we perform an experiment where material is punched out of a meniscus geometry (see Fig. 10). The surface of the meniscus is discretized into $n = 10k$ vertices, and the deformation is governed by 100 simulation elements. In each cut, 100 vertices are generated on average to fill the cut surfaces. The correspondences are efficiently updated by inserting the newly generated surface vertices into the sorted Z-curve list. As a consequence, the performance of the update does not depend on the size of the octree. To show the benefits of our approach, we compare it to a naive method where the correspondences are explicitly stored in each element of the octree. Consequently, the naive method must update all elements, whose number grows exponentially with the tree size. Figure 6 reveals that the performance of our method stays constant, while the performance of the naive approach grows with the number of octree refinements.

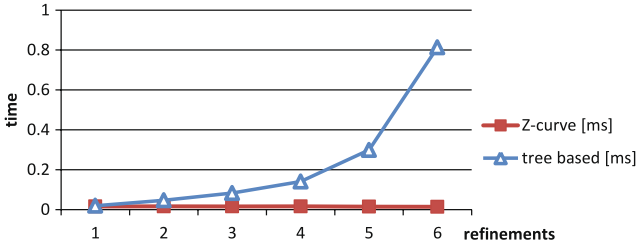


Fig. 6 Diagram of the performance of the vertex-element correspondence update. The performance of the naive approach (*blue curve*) which embeds the surface vertices on all levels of the octree grows with the size of the tree. In contrast, the performance of our approach (*red curve*) stays constant. This is because the Z-curve enables fast querying of the embedded vertices per simulation element without explicitly storing the correspondence per element

5.1.2 Adaptive cutting

We have proposed employing an adaptive octree simulation mesh to deform the object, and representing the cut mechanically. In contrast to a non-adaptive mesh, this has the advantage that the mesh can be refined on-the-fly in order to represent the cut, while large parts of the mesh have a lower resolution. This is even possible in a setting where the cut region is not known a priori. To illustrate the benefits, we perform an experiment where a deformed object is cut into two pieces (see Fig. 7(a)). We simulate the deformable object with our adaptive octree mesh, and with a non-adaptive hexahedron mesh. The blade is discretized into 30 vertices. In Fig. 7(b), the resolution of the adaptive mesh is not sufficient to represent the cut mechanically. In Fig. 7(c), a larger adaptive resolution of 54 elements has been chosen in order to represent the cut mechanically. The time to compute one time step is 5 ms. In contrast, the non-adaptive method has a uniform high resolution, resulting in 144 elements. This leads to a total time of 15 ms to compute one time step.

5.1.3 Cutting deformed geometries

We have proposed to cut in the material space in order to circumvent the challenges that come with ill-shaped geometric primitives [6]. In turn, this requires to back-transform the blade geometry. To illustrate this, we perform an experiment where a strongly deformed object is cut by a complex blade geometry (see Fig. 8(a)). The shape of the punched out material fits the resulting hole of the undeformed object surprisingly well, as highlighted in Fig. 8(b)), although the proposed back-transformation reduces the richness of the blade deformations.

