Topology-Adaptive Interface Tracking Using the Deformable Simplicial Complex

MAREK KRZYSZTOF MISZTAL and JAKOB ANDREAS BÆRENTZEN
Technical University of Denmark

We present a novel, topology-adaptive method for deformable interface tracking, called the *Deformable Simplicial Complex* (DSC). In the DSC method, the interface is represented explicitly as a piecewise linear curve (in 2D) or surface (in 3D) which is a part of a discretization (triangulation/tetrahedralization) of the space, such that the interface can be retrieved as a set of faces separating triangles/tetrahedra marked as *inside* from the ones marked as *outside* (so it is also given implicitly). This representation allows robust topological adaptivity and, thanks to the explicit representation of the interface, it suffers only slightly from numerical diffusion. Furthermore, the use of an unstructured grid yields robust adaptive resolution. Also, topology control is simple in this setting. We present the strengths of the method in several examples: simple geometric flows, fluid simulation, point cloud reconstruction, and cut locus construction.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations; geometric algorithms, languages, and systems*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Graphics data structures and data types*

General Terms: Algorithms

Additional Key Words and Phrases: Deformable models, surface tracking, tetrahedral meshes

ACM Reference Format:

Misztal, M. K. and Bærentzen, J. A. 2012. Topology-adaptive interface tracking using the deformable simplicial complex. ACM Trans. Graph. 31, 3, Article 24 (May 2012), 12 pages.
DOI = 10.1145/2167076.2167082.
http://doi.acm.org/10.1145/2167076.2167082.

This work was partially funded by a grant from the Danish Agency for Science, Technology and Innovation.

Authors' addresses: M. K. Misztal and J. A. Bærentzen, Department of Informatics and Mathematical Modelling, Technical University of Denmark, Asmussens Allé, DTU Building 305, DK-2800 Kongens Lyngby, Denmark; email: {mkm, jab}@imm.dtu.dk.

DOI 10.1145/2167076.2167082
http://doi.acm.org/10.1145/2167076.2167082

1. INTRODUCTION

Deformable interfaces are useful in a great many applications, such as fluid dynamics where we need to track the interface between fluids, 3D modeling where the interface is the surface of the object, and image analysis where such methods are used in segmentation and object recognition.

Topological adaptivity—meaning that interface components may split and merge—is crucial to many applications of deformable interface methods, and whenever this is the case, Eulerian methods, such as the well-known *level set method* and its numerous variations are often employed. While many of these methods are powerful, they tend to work by sampling a function whose 0-level set represents the interface on a regular grid. This sampling tends to introduce numerical diffusion and cannot represent fine-scale details. In this project, it has, therefore, been our goal to avoid regular grids altogether but without losing the most important advantages of the Eulerian outlook.

The main contribution of this article is that we introduce the deformable simplicial complex, DSC, a generic method for tracking deformable interfaces which combines many of the advantages of Eulerian and Lagrangian mehods. In particular, the method suffers from little numerical diffusion, allows for transparent topology changes, and provides an explicit triangle mesh (in 3D) representation of the interface which changes only where needed between time steps. We investigate several applications: fluid dynamics, point cloud reconstruction, and cut locus construction.

2. RELATED WORK

Traditionally, methods for deformable interface tracking fall into two categories: *explicit* (*Lagrangian*) and *implicit* (*Eulerian*). Traditional Lagrangian methods, such as active contours or snakes, parametrize the interface and apply the deforming velocity field ($\mathbf{u}$) directly to the interface points ($\mathbf{p}$).

$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = \mathbf{u}(\mathbf{p})$$

This approach leads to trouble once the topology of the interface changes. An efficient collision detection mechanism is needed to detect self-intersections of the interface, and once it happens, costly reparametrization is needed, along with surgical cuts (as in Glimm et al. [1995], although in recent work by Brochu and Bridson [2009] this problem is mitigated by not allowing self-intersections). Those problems do not occur in Eulerian methods, such as the *Level Set Method* (*LSM*, [Osher and Fedkiw 2002]). LSM represents a n-dimensional interface as the 0-level set of a $(n+1)$-dimensional function $\varphi(x_1, \ldots, x_n, x_{n+1})$ (signed distance function is usually the choice), defined on the nodes of a regular grid. The evolution of the interface due to the velocity field $\mathbf{u}$ is then described by the following partial differential equation, also known as that *level set equation*.

$$\frac{\partial \varphi}{\partial t} + \mathbf{u} \cdot \nabla \varphi = 0$$

This approach provides trivial and robust topological adaptivity. However, the basic LSM also exhibits several drawbacks: it is bound to a certain scale, it suffers significant numerical diffusion for features near the sampling rate irrespective of discretization, it does not allow explicit interface representation, it does not support multiple phases, and it relies on calculations in one dimension greater than the interface itself. Some of the drawbacks of the LSM have been successfully mitigated by numerous methods built on top of the LSM mechanism, at the expense of the method's simplicity.

In order to address the disadvantages of purely Lagrangian or Eulerian methods, several hybrid methods emerged. The *Particle Level Set Method* (PLSM) by Enright et al. [2002] uses Lagrangian particles around the interface and advects them with the flow. Characteristics obtained this way are used to compensate for numerical diffusion in basic LSM, which results in significantly lower volume loss and preservation of sharp details. Losasso et al. [2006] used PLSM to successfully simulate multiple interacting fluids.

