ARTICLE IN PRESS

Graphical Models xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Splitting meshless deforming objects with explicit surface tracking

Denis Steinemann, Miguel A. Otaduy *, Markus Gross

Computer Graphics Laboratory, ETH Zurich, IFW D29.1, Haldeneggsteig 4, Zurich CH-8092, Switzerland

ARTICLE INFO

Article history:
Received 23 February 2007
Received in revised form 8 December 2008
Accepted 30 December 2008
Available online xxxx

Keywords:
Computer animation
Deformable objects
Topological changes
Cutting
Fracture
Meshless discretization

ABSTRACT

We present a novel algorithm for efficiently splitting deformable solids along arbitrary piecewise linear crack surfaces in cutting and fracture simulations. The algorithm combines a meshless discretization of the deformation field with explicit surface tracking using a triangle mesh. We decompose the splitting operation into a first step where we synthesize crack surfaces, and a second step where we use the newly synthesized surfaces to update the meshless discretization of the deformation field. We present a novel visibility graph for facilitating fast update of shape functions in the meshless discretization. The separation of the splitting operation into two steps, along with our novel visibility graph, enables high flexibility and control over the splitting trajectories, provides fast dynamic update of the meshless discretization, and allows for an easy implementation. As a result, our algorithm is scalable, versatile, and suitable for a large range of applications, from computer animation to interactive medical simulation.

1. Introduction

Many applications of computer graphics exploit the simulation of cutting and fracture of virtual objects. To cite some prominent examples, the growing field of medical simulation has targeted numerous medical interventions that involve cutting of tissue; modeling by virtual carving and sculpting requires elementary operations that produce topological changes; and videogames and special effects for feature films often animate exploding and breaking objects.

The simulation of cutting and fracture of deformable objects encompasses two main aspects [12]: the geometric and topological aspect, concerned with the synthesis of crack surfaces and how these surfaces affect the discretization of the domain in the physically-based simulation; and the mechanical aspect, concerned with the computation of forces, deformations, and crack initiation and propagation. In this paper, we are concerned with the geometric and topological aspect of simulating cutting and fracture, and to that respect, they are analogous operations. However, from the mechanical point of view, and for practical purposes of computer simulation, it is common to distinguish *virtual cutting* [8,15], where crack propagation is explicitly defined by the surface swept by a virtual blade object, from *virtual fracture* [38,33,31], where crack propagation is determined from simulated material stress.

The geometric and topological aspect of cutting and fracture cannot be completely separated from the mechanical aspect, since the choice of discretization method largely influences the requirements of the geometric algorithms for splitting deformable solids. Two major discretization methods have been proposed for simulating deforming objects under splitting, and both methods present advantages and disadvantages, often due to the direct link between the discretization method and the surface representation.

In commonly used finite element methods (FEM), the stability of the simulation strongly depends on the quality of the mesh, which tends to be preserved by introducing constraints in the splitting of mesh elements [8,15,30]. The use of dense meshes diminishes the visual artifacts caused by element splitting constraints, at the cost of slow computations. Similar to our work, the virtual node algorithm [26] focuses on the geometric and topological aspects of cutting and fracture. It enables highly detailed

* Corresponding author. Fax: +41 (0)44 632 1596.
E-mail addresses: deniss@inf.ethz.ch (D. Steinemann), otaduy@inf.ethz.ch (M.A. Otaduy), grossm@inf.ethz.ch (M. Gross).

1524-0703/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.gmod.2008.12.004

surfaces without compromising the stability of the simulation, but it relies on a dense underlying mesh.

Recently, meshless methods have been proposed for simulating fracture in computer graphics [29,34], as they enable lightweight restructuring of the discretization of the models. FEM meshes imply a partitioning of objects for defining interpolation functions and evaluating differential operators, while meshless methods offer a solid mathematical framework for approximating those operators without a need for explicitly partitioning the space. However, unlike ours, previous meshless methods in computer graphics use implicit surface definitions. These definitions limit the control on splitting trajectories and highly complicate the evaluation of boundary conditions. As we show in our simulations (see e.g., Fig. 1), the possibility to split surfaces progressively leads to complicated self-collision situations, which are difficult and expensive to resolve with implicit surfaces.

In this paper, we present a novel approach for the efficient splitting of deformable objects. It combines the advantages of explicit mesh-based surface representations with the flexibility of a meshless discretization of the underlying deformation field. This combination allows us to decouple the synthesis of crack surfaces from the update of the discretization during fracture or cutting. The method is highly scalable while retaining arbitrary and highly controllable split surfaces. Our approach encompasses two main contributions:

- An algorithm for crack generation decomposed into two sequential operations: first, meshing an explicit crack surface from the propagation of cutting or fracture fronts, without restrictions on the topology or trajectory of the front; and second, updating the meshless discretization of the simulation domain, using the readily available explicit crack surface.
- A visibility graph for storing proximity information in the meshless discretization of the simulation domain. The graph is created as a preprocess, and it is locally and efficiently updated at runtime exploiting the explicit crack surfaces. Shape functions of affected simulation nodes are also efficiently updated by using distances along visible paths in the graph. Even though the visibility graph stores connections between simulation nodes, it does not require a partition of the simulation domain as in FEM methods.

Our algorithm is highly scalable, making it suitable for a large range of applications. We have demonstrated it in examples such as an interactive hysteroscopy simulation in Fig. 9, or the cutting simulation shown in Fig. 1.

The rest of the paper is organized as follows. After discussing related work in Section 2, we discuss the simulation of deformable objects using meshless discretizations in Section 3, paying special attention to the handling of the boundary. In Section 4 we discuss the implications of cutting and fracture on the meshless simulation of deformable objects, and we introduce the main steps of our algorithm. Section 5 describes the meshing of explicit arbitrary crack surfaces, while Section 6 describes the visibility graph and the dynamic update of the meshless discretization. In Section 7, we present our simulation results and, in Section 8, we discuss potential limitations and future research directions.

2. Related work

The simulation of breaking objects was introduced to computer graphics almost twenty years ago [38]. Nowadays, finite element methods (FEM) have become a very popular approach for the physically-based simulation of deformable objects that are cut or fractured. O'Brien et al. developed models for brittle [33] and ductile fracture