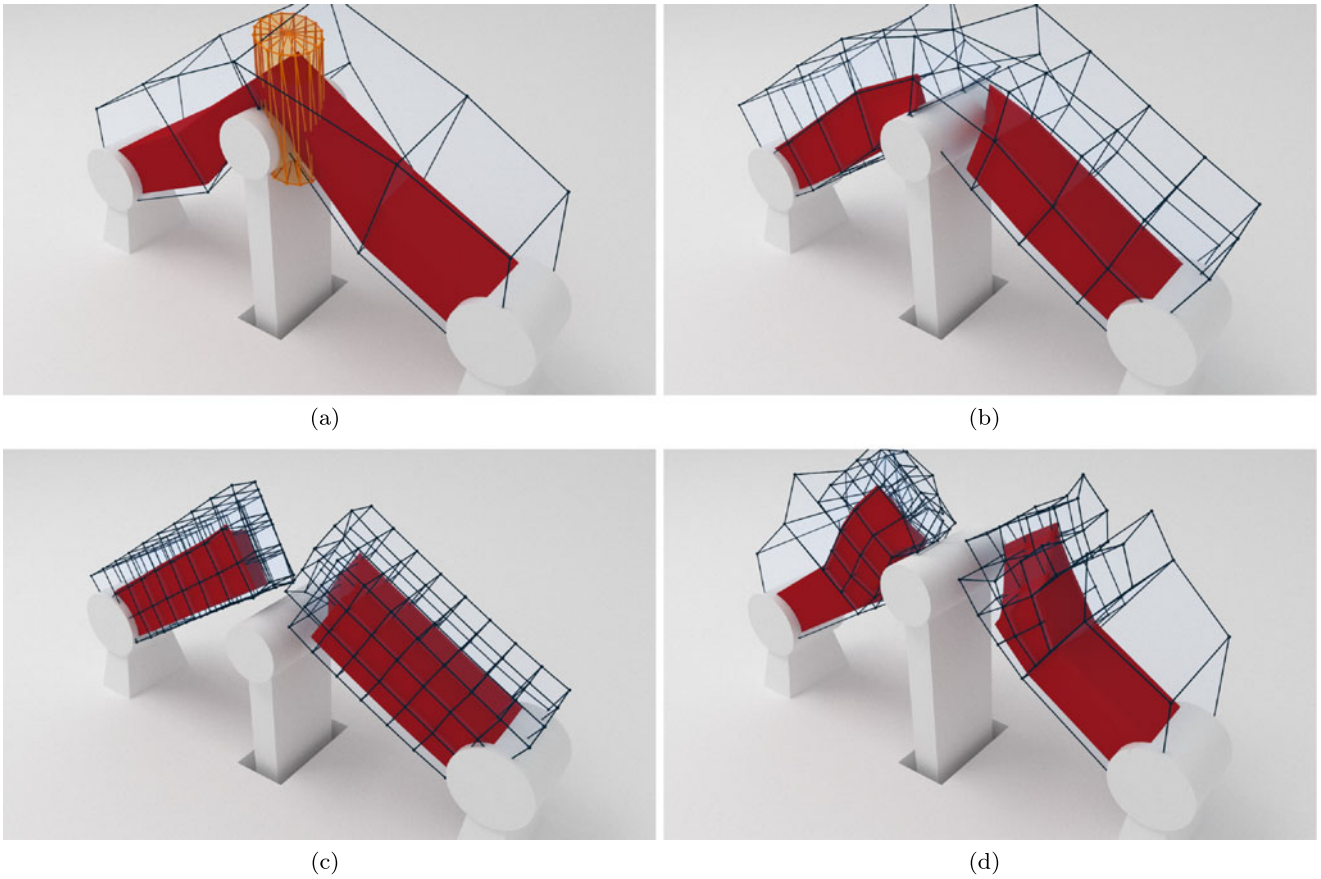


Fig. 7 The red object (**a**) is deformed and cut by a cylindrical blade surface (*in orange*). However, the coarse simulation mesh resolution in (**b**) is not able to resolve the cut. A uniform high-resolution mesh (**c**) is able to resolve the cut, but takes a total of 15 ms to compute one time step. In contrast, our method (**d**) is adaptive and only spends simulation elements where they are needed to resolve cuts, i.e., material that is in close proximity. Computing one time step takes only 5 ms

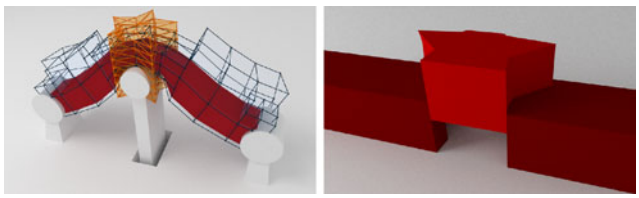


Fig. 8 A strongly deformed object is cut by a complex blade geometry. The punched out material fits well with the resulting hole in the material space of the deformed object

5.1.4 Cutting folded objects

Our method allows for robust cutting even in challenging scenarios that include folded and heavily compressed objects. This is because we extract the back-transformation operator per connected material component. Figure 9 shows an object with a side length of 10 cm which is strongly deformed. Four refinements lead to 256 simulation volumes and some of them are compressed to zero volume by the heavy glass cylinder. A cylindrical blade with a radius of 2 cm driving through an incision of the glass cylinder punches material out of the deformed object (Fig. 9, top). In the resting state of the object (Fig. 9, bottom), it becomes evident that two holes have been punched out, as expected. This is because we have determined two independent material components, each having a different back-transformation operator. The total time for the cut is 172 ms.

5.2 Application

In order to illustrate the applicability of our approach, we integrate it into a prototype of an endoscopic surgery simulator. In this intervention, the surgeon punches material out of a damaged meniscus in order to reduce the pain in the knee joint. The diameter of the punch is 2 mm. Screen-shots of a simulated intervention are depicted in Fig. 10. Cutting the surface geometry of the meniscus takes 60 ms, cutting the material graph takes 22 ms, and cutting the simulation mesh takes 4 ms, resulting in 86 ms to cut the meniscus. Consequently, our method is well-suited for interactive scenarios.