Some methods are based on triangle meshes, but use voxel grids to resolve topological changes. One of the earliest examples is the topologically adaptive snakes method by McInerney and Terzopoulos [2000]: the interface is represented with a triangle mesh, but a voxel grid is used to resolve topological changes, leading to many of the issues of Eulerian methods and, moreover, movement of the interface is restricted to pure expansion or contraction. In more recent work, Wojtan et al. [2009] use a Lagrangian approach and voxel grid-based method only to locally resolve topological changes (and simplify complex areas). Since the changes are only local and as needed, the method is able to pass the Enright test [Enright et al. 2002] with no visible changes to the rotating geometry. However, the method does require building a signed distance field each time step and the scale of the voxel grid affects the results.

A different, semi-Lagrangian approach has been presented by Bargteil et al. [2006]. Instead of advecting a triangle mesh with the flow, it is reconstructed at every time step but the correspondence between interface points before and after the advection step is retrieved, allowing to track surface characteristics, such as texture coordinates.

Some authors detect intersections using collision detection and resolve topology changes using mesh transformations. Lachaud and Montanvert [1999] propose a method where violation of an edge length constraint indicates an intersection or self-intersection, and a so-called axial transformation is used to create a tunnel if two components merge, and, if a component splits, an annular transformation is used to disconnect the pieces. Zaharescu et al. [2007, 2011] propose a method where self-intersections are removed from an evolving triangle mesh at each time step using a method that greatly resembles Boolean operations on meshes.

The work most directly related to ours is the method due to Pons and Boissonnat [2007a]. The authors proposed a method that is based on a triangle mesh representation of the interface, but once the vertices have been moved, a restricted Delaunay tetrahedralization of the interface is performed. A test is performed on each of the new tetrahedra in order to label them as interior or exterior. If a vertex is found to be shared only by identically labeled tetrahedra, it is removed. This method shares a number of advantages with our method. In particular, it can be extended to multiphase simulations, however, Delaunay remeshing of the point cloud at every time step does not allow one to handle small, sharp features properly, unless the sampling of the surface is very dense. Furthermore, there is no detection of what happens between time steps. Arguably a small object could pass through a thin wall if the time step was not properly tuned, and the precise points where interface collisions occur are not detected. Lastly, it would be difficult to extend their method to do topology control, which is simple with our approach. Another paper by the same authors [Pons and Boissonnat 2007b] presents a 2D version of the algorithm. It shows similarity to our 2D method (no remeshing takes place, edge swaps are performed to improve the quality of the embedding mesh), however, the authors do not utilize Steiner vertex insertion, which is possibly the reason why they failed to generalize the method to 3D. Moreover, the method is limited to domains which are topologically equivalent to a disk.

A recent paper by Wicke et al. [2010] shows a new method for FEM simulation built on top of a kinetic tetrahedral mesh used to discretize an elastoplastic material. The mesh improvement routine bears resemblance to our method but it does not allow changes in the topology of the interface other than fractures, which are produced through fracture simulation built on top of their simulation framework rather than on the mesh improvement level. In particular, their method does not support merging of volumes of the material. In contrast, the DSC method allows all kinds of interface topology changes and, like in the level set method, they are produced *automatically*, requiring no surgical cuts or user interaction.

3. DEFORMABLE SIMPLICIAL COMPLEX

Like the level set method [Osher and Fedkiw 2002], DSC is a method for dealing with deformable interfaces. In the DSC method, the interface is represented explicitly as a set of faces of simplices belonging to a simplicial complex one dimension higher. These simplices belong either to the object or the exterior. Simplices never straddle object boundaries. Thus, in 2D, the computational domain is divided into triangles, and the deforming interface is the set of line segments which divide interior triangles from exterior triangles. Similarly, in 3D, the interface is the set of triangles dividing interior tetrahedra from exterior tetrahedra. Both the 2D and 3D case are illustrated in Figure 1.

The interface deformation is performed by moving the vertices, and this means that the method preserves the advantages of the Lagrangian methods: it suffers little numerical diffusion, and there is an explicit representation of the interface which, furthermore, does not change *gratuitously* between time steps. Moreover, the simplicial complex does not have to be regular meaning that we can allow details of significantly different scale in the same grid (refer to Figure 1 left).

On the other hand, our approach also shares what we perceive as the biggest advantage of the Eulerian methods. Whenever the interface moves, the triangulation is updated to accommodate the change. If two different interface components collide, this change causes them to merge. Thus, topology is allowed to change transparently to the user, although with our method it is also possible to disallow topological changes.

3.1 Method Description

We will first describe the DSC method in 2D. In each time step, we iteratively move points toward their destinations (in arbitrary order, without inverting any triangles), and then improve the mesh until all vertices reach their final destinations. If the displacement of a vertex does not invert any triangle in its star (*1-ring*) then it is performed. Otherwise, it is moved as far as possible along a straight line connecting the old and the new destination. When all vertices have moved, we perform a *mesh improvement* routine and the displacement of the vertices to their final positions is continued in the next substep, taking as many substeps as required to reach the end of the time step.