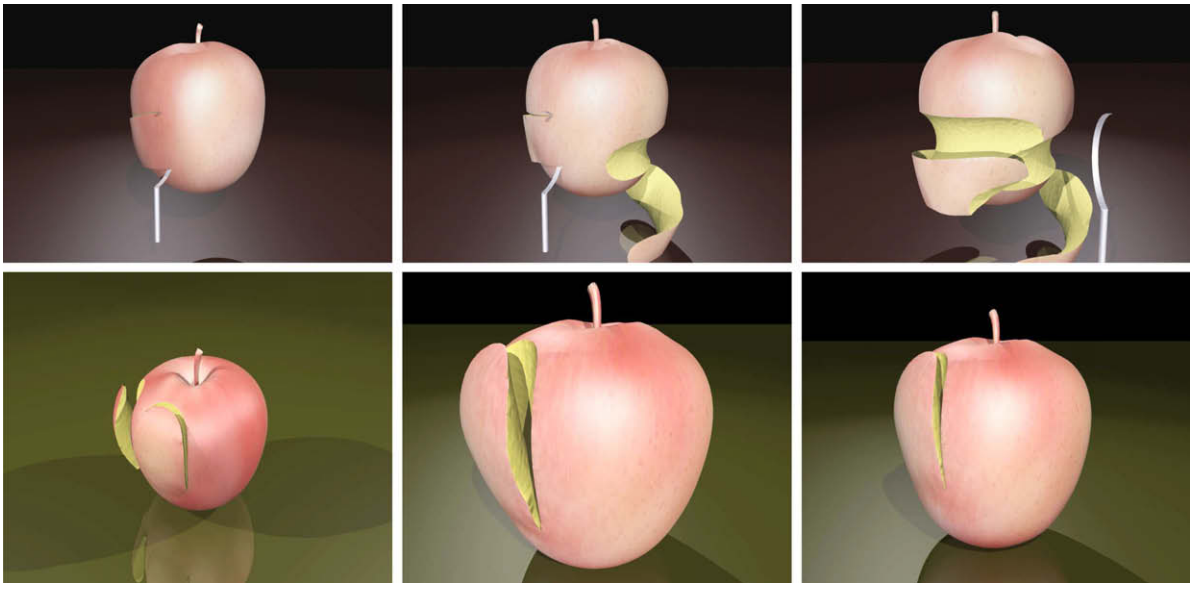

Fig. 1. *Peeling a virtual apple.* As the peeler slices through the apple, the skin deforms and falls off. The apple consists initially of 2121 simulation nodes and 6124 triangles, and it is peeled at 25 fps.

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

[31] based on FEM discretization of continuum mechanics equations, and also demonstrated the generation and propagation of cracks from eigen analysis of the stress tensor.

From the geometric and topological point of view, many approaches have been proposed for dealing with the splitting of FEM mesh elements as the result of cutting or fracture. The splitting of tetrahedral elements by planar cuts can be described by a small set of configurations [8,15], and can also be handled in a progressive manner [28,5]. However, arbitrary decomposition of tetrahedra may produce badly-shaped sub-elements that destabilize the simulation. Hence, many authors have adopted various element decomposition constraints. In situations where speed is a priority, such as virtual cutting in medical simulation, the decomposition constraints include deleting elements as they are crossed by a blade [9], or snapping nodes of the elements to the trajectory swept by the blade [30]. Others have followed hybrid approaches that combine snapping and decomposition [37]. The virtual node algorithm [26] detaches the surface representation from the FEM mesh, and replicates nodes for simulation purposes. It enables arbitrary cuts through mesh elements, as long as one original node falls on each side of the cut. When combined with dense underlying FEM meshes, the virtual node algorithm enables practically arbitrary cutting trajectories, at the price of slow computations. Recently, this algorithm has also been integrated with methods for fracturing rigid materials [6].

In the last decade, new techniques such as the extended finite element method [27] or meshless methods [7] have been designed in the field of computational mechanics for avoiding the volumetric remeshing of FEM in the simulation of fracture. As opposed to FEM, meshless methods do not require a partition of the volume of the deformable objects. This freedom comes at the price of more complex definitions of the shape functions of simulation nodes, but in the simulation of fracture or cutting it offers the important advantages of adaptive resampling and reconfiguration of the shape functions [24,14], without the possible stability problems induced by badly-shaped mesh elements. Meshless methods have previously been proposed for simulating fracture of deformable models in computer graphics as well [29,34]. These approaches employ meshless representations of the crack surfaces, which can be very efficiently sampled as cracks propagate. However, the meshless surface representations are defined implicitly through a moving least squares (MLS) approximation [25], and expensive ad-hoc book-keeping of connectivity is necessary near the crack front to avoid sampling problems and reach the desired surface topology [3]. Similarly, the implicit surface definition imposes a heavy computational burden on the evaluation of boundary conditions, and limits the explicit control of the trajectories of cracks. This could be especially problematic in virtual cutting, where the cut surfaces must align with the virtual blade to avoid artifacts. We also adopt meshless discretization methods for the simulation of cutting and fracture, but we model the crack surfaces explicitly using triangle meshes. We share this strategy with previous approaches in computational mechanics [20,13], but we introduce more efficient methods to handle cracks in geometrically rich objects for visual applications.

Accurate evaluation of surface boundaries is a key aspect with meshless methods, as it governs the reconfiguration of shape functions after cracks propagate. The *visibility criterion* [7] cancels the shape functions if two points are not visible within the object, the *diffraction method* [32] weights the Euclidean distance between two points by their distances to the crack tip, and the *transparency method* [32] adds to the Euclidean distance a factor that depends on the distance to the crack tip. The diffraction and transparency methods were designed for simple 2D cracks with a well-defined crack tip. They have also been used in 3D, with triangle meshes in computational mechanics [20,13], and with meshless surfaces in computer graphics [34], even though they generalize poorly to jagged 3D cracks and are computationally expensive, as discussed by Duflot [13]. Instead, we propose the use of a visibility graph for estimating the distance along fully visible paths, thus handling naturally and efficiently both original concavities and arbitrary crack surfaces.

The concept of visibility graph is associated with the *Euclidean shortest path* (ESP) problem [19]: finding the shortest visible path between two points given a set of polyhedral obstacles. The ESP between two 3D points, if a path exists, is formed by *visibility edges* that connect the two 3D points and points on obstacle edges. Finding the ESP in 3D is *NP*-hard, but several polynomial approximations have been proposed, which sample obstacle edges to construct a 3D visibility graph [11]. Graphs over point clouds are also common to other geometric problems, such as surface reconstruction [18,2,23] or proximity queries [17]. For the construction of our visibility graph, we adapt the Riemannian graph [18].

3. Simulation of meshless deforming objects

In this section, we outline our geometric algorithm for splitting deformable models in cutting or fracture simulation. First, however, we overview the meshless discretization of deformable models, and we discuss the geometric and topological implications of cutting and fracture in the discretization and representation of the models.

3.1. Meshless discretization

Given a 3D solid object with material coordinates $\mathbf{x}$ that parameterize the volume of the object, a displacement field $\mathbf{u}(\mathbf{x})$ defines the deformed positions of internal particles in world coordinates as $\mathbf{x}+\mathbf{u}(\mathbf{x})$. We follow the framework of Müller et al. [29] for modeling dynamic deformations using meshless discretizations. The deformation field is sampled at a discrete set of simulation nodes $P=\{p_i\}$, and it can be approximated at any position in the object as $\mathbf{u}=\sum_i \Phi_i(\mathbf{x})\mathbf{u}_i$, using shape functions Φ_i computed, for example, by moving least squares (MLS) approximation [25]. For possible options in the design of shape functions for meshless methods, we refer to [14,29,34].

Each shape function $\Phi_i(\mathbf{x})$ is weighted by a smoothly decaying kernel $\omega(\mathbf{x},\mathbf{x}_i,r_i)$, based on the support radius

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

r_i of node p_i. The support radius of shape functions should be sufficiently small to adequately discretize gradients [22], and is often estimated based on the distance to the kth nearest simulation node [29]. We define as *neighbors* of a simulation node p_i the nodes for which the value of the kernel function ω_i of p_i, and thus also the shape function Φ_i, is larger than a small cutoff value. Without loss of generality, we can consider as neighbors of p_i those nodes that are closer than the support radius r_i of p_i. In order to account for material discontinuities introduced by cracks and original surface concavities, we define the *material distance* between nodes p_i and p_j of object A, as the ESP between $\mathbf{x}_i$ and $\mathbf{x}_j$, subject to the boundary surface of A. As explained later in Section 6, we approximate the ESP by the graph-based distance using a visibility graph.