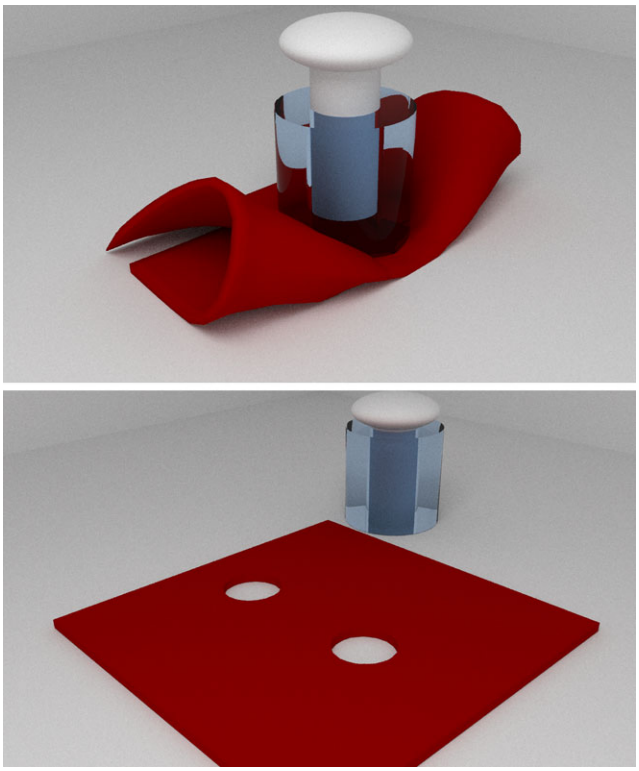


Fig. 9 The deformable object is being compressed by a glass cylinder. A cylindrical blade punches material out of the deformed object. The resulting resting state of the object reveals that two holes have been punched out, as expected. This has been enabled by extracting a back–transformation operator per connected material component

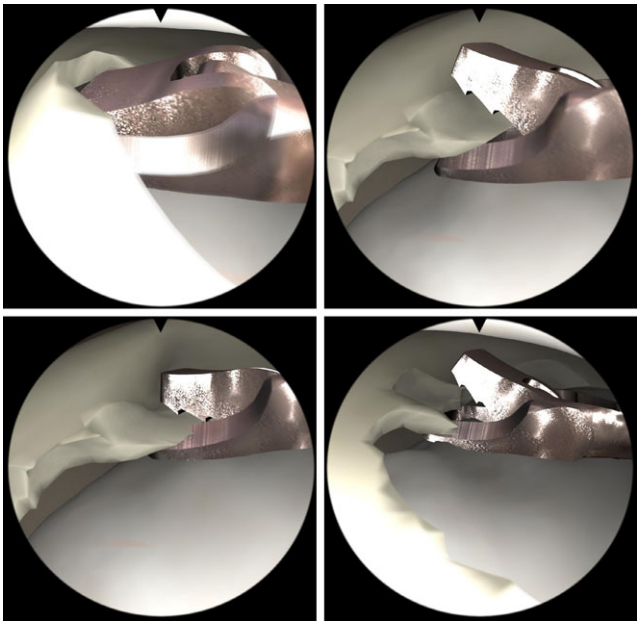


Fig. 10 Screen-shots of a prototype of an endoscopic surgery simulator. We employ our method to simulate the cutting of the meniscus. On average, one cut takes 86 ms, which highlights that our method can be employed in interactive scenarios

6 Conclusion

We have presented an approach for interactively cutting deformable objects. We have illustrated that an octree simulation mesh, equipped with a *refine*- and a *node split*-operation, is well suited for representing the cut mechanically. A high-resolution surface geometry that is embedded into the simulation elements is used to represent the cut visually. By back-transforming the blade geometry into the material space, both the simulation mesh and the surface geometry of the object are cut in the resting state, which improves the robustness of the approach. We have further highlighted that an efficient update of the correspondences between the simulation elements and the vertices of the embedded surface geometry is crucial. To accomplish this, we have proposed employing Z-curves to store and update these correspondences. Various examples have underlined the benefits of our approach.

6.1 Limitations and future work

Currently, our approach only considers non-progressive cutting with volumetric blade geometries. Although many real-life procedures are of this type, we will investigate into overcoming these limitations. In addition, we plan to combine the cutting approach with a scheme to handle collisions and self-collisions.