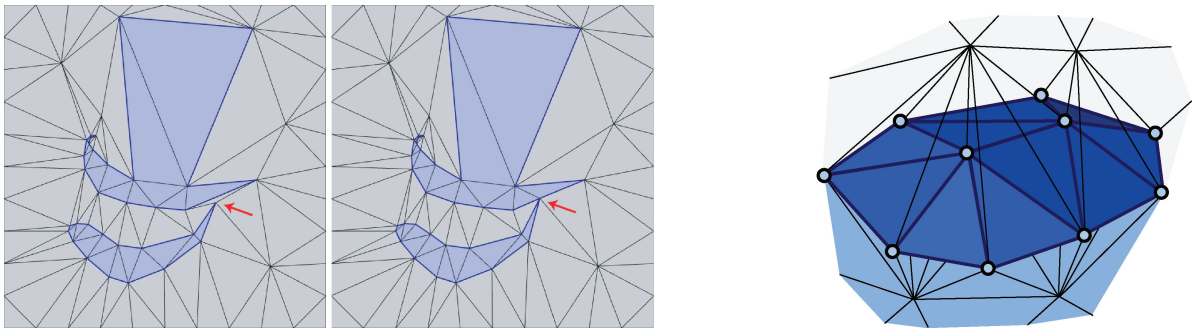


Fig. 1. Interface representation in the deformable simplicial complex (2D on the left, 3D on the right). Exterior triangles (tetrahedra) are light gray, interior – blue. Simplices belonging to the interface (edges and vertices in 2D; faces, edges, and vertices in 3D) are highlighted in dark blue. On the left, the red arrow indicates where a topological change takes place. Note also the difference in scale between the largest and the smallest triangles.

The mesh improvement step aims at improving the quality of the mesh in order to decrease the likelihood of situations where the displacement of a vertex to its final position is not possible and to remove the degenerate triangles created when such situations occur. The following operations are used in the 2D mesh improvement routine.

—*Mesh quality improvement.* Moving vertices tends to introduce degenerate triangles and it almost invariably reduces the quality of the simplices. Consequently, we need to improve the quality, both because poorly shaped triangles are the most likely to be inverted as we move the interface vertices (and hence we cannot displace them in one go) and also because we often want to use the mesh as a computational grid and the quality of elements might significantly affect the accuracy of the results. We perform Laplacian smoothing of the noninterface vertices. Edge flips are performed for all non-Delaunay edges (determined using the empty circumcircle property) in the mesh which are not interface edges.

—*Interface topology changes.* Edge flips are also performed when an interface edge is the longest edge of a *cap* (nearly degenerate triangle with its obtuse angle greater than a certain threshold value θ_{cap}, $\cos\theta_{cap} = -0.99$) and if the vertex of the cap opposite to this edge (*cap tip*) lies on the interface as well; the newly created triangles are labeled according to the other triangle adjacent to the flipped edge (see Figure 1 left).

—*Detail control.* In order to make the mesh improvement effective, we need some *degrees of freedom*, that is, extra noninterface vertices, also known as *Steiner* vertices. We add Steiner vertices by inserting vertices in the barycenters of needles (triangles with one extremely small angle $\theta_{needle} = 0.01\pi$ and longest to shortest edge length ratio greater than 10). This will produce two triangles with even smaller angles, but these are removed by edge flips. However, we must also make sure that we do not introduce too many vertices, otherwise the complexity of the mesh might grow dangerously high. Noninterface edges are removed if this can be done without changing the interface and without introducing degenerate triangles or triangles with a minimum angle smaller than a threshold 0.1π. This also removes a noninterface vertex.

—*Degeneracy removal.* Whenever we move a vertex as far as it is possible without inverting the triangles in the mesh, we introduce degenerate triangles (area of which is close to 0). Hence we have to remove triangles with one of the angles smaller than a certain threshold angle $\theta_{degenerate} = 0.005\pi$ or with area smaller than $a_{degenerate} = 0.5\bar{e}^2 \sin\theta_{degenerate}$ (where $\bar{e}$ is the average edge length in the original interface) through edge collapse of the shortest edge, if it produces a valid mesh.

3.2 3D Deformable Simplicial Complex

The pseudocode of the 3D version of DSC is shown in Algorithms 1–3. It follows the main steps of the 2D algorithm, however, some of the tools useful in the 2D case exhibit mediocre performance in 3D. Laplacian smoothing may produce inverted tetrahedra in 3D despite working quite well in 2D. Likewise, while in the 2D case Delaunay meshes are usually high quality, in 3D they often contain numerous, nearly flat tetrahedra (called *slivers*), which easily get inverted due to a small displacement of their vertices and introduce significant errors in finite element computations [Shewchuk 2002].