3.2. Interpolation of deformations and surface tracking

For animating points on the boundary of the object as it deforms, we follow the approach of Müller et al. [29], using the MLS approximation of the deformation field. Specifically, the displacement $\mathbf{u}$ of a vertex v with material coordinates $\mathbf{x}$ can be computed based on the displacements $\mathbf{u}_i$ of the simulation nodes as

$$\mathbf{u} = \frac{1}{\sum_i \omega(\mathbf{x}, \mathbf{x}_i, r_i)} \sum_i \omega(\mathbf{x}, \mathbf{x}_i, r_i)\left(\mathbf{u}_i + \nabla \mathbf{u}_i^T (\mathbf{x} - \mathbf{x}_i)\right). \quad (1)$$

In the simulation of cutting or fracture, we also require the definition of the inverse mapping, from the world coordinates $\mathbf{x} + \mathbf{u}$ to material coordinates $\mathbf{x}$. For points on the surface, we compute the material coordinates $\mathbf{x}$ by linearizing the mapping function $\mathbf{u}(\mathbf{x})$ over each surface triangle using barycentric coordinates. For points in the interior of an object, we solve for $\mathbf{x}$ in Eq. (1) after estimating kernel weights ω_i in world coordinates.

3.3. Simulation loop with collision handling

In the simulation of splitting objects, robust handling of collisions gains perhaps higher importance than in regular simulations. The reason is that, when material is split, two new surfaces appear in parallel close proximity, which is known to be one of the most challenging scenarios for collision detection and response. Actually, as our algorithm provides higher versatility in the splitting trajectories, the complexity of the contact scenarios increases too. Progressive splitting produces pairs of curved surfaces of the same object in parallel close proximity, which can be regarded as one of the most complex cases of self-collision. Moreover, the splitting front is often a sharp feature on the surface, leading to complex pinching situations [40,41]. Self-collisions are not a major problem in fracture events under tensile stress, as the surfaces move apart naturally, but it is indeed a major problem in e.g., cutting events under compressive stress, such as the apple peeling example in the bottom row of Fig. 1, or the cut Jell-O in Fig. 8.

In our simulations, we separate the handling of inter-object collision and self- or intra-object collision. For inter-object collisions, a point on one object can be queried against the distance function of another object, thus making the situation well-suited for penalty-based methods. After each simulation step, we perform a proximity query using a k-d tree as in [21], and we define penalty-based forces which are added in the next forward dynamics step in the simulation loop described by Müller et al. [29].

For intra-object collisions, penalty-based approaches are not applicable, as a point on an object queried against the distance function of the object itself would always return a zero distance. Hence, we propose a constraint-based approach similar to [4]. At every dynamics simulation step, we perform first a collision-free forward dynamics update, then we identify collision constraints, and we finally add collision response that yields a collision-free configuration. We compute colliding primitive pairs using the spatial hashing acceleration technique of [39], followed by primitive-level continuous collision tests [35]. We formulate velocity constraints for colliding primitive pairs, and we distribute the collision impulses to simulation nodes according to the weights defined in Section 3.1. Given the updated velocities, we recompute positions of simulation nodes and vertices, and we perform an additional sanity-check collision detection step. If some primitive pair still collides, we perform a constraint-correction step by retrieving the associated simulation nodes to the time of collision.

4. Overview of the splitting algorithm

A *crack* is defined as a new surface of a solid object formed when atomic or molecular bonds are broken. In the scope of computer graphics, atomic or molecular bonds are modeled at a macroscopic level by internal forces, and a crack can be defined as a surface across which internal forces are disrupted. When separated by a crack, particles that are adjacent in material coordinates are free to split and move away from each other in world coordinates, hence, as discussed by O'Brien and Hodgins [33], cracks may induce discontinuities in the displacement field $\mathbf{u}(\mathbf{x})$, as shown in Fig. 2.

A crack can be characterized geometrically by two surface sheets joined at a sharp crease, the *splitting front*. As discussed in the introduction, in computer simulation it is common to distinguish between virtual fracture and virtual cutting depending on whether the propagation of splitting fronts is defined from simulated material stress or explicitly from the motion of a virtual blade object. From the geometric and topological point of view, both virtual cutting and fracture involve two main operations: (i) synthesize crack

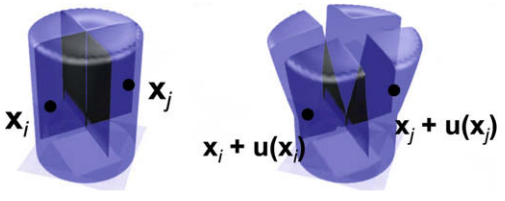

Fig. 2. *Crack surfaces.* A crack defines a discontinuity in the mapping from material coordinates $\mathbf{x}$ (left) to the deformed state $\mathbf{x} + \mathbf{u}(\mathbf{x})$ in world coordinates (right). Shape functions in meshless discretization methods must account for these discontinuities.

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

surfaces and cut the original surface as splitting fronts propagate and (ii) update the discretization of the simulation domain such that internal forces are disrupted across crack surfaces. With meshless discretizations, the second operation reduces to updating the set of neighbors and the shape function of each simulation node according to new material distances that account for crack discontinuities.

As summarized in the introduction, our algorithm for splitting deforming objects handles sequentially the two main geometric and topological operations mentioned above. We dynamically update the meshless discretization exploiting the readily available explicit crack surfaces and a novel visibility graph. The algorithm consists of the following steps:

1. Propagate splitting fronts (Section 5.1).
2. Intersect them with the surface of the object (Section 5.2).
3. Trim and triangulate crack surfaces (Section 5.2).
4. Cut the visibility graph with crack surfaces (Section 6.3).
5. Sample new nodes near crack surfaces (Section 6.4).
6. Update neighbors of simulation nodes (Section 6.3).
7. Update node neighbors of surface vertices (Section 6.5).

5. Progressive cracks

In this section, we present our general algorithm for progressively meshing crack surfaces on deforming objects. First, we discuss the propagation of splitting fronts, the generation of splitting surfaces as fronts are swept, and the handling of topological events produced by intersections of splitting surfaces and the surface of the deforming object. Next, we describe the process of meshing the crack surface, cutting the original surface of the object, and connecting them together. Due to the explicit representation of splitting surfaces, this process can be efficiently performed as a sequence of well-known geometric operations: triangle mesh intersection, trimming, triangulation, and stitching.