Acknowledgement This work has been supported by the Swiss CTI project *10534.1 PFLS-LS*.

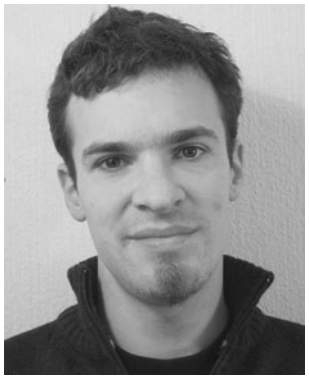
References

1. Bielser, D., Gross, M.H.: Interactive simulation of surgical cuts. In: Proc. of the 8th Pacific Conference on Computer Graphics and Applications, p. 116 (2000)
2. Capell, S., Green, S., Curless, B., Duchamp, T., Popović, Z.: A multiresolution framework for dynamic deformations. In: Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 41–47 (2002)
3. Delingette, H., Cotin, S., Ayache, N.: A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In: Proceedings of the Computer Animation, CA '99, p. 70 (1999)
4. Dequidt, J., Marchal, D., Grisoni, L.: Time-critical animation of deformable solids: Collision detection and deformable objects. Comput. Animat. Virtual Worlds **16**(3-4), 177–187 (2005)
5. Dick, C., Georgii, J., Westermann, R.: A hexahedral multigrid approach for simulating cuts in deformable objects. IEEE Trans. Vis. Comput. Graph. **99**, 1077–2626 (2011)
6. Fierz, B., Spillmann, J., Harders, M.: Stable explicit integration of deformable objects by filtering high modal frequencies. J. WSCG **18**, 81–88 (2010)
7. Forest, C., Delingette, H., Ayache, N.: Removing tetrahedra from a manifold mesh. In: CA '02: Proceedings of the Computer Animation, p. 225 (2002)
8. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '97, pp. 209–216 (1997)