To overcome the first difficulty, we use *smart Laplacian smoothing* (moving a vertex towards the barycenter of its neighbors only if it improves mesh quality locally) supported by L. Freitag's *optimization-based smoothing* [Freitag et al. 1995]. Also, Delaunay tetrahedralization is used only to create the initial tetrahedralization of the domain (using TetGen [Si 2004]), which then undergoes mesh improvement (analogous to the mesh improvement algorithms described in Freitag and Ollivier-Gooch [1997] and Klingner and Shewchuk [2007]; see Algorithm 3). Instead of using the Delaunay quality measure of a tetrahedron (minimum solid angle) in our mesh quality improvement operations, we aimed for a quality measure which would penalize both slivers and long, needle-shaped tetrahedra, characterized by near-zero volume despite the fact that the distances between their vertices (edge lengths) might be large. We decided to use the volume-length ratio [Parthasarathy et al. 1994]

$$Q(\sigma) = 6\sqrt{2}\frac{V(t)}{l_{\text{rms}}^3},$$

where $V(t)$ is the oriented volume of a tetrahedron t, and l_{rms} is the average (root-mean-squared) of the lengths of its edges.

$$l_{\text{rms}} = \sqrt{\frac{l_{12}^2 + l_{13}^2 + l_{14}^2 + l_{23}^2 + l_{24}^2 + l_{34}^2}{6}}$$

This function is scale invariant and measures how different a tetrahedron is from the regular tetrahedron (it equals 1 for the regular tetrahedron, and it is close to 0 for slivers and needles). This is of course not the only quality measure having those properties (for other options, see Hansen et al. [2009] and Shewchuk [2002]), we chose it for its simplicity and smoothness.

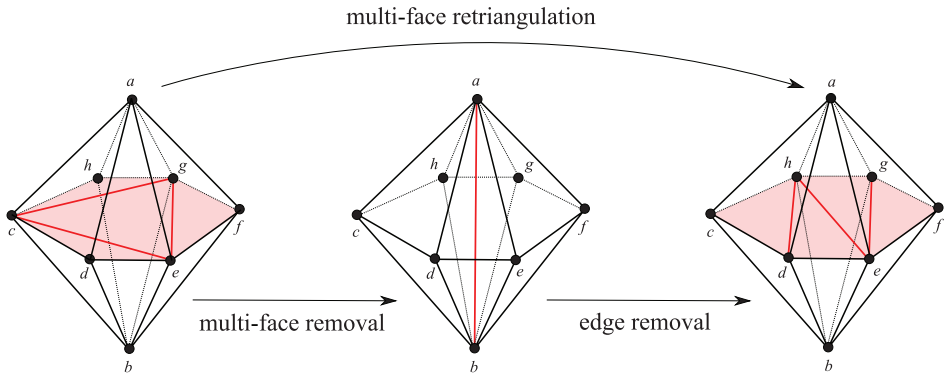


Fig. 2. Topological operations (*swaps*) used for mesh reconnection in 3D DSC. Each operation is performed only if it improves the minimum quality locally.

In 3D we support both volumetric and surface mesh improvement operations.

3.2.1 *Volumetric Mesh Improvement.* Analogously to the 2D case, the tetrahedral mesh used in the 3D DSC needs to be improved after each step of the deformation. The following operations are used in the mesh improvement step.

—*Mesh quality improvement.*
(a) Smart Laplacian smoothing of the noninterface vertices and *optimization-based smoothing*, which moves the vertex v in a way that maximizes the minimal quality of the tetrahedra in its coboundary (this is clearly a nonsmooth optimization problem, for the details see Freitag et al. [1995]). Optimization-based smoothing is computationally expensive and we only use it if smart Laplacian smoothing fails to improve the minimum quality Q in the 1-ring of v above 0.05.
(b) Likewise, we perform topological operations on the 3D meshes (generalizations of the *edge flip* in the 2D case): *edge remove*, *multiface remove*, *multiface retriangulation* (shown in Figure 2, for more detail see Freitag and Ollivier-Gooch [1997], Klingner and Shewchuk [2007], and Misztal et al. [2009]) if they improve minimal quality locally. Since the topological operations are computationally expensive, we only perform them for tetrahedra of quality less than 0.1 (if an application calls for higher-quality meshes, this parameter can be set higher, at the cost of increase in computation time).

—*Interface topology changes.* Topology changes occur when vertices from one component of the interface touch another part of the interface. In this case the two interface parts will be separated only by one or more degenerate tetrahedra. Those tetrahedra are either relabeled (switched from inside to outside, or the other way round) or removed as a part of the degeneracy removal (discussed shortly).
  —*Tetrahedron relabeling.* One can generalize the 2D cap-flip to the 3D case by relabeling a nearly flat tetrahedron t (quality lower than $q_{relabel} = 0.01$) trapped between two parts of an interface. This is a case when the largest face f of t lies on the interface and the vertex v opposite to f also lies on the interface and its orthogonal projection onto f lies within f's hull. In order to improve the interface mesh we also relabel t if $Q(t) < \frac{q_{relabel}}{2}$ and the change of label would decrease the total area of the interface. Since, in some way, relabeling acts like "combinatorial surface tension", $q_{relabel}$ might be relaxed in some applications, such as fluid simulation.
  —*Detail Control.* Noninterface edges are collapsed if their endpoints do not lie on the interface or the DSC mesh boundary and if the collapse does not produce degenerate, inverted, or low-quality (less than a threshold $q_{collapse} = 0.25$ in our experiments) tetrahedra. Edges which do not belong to the interface are split if they either connect the domain boundary with the interface or two interface vertices and if they are longer than a threshold $l_{split} = 4\bar{e}$, where $\bar{e}$ is the average edge length in the original interface mesh. This ensures that the interface has enough freedom to move. The aforementioned parameters are somewhat arbitrary and could be modified. The values we have selected are rather aggressive, aimed at avoiding increase in the complexity of the mesh.