5.1. Front propagation

We concentrate here on synthesizing crack surfaces on a 3D object A whose boundary surface is a watertight triangle mesh S_A. A splitting front F is a 3D curve, possibly non-manifold, and possibly closed. We decompose the front into open 1-manifold components, represented by piecewise linear curves. Then, a splitting front F can be described by a sequence of points $\{f_1,\ldots,f_n\}$, as shown in Fig. 3b. As cracks are essentially surfaces that disconnect particles in material coordinates, it is appropriate to define the position of front points f_i in material coordinates $\mathbf{x}_i$. In the simulation, the splitting front is sampled at discrete time steps. We refer as *splitting surface* S_S to the surface swept by the front between two consecutive time steps. We approximate the splitting surface by assuming linear trajectories for the front points, and triangulating the surfaces defined by pairs of front points, as shown in Fig. 3b. If front points move more than a user-defined threshold between time steps, the time step may be decomposed into smaller substeps for better approximation of the splitting surface. We distinguish between the splitting surface S_S and the actual crack surface, which is the portion of the splitting surface trimmed when the front intersects with the surface S_A of the object A. The positions of front points f_i are defined initially in world coordinates, either by sampling a virtual blade object (in virtual cutting), or from eigen analysis of the simulated stress tensor (in virtual fracture). We compute the material coordinates $\mathbf{x}_i$ of front points by inverting the displacement function (1), as described in Section 3.2. When the mapping is not well approximated by a piecewise linear function, the splitting front may be upsampled by adding extra points to the new front F_{k+1} and drawing extra edges to the old front F_k. Similarly, the front may be downsampled by collapsing pairs of points in the new front F_{k+1}. In virtual cutting, a blade may intersect different parts of the surface of an object. Then, points on the blade must be mapped to different front components in material coordinates, depending on the local mapping of each intersection region.

As described by Pauly et al. [34], the topological changes produced during crack propagation can be described through a combination of four elementary events: crack initiation, branching, merging, and termination. We exploit the explicit representation of our splitting surfaces to efficiently detect and handle the various events. Crack initiation, branching, and termination are automatically handled by computing the intersections between the splitting surface S_S and the surface of the object S_A, as shown in Fig. 3, where the new front F_{k+1} branches into two components. Crack merging, on the other hand, is detected as the intersection of splitting surfaces. If splitting surfaces intersect, we force the merging by snapping nodes at the new front(s).

5.2. Trimming and triangulation

Given the surface S_A of an object A, and a newly swept splitting surface S_S, we compute their intersection curves (which we refer to as *trimming curves*) by connecting edge-triangle intersection points (see Fig. 3c). Since the intersections between the old front F_k and S_A are known from the previous frame, we efficiently find a subset of the trimming curves by walking along S_A until we reach intersections between the new front F_{k+1} and S_A. In order to find other possible trimming curves, like the one that produces the front branching event in Fig. 3, we perform intersection queries between edges of S_A and triangles of S_S and vice versa, accelerated by the use of spatial hashing techniques [39]. After intersecting the surfaces, we must decompose the intersecting triangles of both surfaces S_A and S_S, and stitch the resulting patches together at the trimming curves. Note that the crack surface maps to two different surfaces in world coordinates and, in order to maintain a watertight triangle mesh, it must be handled in material coordinates as two collocated meshes with opposite normals.

We handle the meshing of each trimmed triangle T of the surfaces S_A and S_S individually but in a uniform manner. Hereafter we discuss the meshing of trimmed triangles in the splitting surface S_S, but the same procedure is valid for triangles in S_A. Without loss of generality, every inter-

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

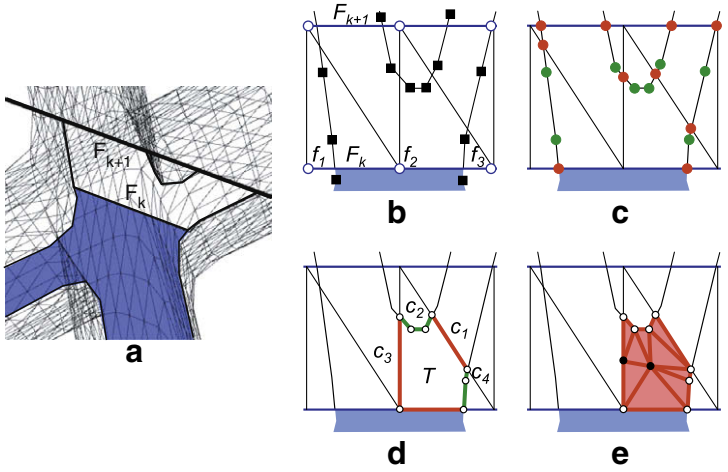

Fig. 3. *Stages of progressive crack synthesis.* From left to right: (a) The X letter being cut by a blade, while the crack surface (in blue) is progressively meshed. (b) The splitting front defined by a blade propagates from F_k to F_{k+1}, and the splitting surface S_S is triangulated. Black squares indicate cross-sections of edges of the surface of the object, S_A. (c) The intersections between S_S and S_A are detected. In red, intersections of edges of S_S and triangles of S_A and, in green, intersections of edges of S_A and triangles of S_S. (d) Each triangle T of the splitting surface S_S is trimmed. The connected alternating sequence of triangle boundary curves (in red) and trimming curves (in green) yields a trimming loop. (e) The newly swept crack surface is meshed by triangulating separately the 2D polygon defined by each trimming loop. Additional points (in black) are inserted to improve the quality of the triangulation.(For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

secting triangle $T \in S_S$ is trimmed by S_A into a set of 2D polygons, possibly with holes. The boundary of each polygon resulting from trimming, which we refer to as *trimming loop*, is a connected sequence of trimming curves $\{c_i\}$ and triangle boundary curves $\{c_j\}$, as shown in Fig. 3d. All trimming loops of a trimmed triangle $T \in S_S$ can be found efficiently by walking between intersection points of its edges with triangles of S_A, alternating steps along trimming curves and triangle boundary curves, until loops are completed and all intersection points are visited. Similarly, inner holes of the polygons enclosed by trimming loops can be found by walking between intersection points of T with edges of S_A.

Once a trimming loop is detected and possible inner holes are identified, the enclosed polygon can be triangulated as shown in Fig. 3e, using fast state-of-the-art 2D polygon triangulation algorithms [36]. Note that, typically, each trimming loop consists of a small O(1) amount of vertices. To reduce robustness problems, before triangulation we collapse pairs of intersection points that lie very close from each other, and after triangulation we try to eliminate near-degenerate triangles, since they could cause problems if further cracks passed through them. Depending on the application, we propose further processing of the crack surfaces for enhanced surface detail control. Local decimation, as previously applied by others [16], can serve to increase performance in interactive cutting simulations. In computer animation of brittle fracture, on the other hand, it is possible to apply further subdivision for obtaining rich jagged surfaces.

6. Visibility graph

In this section, we present a visibility graph for storing proximity information in meshless discretizations. We first define the graph, and we describe how neighbors of simulation nodes can be efficiently determined using a modified Floyd–Warshall algorithm. Next, we describe the handling of cutting and fracture, through the update of the visibility graph using our explicit crack meshes, as well as adaptive resampling. To conclude, we describe an efficient augmentation of the graph in order to store and update node neighbors of surface vertices for the animation of the surface mesh.

6.1. Defining the visibility graph

As introduced in Section 3.1, the evaluation of shape functions for meshless simulation requires knowledge about the material distance between pairs of nodes. Given a pair of nodes $\{p_i, p_j\}$ of an object A, we approximate their material distance as the shortest path distance $d_G(p_i, p_j)$ along a visibility graph G in the interior of A. As mentioned in Section 2, finding exact shortest paths with polyhedral obstacles in 3D is *NP*-hard, therefore we construct an approximate visibility graph G by drawing visibility edges between pairs of nodes.