9. Grinspun, E., Krysl, P., Schröder, P.: CHARMS: a simple framework for adaptive simulation. ACM Trans. Graph. (Proc. SIGGRAPH) **21**(3), 281–290 (2002)
10. Ihmsen, M., Akinci, N., Becker, M., Teschner, M.: A parallel SPH implementation on multi-core CPUs. Comput. Graph. Forum **30**, 99–112 (2010)
11. Irving, G., Teran, J., Fedkiw, R.: Invertible finite elements for robust simulation of large deformation. In: Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 131–140 (2004)
12. Jeřábková, L., Kuhlen, T.: Stable cutting of deformable objects in virtual environments using xfem. IEEE Comput. Graph. Appl. **29**(2), 61–71 (2009)
13. Kaufmann, P., Martin, S., Botsch, M., Grinspun, E., Gross, M.: Enrichment textures for detailed cutting of shells. In: ACM SIGGRAPH 2009 papers, SIGGRAPH '09, pp. 50:1–50:10 (2009)
14. Kaufmann, P., Martin, S., Botsch, M., Gross, M.H.: Flexible simulation of deformable models using discontinuous Galerkin fem. Graph. Models **71**(4), 153–167 (2009)
15. Labelle, F., Shewchuk, J.R.: Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. ACM Trans. Graph. (Proc. SIGGRAPH) **26**(3), 57 (2007)
16. Gissler, M.T.M., Ihmsen, M.: Efficient uniform grids for collision handling in medical simulators. In: International Conference on Computer Graphics Theory and Applications (GRAPP) (2011)
17. Martin, S., Kaufmann, P., Botsch, M., Wicke, M., Gross, M.H.: Polyhedral finite elements using harmonic basis functions. Comput. Graph. Forum **27**(5), 1521–1529 (2008)
18. Molino, N., Bao, Z., Fedkiw, R.: A virtual node algorithm for changing mesh topology during simulation. ACM Trans. Graph. **23**(3), 385–392 (2004)
19. Mor, A.B., Kanade, T.: Modifying soft tissue models: Progressive cutting with minimal new element creation. In: MICCAI '00: Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 598–607 (2000)
20. Müller, M., Heidelberger, B., Teschner, M., Gross, M.: Meshless deformations based on shape matching. ACM Trans. Graph. (Proc. SIGGRAPH) **24**(3), 471–478 (2005)
21. Nesme, M., Faure, F., Payan, Y.: Hierarchical multi-resolution finite element model for soft body simulation. In: 2nd Workshop on Computer Assisted Diagnosis and Surgery, March, p. 2006 (2006)
22. Nesme, M., Kry, P.G., Jerábková, L., Faure, F.: Preserving topology and elasticity for embedded deformable models. In: SIGGRAPH '09: ACM SIGGRAPH 2009 papers (2009)
23. Nienhuys, H.-W., van der Stappen, A.F.: A surgery simulation supporting cuts and finite element deformation. In: Medical Image Computing and Computer-Assisted Intervention (MICCAI). Lectures Notes in Computer Science, vol. 2208, pp. 145–152. Springer, Berlin (2001)
24. Pascucci, V., Frank, R.J.: Global static indexing for real-time exploration of very large regular grids. In: Proceedings of the 2001 ACM/IEEE conference on Supercomputing, Supercomputing '01, p. 2-2 (2001)
25. Pauly, M., Keiser, R., Adams, B., Dutré, P., Gross, M., Guibas, L.J.: Meshless animation of fracturing solids. In: ACM SIGGRAPH 2005 Papers, SIGGRAPH '05, pp. 957–964 (2005)
26. Pietroni, N., Ganovelli, F., Cignoni, P., Scopigno, R.: Splitting cubes: a fast and robust technique for virtual cutting. Vis. Comput. **25**, 227–239 (2009)
27. Seiler, M., Spillmann, J., Harders, M.: A threefold representation for the adaptive simulation of embedded deformable objects in contact. J. WSCG **18**(1–3), 89–96 (2010)
28. Sifakis, E., Der, K.G., Fedkiw, R.: Arbitrary cutting of deformable tetrahedralized objects. In: SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 73–80 (2007)
29. Sifakis, E., Shinar, T., Irving, G., Fedkiw, R.: Hybrid simulation of deformable solids. In: Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 81–90 (2007)
30. Steinemann, D., Harders, M., Gross, M., Szekely, G.: Hybrid cutting of deformable solids. In: VR '06: Proceedings of the IEEE Conference on Virtual Reality, pp. 35–42. IEEE Computer Society, Los Alamitos (2006)
31. Steinemann, D., Otaduy, M., Gross, M.: Fast adaptive shape matching deformations. In: Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2008)
32. Steinemann, D., Otaduy, M.A., Gross, M.: Fast arbitrary splitting of deforming objects. In: SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 63–72 (2006)
33. Terzopoulos, D., Fleischer, K.: Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In: SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, pp. 269–278 (1988)
34. Wicke, M., Botsch, M., Gross, M.: A finite element method on convex polyhedra. Comput. Graph. Forum (Proc. Eurographics) **26**(3), 355–364 (2007)
35. Wojtan, C., Thürey, N., Gross, M., Turk, G.: Deforming meshes that split and merge. ACM Trans. Graph. **28**(3), 1–10 (2009)
36. Wojtan, C., Turk, G.: Fast viscoelastic behavior with thin features. ACM Trans. Graph. (Proc. SIGGRAPH) (2008)

Martin Seiler received his M.Sc. degree in computer science from ETH Zürich in 2009. He is now a Ph.D. candidate at the Computer Vision Laboratory at ETH Zürich. His research interests include computer graphics with an emphasis on physically based modeling and simulation.

Denis Steinemann received his Ph.D. degree in Computer Science from ETH Zurich in 2008. He was a member of the Computer Graphics Laboratory headed by Prof. Markus Gross, where his research focused on interactive physically-based simulation. Dr. Steinemann currently works for the ETH spin-off Virtamed. As Chief Scientific Officer, he is in charge of software development and research, working together with the co-authors to push Virtamed's current and future surgical simulators to the highest possible level.



Jonas Spillmann received his Ph.D. degree in Computer Science from University of Freiburg in 2008, where he was a member of Professor Matthias Teschner's research group. Dr. Spillmann is currently a Postdoctoral fellow at Computer Vision Laboratory at the Swiss Federal Institute of Technology, where he is working with Dr. Harders on interactive surgical simulation.

Matthias Harders studied Computer Science with focus on Medical Informatics at the University of Hildesheim, Technical University of Braunschweig, and University of Houston. He completed his doctoral thesis in 2002 and his habilitation in 2007 at ETH Zurich. Currently, he is lecturer and senior researcher at the Computer Vision Lab of ETH, and leader of the Virtual Reality in Medicine Group. His current research concerns surgical simulation and computer haptics. He is a founder of the IEEE RAS/CS Haptics Technical Committee, the IEEE Transactions on Haptics, the EuroHaptics conference and society. He is also the co-founder of the ETH spin-off company VirtaMed.