—*Degeneracy removal.* The topology of the interface changes when vertices from one part of the interface collide with another part of the interface. This introduces degenerate tetrahedra, and when these are removed as described in the following the topology changes occur.
  (a) *Tetrahedra*. If a tetrahedron's vertices are nearly coplanar (e.g., if the tetrahedron's quality is smaller than a value $q_{min} = 5 \cdot 10^{-3}$) its largest face is found and a tetrahedron removal strategy is chosen accordingly to the position of its opposite vertex by "flattening" the tetrahedron and replacing it with a set of 2, 3, or 4 faces.
  (b) *Faces*. If a face contains an angle smaller than θ_{min}, $\cos_{min} = 0.999$, it can be either a *cap* or a *needle*. If the ratio of the longest edge to the second longest edge in the face is greater than 1.03, the longest edge is split at the projection of the vertex opposite to it and the edge connecting the new vertex and the cap tip is collapsed; otherwise, the face is a needle and the edge opposite to the smallest angle is collapsed (both collapses are performed if they produce a valid mesh).
  (c) *Edges*. Finally, we collapse edges shorter than a threshold value $l_{min} = 0.5e_{min}$ (where e_{min} is the length of the shortest edge in the input interface mesh) if it produces a valid mesh.

3.2.2 *Surface Mesh Improvement.* The success of a Lagrangian method often depends on the quality of the triangulation of the interface [Brochu and Bridson 2009], especially when the velocity field computation depends on the geometry of the interface. Moreover, the quality of the tetrahedralization benefits from better shaped triangles in the interface [Chew 1997; Shewchuk 1998]. However, even if we start with a high-quality triangular mesh, it might quickly deteriorate as we advect the interface. To prevent this we include

ALGORITHM 1: 3DDSC (M, $\mathbf{u}$)
{M is a tetrahedral mesh conforming to the initial interface}
{$\mathbf{u}$ is a velocity function for the interface vertices}

1: $t \Leftarrow 0$
2: **while** $t < T$ **do**
3: **for** each marked (interface) vertex $\mathbf{p}_i$ **do**
4: compute final vertex position $\tilde{\mathbf{p}}_i \Leftarrow \mathbf{p}_i + \mathbf{u}(\mathbf{p}_i) \cdot \Delta t$
5: **end for**
6: flip interface edges
7: $complete \Leftarrow$ **false**
8: $counter \Leftarrow 0$
9: **while not** $complete$ **do**
10: $counter \Leftarrow counter + 1$
11: $complete \Leftarrow$ MOVEVERTICESSTEP(M, $\{\tilde{\mathbf{p}}_i\}$)
12: relabel valid degenerate tetrahedra
13: $T \leftarrow$ set of tetrahedra adjacent to vertices displaced in the previous step
14: smooth all non-interface vertices in T
15: **if** $complete$ **and** $counter \equiv 0 \pmod 4$ **then**
16: IMPROVEMESHSTEP(M, T)
17: **else**
18: smooth all non-interface vertices
19: reconnection step (lines 3-10 in Algorithm 3)
20: remove degenerate tetrahedra
21: remove degenerate faces
22: remove degenerate edges
23: **end if**
24: **end while**
25: $t \Leftarrow t + \Delta t$ {Δt is a time-step}
26: **end while**

ALGORITHM 2: MOVEVERTICESSTEP (M, $\{\tilde{\mathbf{p}}_i\}$)
{$\{\tilde{\mathbf{p}}_i\}$ is a set of the new positions of the interface vertices}

1: $complete \Leftarrow$ **true**
2: **for** each marked vertex $\mathbf{p}_i$ **do**
3: **if** $\|\mathbf{p}_i - \tilde{\mathbf{p}}_i\| > 0$ **then**
4: compute the intersection t_0 of the ray $\mathbf{p}_i + t \cdot (\tilde{\mathbf{p}}_i - \mathbf{p}_i)$ with the *link* of the vertex $\mathbf{p}_i$
5: **if** $t_0 > 1$ **then**
6: $\mathbf{p}_i \Leftarrow \tilde{\mathbf{p}}_i$
7: **else**
8: move the vertex $\mathbf{p}_i$ to the intersection point $\mathbf{p}_i + t_0 \cdot (\tilde{\mathbf{p}}_i - \mathbf{p}_i)$
9: $complete \Leftarrow$ **False**
10: **end if**
11: **end if**
12: **end for**
13: **return** $complete$

several interface improvement operations which do not significantly alter the geometry of the interface.