Specifically, given an object A discretized by a set of nodes P, we initialize the graph G as a Riemannian graph [18] on P, subject to the constraints imposed by the polyhedral boundary of A. The Riemannian graph of P is an undirected graph formed by the Euclidean minimum spanning tree (EMST) of P, augmented with edges $e(p_i, p_j)$ if p_i is one of the k closest nodes of p_j or vice versa (according to the Euclidean distance). In practice, a value of $k = 26$ based on a regular grid-like sampling yields a sufficiently dense visibility graph where Euclidean distance is well approximated by graph-based distance. Accuracy may be traded for lower memory requirements using smaller values of k. After initializing the graph, we perform visibility tests with the initial surface of the object A to remove edges that cross concavities of the boundary, and obtain the set E of

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

valid visibility edges. Using graph-based distances, we can find the node neighbors of each node p_i as those closer than its support radius, and thus evaluate shape functions. Fig. 4 shows the visibility graph and the set of neighbors of a simulation node near a concave region of a surface, before and after the propagation of a crack.

During the simulation of cutting or fracture, for each node p_i we maintain a set of incident visibility edges $\{e_i\}$, and a set of neighbors $\{p_j\}$ entirely defined by distances along the visibility graph. When cracks propagate, we employ the newly synthesized crack surfaces (Section 5) to perform intersection tests with the visibility edges and update the lists of neighbor nodes if visibility edges are cut. For simplicity, we will describe our algorithm assuming regular sampling and a homogeneous support radius $r_{\max}$. In Section 6.4, we describe the handling of adaptive sampling.

6.2. Initialization of neighbors

Given the initial visibility graph G, we must initialize the set of neighbors of each node p_i by finding all nodes that are closer than its support radius. This initialization reduces to computing all-source shortest paths on the visibility graph G. However, we exploit the fact that distances must be computed only w.r.t. nodes closer than the support radius, and we have designed a modified Floyd–Warshall (MFW) algorithm [10] with expected linear time-complexity in the number of nodes.

The standard Floyd–Warshall algorithm stores a matrix D of pairwise distances between nodes in P. The matrix is initialized with the lengths of the edges of the visibility graph G. The algorithm proceeds by looping over all nodes, testing if a path passing through the current node reduces the distance between any other pair of nodes. In our modified algorithm, we only update distances if they are smaller than the support radius $r_{\max}$. As a result, given n regularly sampled simulation nodes with an average number of neighbors m, the matrix D is sparse with $O(m)$ elements per row, and the time-complexity of the algorithm is $O(m^2 n)$. At termination of the algorithm, the ith row of the matrix stores the distances to the neighbors of node p_i.

6.3. Dynamic neighborhood updates

Upon crack propagation, we intersect the triangles of newly synthesized crack surfaces against the set of visibility edges E, as shown in Fig. 4c. This operation can be accelerated using spatial hashing [39]. We define the set of cut edges E_{cut}, and the set of nodes that need to update neighbors P_{update}. The set P_{update} consists of nodes p_i whose distance to an end point of some cut edge $e \in E_{cut}$ is shorter than $r_{\max} - d_e$, where d_e is the length of edge e. Note that, in this case, a shortest path between p_i and some of its neighbors before crack propagation may cross the edge e, and needs to be recomputed. Note also that nodes $p_i \in P_{update}$ only need to update material distances w.r.t. their old neighbors in P_{update}, but the distances w.r.t. neighbors $p_j \notin P_{update}$ remain constant. Fig. 4c shows the set P_{update} as a crack propagates.

To update neighbors of nodes, we remove the cut edges E_c from the visibility graph G, and we execute MFW on the subgraph defined by the nodes P_{update}, as described in the previous section. From the resulting matrix of MFW, we update material distances and neighborhood information among nodes in P_{update}.

6.4. Adaptive sampling

In adaptively sampled objects, the support radius varies across nodes according to the sampling density, as described in Section 3.1. The definition of node neighborhoods based on pairwise distances may suffer from inconsistencies in adjacent regions with disparate sampling densities, as two nodes p_i and p_j may be determined as neighbors, while a node p_k in the shortest path between p_i and p_j is not a neighbor of either of them. One possibility for handling varying sampling densities is to map the

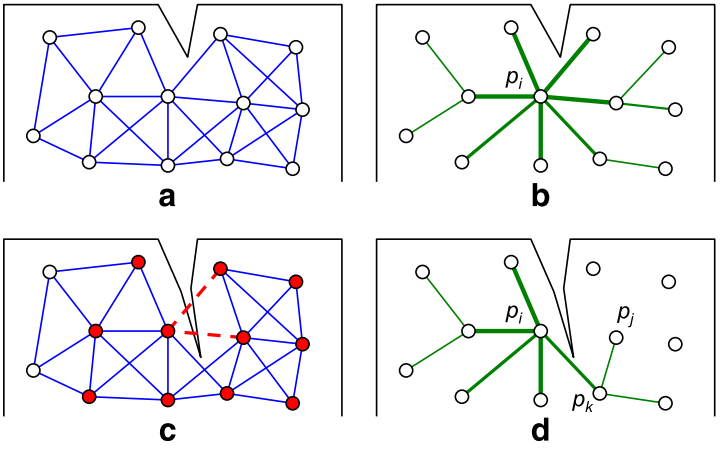

Fig. 4. *Graph and neighborhood updates during crack propagation.* From left to right: (a) Initial visibility graph, accounting for surface concavities. (b) Initial neighbors of a simulation node p_i, where edge width encodes graph-based distance. (c) As a crack cuts edges of the graph (in red), we execute a modified Floyd–Warshall algorithm on the set of nodes P_{update} (in red), in order to update their neighbors. (d) Updated neighbors of node p_i. After the crack cuts the edge (p_i, p_j), the material distance between p_i and p_j is approximated as $\|p_k - p_i\| + \|p_j - p_k\|$ for shape function evaluation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

adaptively sampled domain to a regularly sampled reference domain [22]. Inspired by this idea, we propose a method for warping the visibility graph such that the length of visibility edges is no longer measured in Euclidean space, but in a warped semi-regularly sampled space. We define the length d_e of a visibility edge $e(p_i, p_j)$ by normalizing the Euclidean length w.r.t. the maximum support radius of the nodes p_i and p_j, $d_e = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{\max(r_i, r_j)}$. In this way, the maximum support radius for the execution of the MFW algorithm is normalized as $r_{\max} = 1$. Note that we only use the normalized edge lengths for defining node neighbors, but we resort to unnormalized graph-based shortest path distances for the evaluation of shape functions.

During crack propagation, the interior of an object A must be dynamically resampled in order to conform the sampling density to the dynamically varying boundary. Moreover, in our framework for defining node neighborhoods based on a visibility graph, dynamic resampling fulfills the task of guaranteeing the existence of visible paths inside connected components of the object. We follow the same approach of motion planning algorithms that sample the domain near obstacles in order to guarantee visible paths through narrow passages [1], and we place new nodes by offsetting sample points from the newly swept crack surfaces. Then, we apply the octree-based decomposition proposed by Pauly et al. [34] in order to produce a smooth variation of the sampling density. We connect the set of new nodes P_{new} to the visibility graph by drawing new edges E_{new} to the k closest nodes (subject to boundary constraints). For the computation of neighbor nodes, we simply augment E_{cut} with E_{new}, and we redefine P_{update} accordingly for the execution of the MFW algorithm. Fig. 5a shows the initial sampling of a pumpkin model, with two resolution levels, and Fig. 5b shows the dynamic resampling while the pumpkin is split.