—*Null-space smoothing* [Jiao 2007]: moving each interface, manifold vertex only in the null space of its local quadric metric tensor. This way it does not change the geometry of the interface mesh. However, in our implementation we allow *slight* changes to the geometry in smooth regions (we treat the neighborhood of a vertex as flat, if the ratio between the second greatest eigenvalue of the quadric metric tensor to the greatest eigenvalue is smaller than the *aggressiveness factor* of 0.025).

ALGORITHM 3: IMPROVEMESHSTEP (M, T)

1: split long non-interface edges
2: smooth all non-interface vertices in T
3: **for** each tetrahedron $t \in T$, such that $Q(t) < 0.1$ **do**
4: **for** each non-interface face f of t **do**
5: attempt to remove f using MFRT and MFR
6: **end for**
7: **for** each non-interface edge e of t **do**
8: attempt to remove e using edge removal
9: **end for**
10: **end for**
11: collapse valid non-interface edges

—*Edge flip* of the interface, nonmanifold edge e: performed if e's adjacent interface triangles do not fulfill the 2D Delaunay criterion, if e is not a feature edge (measured by the angle θ_{flip} between the normals to its adjacent faces, $\theta_{flip} < 5^\circ$). Interface edge flip is in fact a restricted case of the edge removal (swap) operation in the embedding tetrahedral mesh.

—*Edge split* for edges longer than a selected threshold value (usually 2 times the average edge length in the original surface mesh).

The aforementioned operations are an optional part of the 3D DSC algorithm. They are all used in the fluid simulation method.

In order to quantify the error introduced by our surface mesh improvement procedures, we have performed an experiment, in which we applied 100 iterations of our surface mesh improvement procedures (with null-space smoothing aggressiveness factors of 0.2, 0.025, and 0.003125) to a Stanford dragon model. The results of our experiment are presented in Table I. For our choice of aggressiveness factor, the final mesh shows significant triangle quality improvement while there is no visible volume or detail loss.

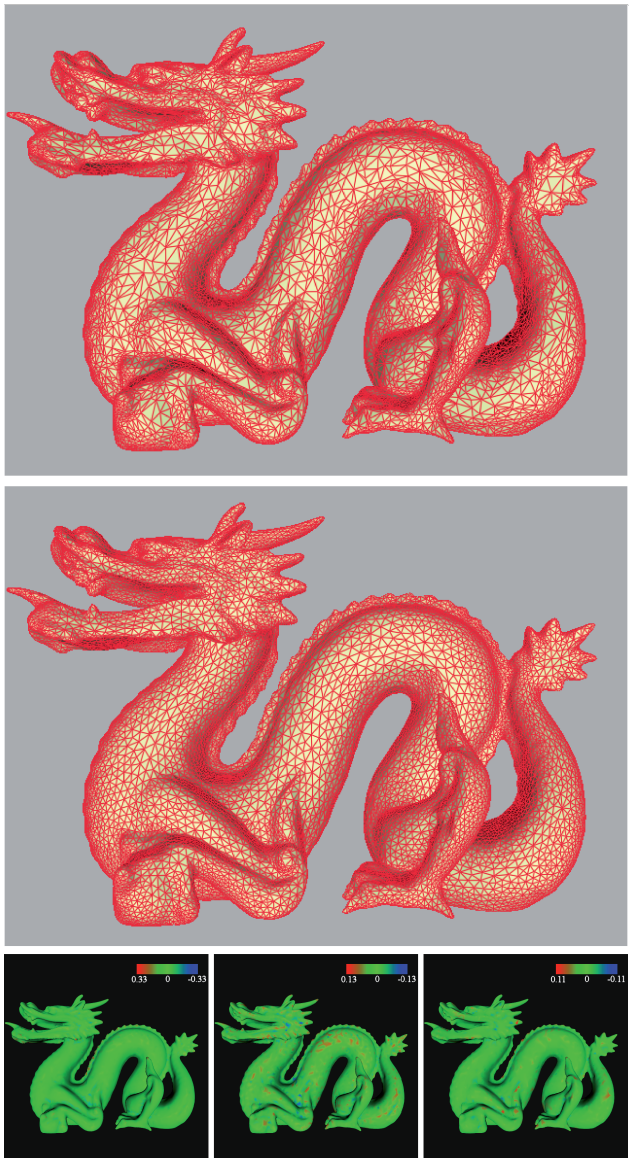
3.3 Implementation

The 2D DSC implementation is built on top of the half-edge data structure [Mäntylä 1988] available in the GEL library [Bærentzen 2010].

The 3D DSC implementation relies on our own C++ implementation of the Incidence Simplicial data structure for 3D simplicial complexes by de Floriani et al. [2010]; we have also implemented all mesh operations used in the algorithm. In its current form, the 3D DSC algorithm requires a triangle mesh as an input. This triangle mesh is then normalized (its barycenter is moved to $(0, 0, 0)$ and all vertices are scaled down by $2.5 \cdot r$, where r is the radius of the minimal sphere centered at $(0, 0, 0)$, containing the whole interface mesh) and tetrahedralized on the inside and on the outside using TetGen [Si 2004] (the outside mesh is bounded by a sparsly subdivided $(-1, -1, -1) \times (1, 1, 1)$ box).

We are currently working on making the code available to the wider public.