6.5. Reconstruction of the surface mesh

The vertices V of the surface mesh S_A are animated according to the motion of simulation nodes P based on Eq. (1). Every vertex must store a set of neighbor nodes, which may vary dynamically due to crack propagation. One could augment the visibility graph G of nodes with the edges of the mesh S_A to cover all vertices V, and thus dynamically update node neighbors of the vertices. However, typically the sampling density of the surface S_A is considerably higher than the sampling density of nodes P, and this difference incurs in a considerable increase in the cost of executing MFW.

We propose a sparse augmentation G_V of the visibility graph for neighborhood computations of mesh vertices. For each node p_m close to the surface of the object, we set one visibility edge to its closest vertex v_m, which we refer to as *master*. We complete the graph by growing trees of *slaves* from the masters in a breadth-first-search (BFS) manner, as shown in Fig. 5c, until we cover the entire surface. We initialize neighbor nodes by assigning to each vertex v the neighbor nodes of p_m, where p_m is the adjacent node of the master of v. For the computation of the weights in Eq. (1), we use Euclidean distances from the vertices to the nodes, as we found that graph-based distances are not a good estimate in this case.

During crack propagation, we must perform local updates of node neighbors in affected vertices due to two general reasons: (i) edges of the augmented visibility graph G_V are cut and (ii) new vertices and nodes are added to the object. Note that new nodes close to the boundary are connected to surface vertices that become masters. We define the set of *orphan* vertices V_{orphan} as those without a valid master. Vertices may be tagged as orphans because they are newly added, they belong to a subtree of slaves that gets disconnected because a surface edge is cut, or their master gets disconnected because a visibility edge to a node is cut. We perform the dynamic update of node neighbors of surface vertices in two steps. First, we grow trees of slaves in BFS manner to assign a master to all orphans. Second, we update distances and neighbors in trees whose masters are connected to nodes in P_{update} (after being augmented with P_{new}), following the same procedure as at their initialization.

7. Results

We have evaluated our technique on diverse applications, including computer animations that involved cutting operations (Figs. 1, 6 and 8), prescored fracture animation

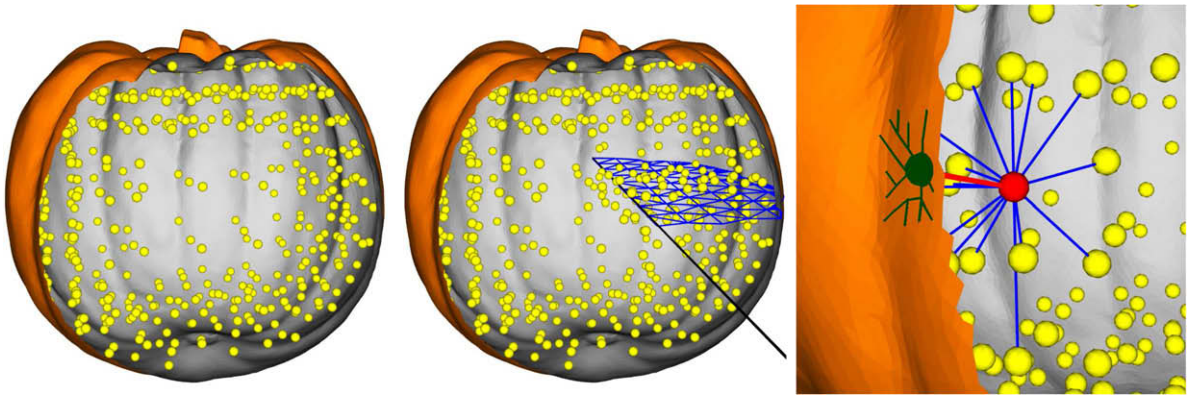

Fig. 5. *Nodes and graph in a pumpkin model.* From left to right: (a) Cross-section of a pumpkin showing the initial adaptive sampling. (b) Resampling of the volume as a crack is meshed. (c) Visibility graph in the locality of a node (in red). In blue, edges to other nodes; in red, edge to a master vertex (in green); and, in green, tree of slave vertices. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

Fig. 6. *Slicing letters*. The blade of the curved wedge cuts the letters smoothly while they deform. The scene consists of a total of 27 K nodes and 100 K-triangles, and the cutting runs at 4 fps.

(Fig. 7), and an interactive surgical simulator (Fig. 9). All demonstrations were executed on a 3.4 GHz Pentium-4 processor PC with 1.0 GB of memory.

Fig. 5b depicts a scenario used for evaluating the performance and scalability of our algorithms. We have halved models of a pumpkin with varying surface and volume sampling densities, following identical cutting trajectories consisting of 12 steps. As shown in Table 1, the time for synthesizing the entire crack surface ranges from 110 ms with a 2.5 K-triangle mesh, to 420 ms with a 10 K-triangle mesh. Note that this trend matches the optimal cost of $O(\sqrt{n})$ for visiting all triangles along a meridian of a regularly sampled spherical surface with n triangles. The time for cutting the visibility graph and updating neighborhood information ranges from 158 ms with a sampling of 2150 nodes, to 303 ms with 8600 nodes. Most importantly, Table 1 demonstrates the scalability of our technique. For the same surface geometry, the throughput of cut triangles per second remains approximately constant independently of the surface sampling density, and the throughput of node neighborhood updates per second remains approximately constant independently of the volume sampling density.

Fig. 1 shows a simulation where an apple is peeled with a curved knife. The apple consists initially of 6124 triangles, 2345 simulation nodes, and 52.4 K visibility edges. During the simulation, the number of triangles increases to 14,504, and the number of nodes to 3149. Splitting operations run at an average of 25 fps, which implies a performance improvement of two orders of magnitude compared to previous meshless approaches in computer graphics [34], thanks to our fast visibility queries and localized updates of graph-based distances. The complete simulation is executed at 3 fps. The animation also demonstrates the

Fig. 7. *Smashed pumpkins*. Our algorithm for synthesizing crack surfaces is used for prescoring fracture.

Table 1
Timings for crack synthesis and graph update. In a cutting scenario similar to Fig. 5b, the throughput of cut triangles per second and node neighborhood updates per second remains almost constant for varying surface and volume sampling densities.

Initial mesh	Final mesh	Graph edges	Meshing crack (ms)	Triangles cut	Triangles per second
2500	3252	94.4 K	110	158	1436
10,000	11,260	95 K	200	276	1380
40,000	42,464	94.9 K	420	523	1245
Initial nodes	Final nodes	Graph edges	Updating graph (ms)	Nodes updated	Updates per second
2150	2764	47.6 K	158	590	3734
4300	5145	95 K	247	847	3429
8600	9574	185.4 K	303	1239	4089

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

effectiveness in capturing arbitrary splitting trajectories. Specifically, the crack surfaces conform accurately to the trajectory of the knife, showing sharp features at the junction with the original surface of the apple, but smooth behavior along the direction of the junction. Moreover, the cut pieces of skin are adaptively sampled, and deform naturally until they fall off. Computer-generated peeling imposes multiple challenges on previous techniques. Previous meshless approaches would have difficulties producing sharp features and evaluating visibility queries for updating shape functions. On the other hand, FEM approaches would require dense meshes and slow computations to provide similar control on the cutting trajectories and the thickness of cut pieces of skin. The bottom row of Fig. 1 shows a different cutting trajectory, where the pieces of apple skin retract instead of falling away. This scenario presents challenging self-collision situations, which are handled robustly, but collision handling becomes the bottleneck and lowers the performance to approximately 2 s/frame.

The letters depicted in Fig. 6 present thin features that require very dense sampling for guaranteeing accurate and stable deformations. The initial scene consists of 27 K simulation nodes and 100 K triangles. With this dense sampling, the simulation runs at an average of 2.5 s/frame, but the splitting operations take only 10% of the computations. This animation demonstrates again the scalability of our technique and its ability to split deforming objects along arbitrary trajectories. We have also explored the application of our technique for prescoring fracture animations of solids. As an example, the pumpkins depicted in Fig. 7 are split into pieces interactively as a preprocess, for later use in a fracture animation.

Compared to implicit surface representations in previous meshless simulation approaches, the use of an explicit surface mesh enables robust self-collision handling, as demonstrated in Fig. 8, which shows spiral cuts being made on a block of Jell-O. Initially, the partially split pieces move apart, and then they clamp together, inducing challenging self-collisions. When the cuts terminate, the four disjoint pieces of Jell-O collapse. The model is initially sampled with 920 nodes and 6200 triangles. The simulation runs at 4.1 s/frame, but the computations are highly dominated by self-collision handling (more than 90%).

One of the major advantages of our technique is that it allows stable arbitrary splitting at very diverse resolutions and model complexities. We have exploited this feature in an interactive simulator of hysteroscopy procedures, where malicious polyps are cut from the uterus cavity, as shown in Fig. 9, using a haptic device as a 3D input tracker. In the simulation, the scalpel is modeled as a sharp curve that can cut in all directions when the blade is active, and there is no scalpel-polyp collision response or handling of self-collisions. The model of the polyp shown in the figure consists of a constant number of 275 simulation nodes. The surface mesh complexity increases from 1334 to 4830 triangles. Splitting operations run at all times at *more* than 45 fps, and the complete simulation runs at 21 fps. Previous techniques for cutting in surgical simulators have often encountered problems with progressive cutting, partial cuts, or changing cutting directions, but all these features are handled stably and efficiently with our technique. Naturally, after reiterative cutting through the same volume, the resolution of the models may grow to levels that cannot be handled interactively.

8. Limitations and future work

In this paper, we have presented a novel algorithm for splitting deformable solids in a fast and arbitrary manner. It adopts meshless discretization methods, and incorporates a novel visibility graph for efficiently updating shape functions in the meshless discretization upon topological changes. We decompose the splitting operation into two steps: meshing crack surfaces, and the update of the graph and the discretization. We have demonstrated the versatility, efficiency, and scalability of our algorithm, and the ability to produce smooth arbitrary cracks in a fast and stable manner. Nevertheless, our algorithm presents several limitations (some of them common to other methods), which set the lines for future improvements.

The quality of the surface mesh degrades when the same region is split multiple times. Local surface remeshing is probably a viable solution, as it does not affect the discretization of the simulation domain in our algorithm. FEM-based techniques, however, would require volumetric remeshing to avoid stability problems. In some applications, it is worth exploring the combination of meshless methods in deforming regions that can potentially be split, with traditional FEM methods in regions that are never split.

Regarding the visibility graph, we are exploring possibilities for better approximation of distances in concave regions by adding a small subset of the surface vertices to the visibility graph, and it is also worth to design means for reducing memory requirements. For handling plastic deformations and the subsequent modifications of the vis-

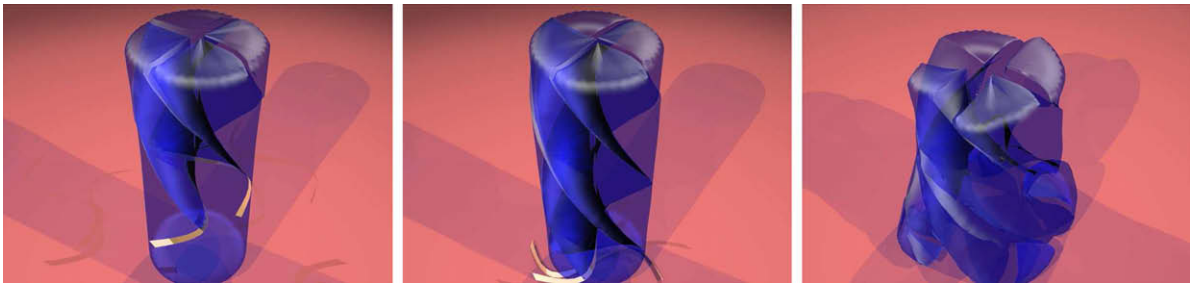

Fig. 8. *Self-Colliding Jell-O.* Spiral cuts induce challenging self-collisions that are handled robustly.

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004

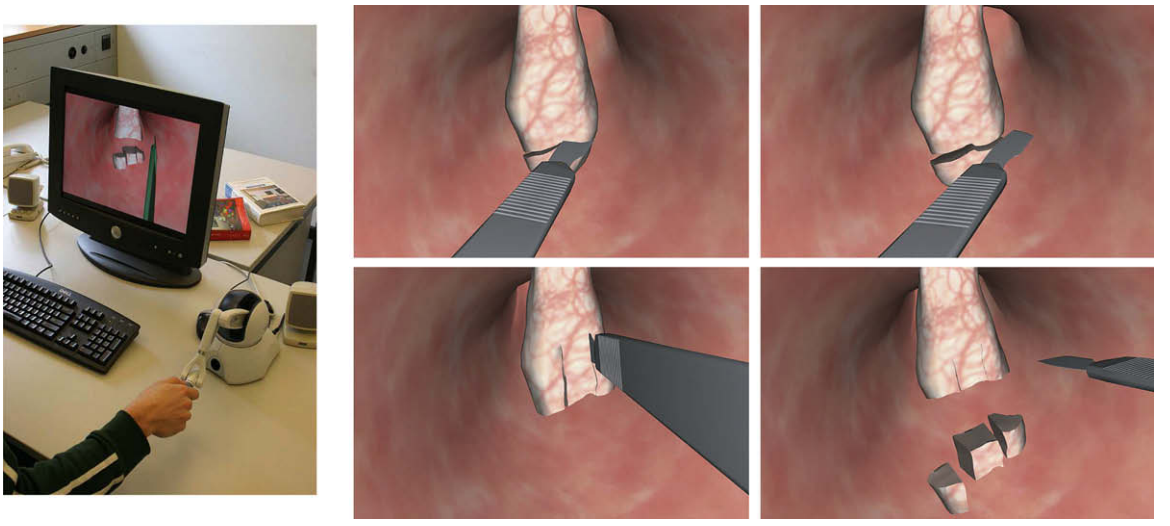

Fig. 9. *Interactive cutting in a surgical simulator*. Cuts produced interactively at *more* than 45 fps.

ibility graph, we are exploring kinetic data structures [17] in combination with the plasticity model of [29].

To conclude, we are currently pursuing the inclusion of our algorithm for splitting deformable models in various other surgical simulation applications. This task will require additional research for fast handling of self-collisions, physically-based cutting and tool-object interaction, and force feedback.

Acknowledgments

The authors would like to thank Richard Keiser, Martin Wicke, Nico Galoppo, Oliver Buechi, Roni Oeschger and Simon Bucheli for their support in this project. This research has been supported by the NCCR Co–Me of the Swiss National Science Foundation.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.gmod.2008.12.004.