Table I. Surface Mesh Improvement Results



aggressiveness	max. distance	avg. distance	volume loss
$2 \cdot 10^{-1}$	0.332	0.0221	$8.47 \cdot 10^{-3}$ %
$2.5 \cdot 10^{-2}$	0.132	0.0138	$7.76 \cdot 10^{-3}$ %
$3.125 \cdot 10^{-3}$	0.106	0.00375	$8.77 \cdot 10^{-3}$ %

The first image from the top shows the original mesh, the second one, the same mesh after 100 iterations of surface mesh improvement with the aggressiveness factor of 0.025. The third row shows the signed distances (relative to the average edge length in the original mesh) of vertices in improved meshes from the original mesh, from left to right: 100 iterations with aggressiveness of 0.2, 0.025, and 0.003125. The table at the bottom shows the maximum absolute distance, the average absolute distance and volume loss for each of the experiments (the dragon model is courtesy of the Stanford 3D scanning repository http://graphics.stanford.edu/data/3Dscanrep/).

4. APPLICATIONS

4.1 Simple Geometric Flows

The benchmark tests for deformable models are simple geometric flows: rotation, mean curvature flow, offsetting (motion in the normal direction), and the Enright test [Enright et al. 2002; Osher and Fedkiw 2002].

4.1.1 *Rotation.* Rotation with an angular velocity ω around an axis $\mathbf{e}$ is performed by multiplying the interface vertices' positions by a rotation matrix $\mathbf{R}(\mathbf{e}, \Delta\theta)$ (where $\Delta\theta = \omega \cdot \Delta t$, Δt is the time step) in every time step. This simple geometric flow poses a challenge for the level set method, as the numerical diffusion causes the interface to lose sharp details and volume [Osher and Fedkiw 2002] and requires elaborate fixes to prevent it. The deformable simplicial complex, like Lagrangian methods, does not suffer such problems, as can be seen in Figure 3.

4.1.2 *Mean Curvature Flow.* We compute the mean curvature of the interface using the *cotangent formula* [Pinkall and Polthier 1993]. The results are shown in Figure 4.

4.1.3 *Offsetting.* Motion in the normal direction is performed using *face offsetting* [Jiao 2007]. Displacing the triangle mesh vertices by constant distance in the normal direction gives incorrect results. Instead, we offset the planes containing faces of the triangular mesh in the normal direction and find new vertices' positions from the intersections of these planes. The results are presented in Figure 5.

4.1.4 *The Enright Test.* Enright deformation [Enright et al. 2002] is given the following velocity field:

$$\mathbf{u}(x, y, z) = \begin{pmatrix} 2\cos(\pi t)\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z) \\ -\cos(\pi t)\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z) \\ -\cos(\pi t)\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z) \end{pmatrix},$$

for $t \in (0, 2)$. Since it brings the interface back to the original shape at $t = 1$ and $t = 2$, it is commonly used to evaluate the amount of numerical diffusion (volume and detail loss) introduced by the interface tracking method. The results of the Enright test in the DSC framework (with null-space smoothing switched off), presented in Figure 6, show minimal numerical diffusion.

4.2 Cut Locus Construction

We utilize the possibility to preserve the topology of a front and to give the domain other topology than that of a disk in our cut locus construction method for Riemannian 2-manifolds, described in detail in Misztal et al. [2011]. The *cut locus* of a point $\mathbf{p}$ in a manifold $(\mathcal{M}, \mathbf{g})$ is essentially a set of all those points which are connected to $\mathbf{p}$ by more than one minimizing geodesic [Sakai 1996]. This can also be seen as a set of those points, where equidistance circles centered at $\mathbf{p}$ form cusps and self-intersect. Our algorithm utilizes the second approach. We advect the circle centered at $\mathbf{p}$ along the geodesics connecting its points with $\mathbf{p}$ with constant speed. The advection takes place in the coordinate chart (parametric domain, (u, v)-space) discretized using 2D deformable simplicial complex and the 2D DSC mesh is given torus topology. Whenever the front is about to collide with itself (when this happens, degenerate triangles appear in the mesh) the interface stops, as shown in Figure 8. Some examples of cut locus construction results for tori are presented in Figure 7.

4.3 Point Cloud Reconstruction

In order to demonstrate adaptive resolution of the DSC method, we implemented a simple, topology-adaptive point cloud reconstruction method. Our method is inspired by the algorithm proposed by Hoppe et al. [1993], which is not topology adaptive. The initial interface is the triangulation of a bounding sphere of the point cloud. In every time step we compute a new desired configuration for each face f.