References

[1] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, D. Vallejo, OBPRM: an obstacle-based PRM for 3D workspaces, in: Proceedings of the Workshop on Algorithmic Foundations of Robotics, 1998.
[2] Nina Amenta, Marshall Bern, Manolis Kamvysselis, A new voronoi-based surface reconstruction algorithm, in: Proceedings of SIGGRAPH, 1998, pp. 415–421.
[3] Nina Amenta, Yong J. Kil, The domain of a point set surface, in: Proceedings of the Symposium on Point-Based Graphics, 2004, pp. 139–147.
[4] Robert Bridson, Ronald Fedkiw, John Anderson, Robust treatment of collisions, contact and friction for cloth animation, in: Proceedings of ACM SIGGRAPH, 2002.
[5] Daniel Bielser, P. Glardon, Matthias Teschner, Markus Gross, A state machine for real-time cutting of tetrahedral meshes, in: Proceedings of Pacific Graphics, 2003, pp. 377–386.
[6] Z. Bao, J.-M. Hong, J. Teran, R. Fedkiw, Fracturing rigid materials, IEEE TVCG 13 (2) (2007).
[7] T. Belytschko, Y. Lu, L. Gu, Element-free galerkin methods, International Journal for Numerical Methods in Engineering 37 (1994) 229–256.
[8] D. Bielser, V.A. Maiwald, M.H. Gross. Interactive cuts through 3-dimensional soft tissue, in: Proceedings of the Eurographics '99, vol. 18, 1999, pp. C31–C38.
[9] S. Cotin, H. Delingette, N. Ayache, A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation, The Visual Computer 16 (8) (2000) 437–452.
[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, second ed., MIT Press, 1990.
[11] Joonsoo Choi, Jurgen Sellen, Chee-Keng Yap, Approximate euclidean shortest paths in 3-space, International Journal of Computational Geometry and Applications 7 (4) (1997) 271–295.
[12] B.J. Carter, P.A. Wawrzynek, A.R. Ingraffea, Automated 3D crack growth simulation, International Journal for Numerical Methods in Engineering 47 (2000).
[13] M. Duflot, A meshless method with enriched weight functions for three-dimensional crack propagation, International Journal for Numerical Methods in Engineering (2006).
[14] T.P. Fries, H.G. Matthies, Classification and overview of meshfree methods, Technical report, TU Brunswick, Germany, 2004.
[15] F. Ganovelli, P. Cignoni, C. Montani, R. Scopigno, A multiresolution model for soft objects supporting interactive cuts and lacerations, in: Proceedings of the Eurographics, vol. 19, 2000, pp. C271–C282.
[16] F. Ganovelli, P. Cignoni, C. Montani, R. Scopigno, Enabling cuts on multiresolution representation, in: The Visual Computer, 2001, pp. 274–286.
[17] J. Gao, L.J. Guibas, A. Nguyen, Deformable spanners and applications, in: Proceedings of Symposium on Computational Geometry, 2004.
[18] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, Werner Stuetzle, Surface reconstruction from unorganized points, in: Proceedings of SIGGRAPH, 1992, pp. 71–78.
[19] J. Hershberger, S. Suri, An optimal algorithm for Euclidean shortest paths in the plane, SIAM Journal on Computing 28 (6) (1999).
[20] P. Krysl, T. Belytschko, The element-free Galerkin method for dynamic propagation of arbitrary 3-D cracks, International Journal for Numerical Methods in Engineering 44 (1999).
[21] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, M. Gross, Contact handling for deformable point-based objects, Proceedings of Vision, Modeling and Visualization (2004).
[22] P. Koumoutsakos, Multiscale flow simulations using particles, Annual Review of Fluid Mechanics 37 (2005).
[23] J. Klein, G. Zachmann, Proximity graphs for defining surfaces over point clouds, in: Proceedings of Eurographics Symposium on Point-Based Graphics, 2004.
[24] G.R. Liu, Mesh-Free Methods, CRC Press, 2002.
[25] P. Lancaster, K. Salkauskas, Surfaces generated by moving least squares methods, Mathematics of Computation (1981).
[26] Neil Molino, Zhaosheng Bao, Ron Fedkiw, A virtual node algorithm for changing mesh topology during simulation, in: SIGGRAPH 2004 Proceedings, ACM Transactions on Graphics 23 (3) (2004) 385–392.

[27] N. Moës, J. Dolbow, T. Belytschko, A finite element method for crack growth without remeshing, International Journal for Numerical Methods in Engineering 46 (1999).
[28] A.R. Mor, T. Kanade, Modifying soft tissue models: progressive cutting with minimal new element creation, in: CVRMed-Proceedings, vol. 19, 2000, pp. 598–607.
[29] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, M. Alexa, Point based animation of elastic, plastic and melting objects, in: Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 2004.
[30] H.-W. Nienhuys, A.F. van der Stappen, Combining finite element deformation with cutting for surgery simulations, in: Proceedings of the Eurographics '00, 2000, pp. 274–277.
[31] James F. O'Brien, Adam W. Bargteil, Jessica K. Hodgins, Graphical modeling and animation of ductile fracture, in: SIGGRAPH 2002 Proceedings, ACM Transactions on Graphics (2002) 291–294.
[32] D. Organ, M. Fleming, T. Terry, T. Belytschko, Continuous meshless approximations for nonconvex bodies by diffraction and transparency, Computational Mechanics 18 (1996).
[33] James F. O'Brien, Jessica K. Hodgins, Graphical modeling and animation of brittle fracture, in: Computer Graphics, SIGGRAPH 99 Proceedings, ACM, ACM Press/ACM SIGGRAPH, 1999, pp. 287–296.
[34] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus Gross, Leonidas J Guibas, Meshless animation of fracturing solids, in: SIGGRAPH 2005 Proceedings, ACM Transactions on Graphics 24 (3) (2005) 957–964.
[35] Xavier Provot, Collision and self collision handling in cloth model dedicated to design garments, in: Computer Animation and Simulation '97, 1997, pp. 177–189.
[36] Jonathan Richard Shewchuk, Delaunay refinement algorithms for triangular mesh generation, Computational Geometry Theory: and Applications 22 (1–3) (2002).
[37] Denis Steinemann, Matthias Harders, Markus Gross, Gabor Szekely, Hybrid cutting of deformable solids, in: Proceedings of IEEE VR, 2006, pp. 35–42.
[38] Demetri Terzopoulos, Kurt Fleischer, Modeling inelastic deformation: viscolelasticity, plasticity, fracture, in: Computer Graphics, SIGGRAPH 88 Proceedings, ACM Press, 1988, pp. 269–278.
[39] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranerts, M. Gross, Optimized spatial hashing for collision detection of deformable objects, in: Proceedings of Vision, Modeling, Visualization VMV, 2003, pp. 47–54.
[40] Pascal Volino, Nadia Magnenat-Thalmann, Resolving surface collisions through intersection contour minimization, ACM Transactions on Graphics 25 (3) (2006) 1154–1159.
[41] M. Wicke, H. Lanker, M. Gross, Untangling cloth with boundaries, Proceedings of Vision, Modeling and Visualization (2006) 349–356.

Please cite this article in press as: D. Steinemann et al., Splitting meshless deforming objects with explicit surface tracking, Graph. Models (2009), doi:10.1016/j.gmod.2008.12.004