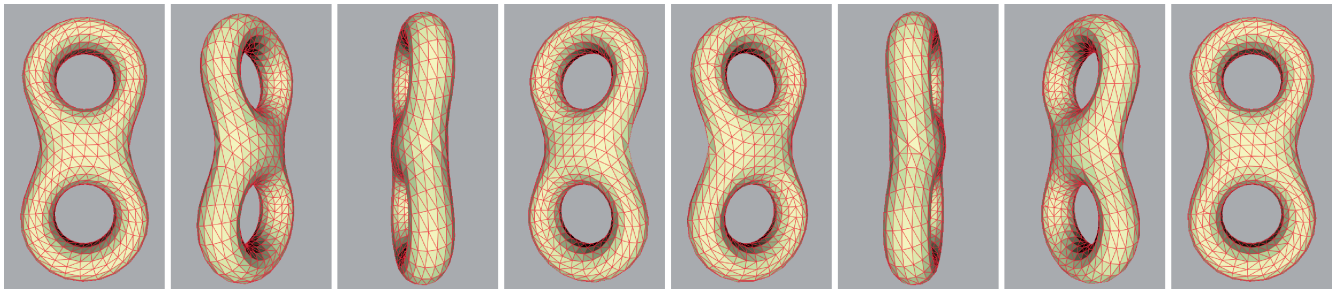


Fig. 3. Rotation in the DSC framework. Changes to the interface mesh are minimal; vertices have been added to the interface in order to relax the DSC mesh.

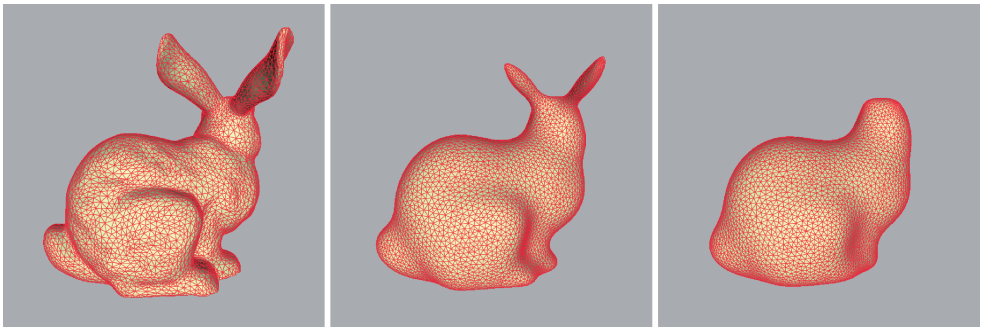


Fig. 4. Mean curvature flow quickly erases high-frequency details of the surface. Note an important property of the DSC, namely the interface mesh only changes in the regions where it is needed, in order to accomodate the deformation. One can notice that most of the triangulation remains essentially unchanged between the frames, except for the ears, where the triangulation changes a lot (since this part of the mesh is affected the most by the mean curvature flow). The bunny model is courtesy of the Stanford 3D scanning repository.

—If the vicinity of the face does not contain any point from the point cloud, we offset it in the normal direction, inwards;

—if the vicinity of the face contains points from the point cloud (and f is the closest face for these points), we compute their least squares minimizing plane.

The plane computed for a given face f is pushed to all vertices incident on f, and a new vertex position is computed as the intersection of all the planes pushed from incident faces. Whenever the plane containing f does not approximate the points assigned to it accurately enough, we subdivide it. This extremely simple approach turns out to give decent results: it handles the sharp features correctly, unlike Poisson reconstruction [Kazhdan et al. 2006] or early approaches based on Radial Basis Functions [Turk and O'brien 2002] (unless normal constraints are used [Shen et al. 2004]). Furthermore, mesh subdivision occurs only when needed (see Figure 9).

4.4 Fluid Simulation

One of the main applications of deformable models is to track the free surface in fluid simulations. The free surface of a fluid undergoes drastic deformation and frequent changes in topology, so robust topological adaptivity is crucial. Traditionally, the level set method has been used alongside regular grid-based Navier-Stokes equation solvers [Bridson 2008]. This approach has proven extremely successful and is widely used in multiple practical applications but it has its limitations. Many of them, such as volume loss or difficulty handling curved solid boundaries, have been addressed by numerous, often very elaborate patches. The lack of explicit interface representation makes it quite difficult to simulate surface phenomena and incorporating them in a fluid simulation framework requires fairly complex operations on the subgrid scale meshes used for tracking the fluid's surface [Wojtan et al. 2010; Thürey et al. 2010].

In Misztal et al. [2010] and Erleben et al. [2011] we present a completely new, finite element method-based approach to fluid simulation, utilizing the DSC method for tracking the free surface of the fluid and using the DSC mesh as a FEM grid with linear elements. Our solver reformulates the incompressible Euler equations (inviscid flow equations) as an optimization problem and allows to couple them with the surface energy term explicitly incorporated into the objective function. In addition, the method automatically handles collisions with arbitrary planar and curved solid boundaries.

A few examples are shown in Figures 10, 11, and 12. Figure 10 shows the surface mesh. Applying a single iteration of Loop subdivision to the interface mesh allows to create plausible looking animations (as shown in Figures 11 and 12).

4.5 Scalability and Performance

The complexity of all the mesh improvement operations used in the 3D DSC algorithm is approximately linear with respect to the number of tetrahedra in the mesh. Our experiments also suggest that the number of tetrahedra in the DSC mesh is at most a quasi-linear function of the number of the interface triangles. Additional tests performed during the simple geometric flow examples have shown that the DSC mesh is dominated by tetrahedra adjacent to the interface (around 70% of all tetrahedra in all DSC meshes in our experiments are adjacent to the interface). In most of our experiments, the ratio of the total number of tetrahedra to the number of interface triangles fluctuates between the values 5 and 6.

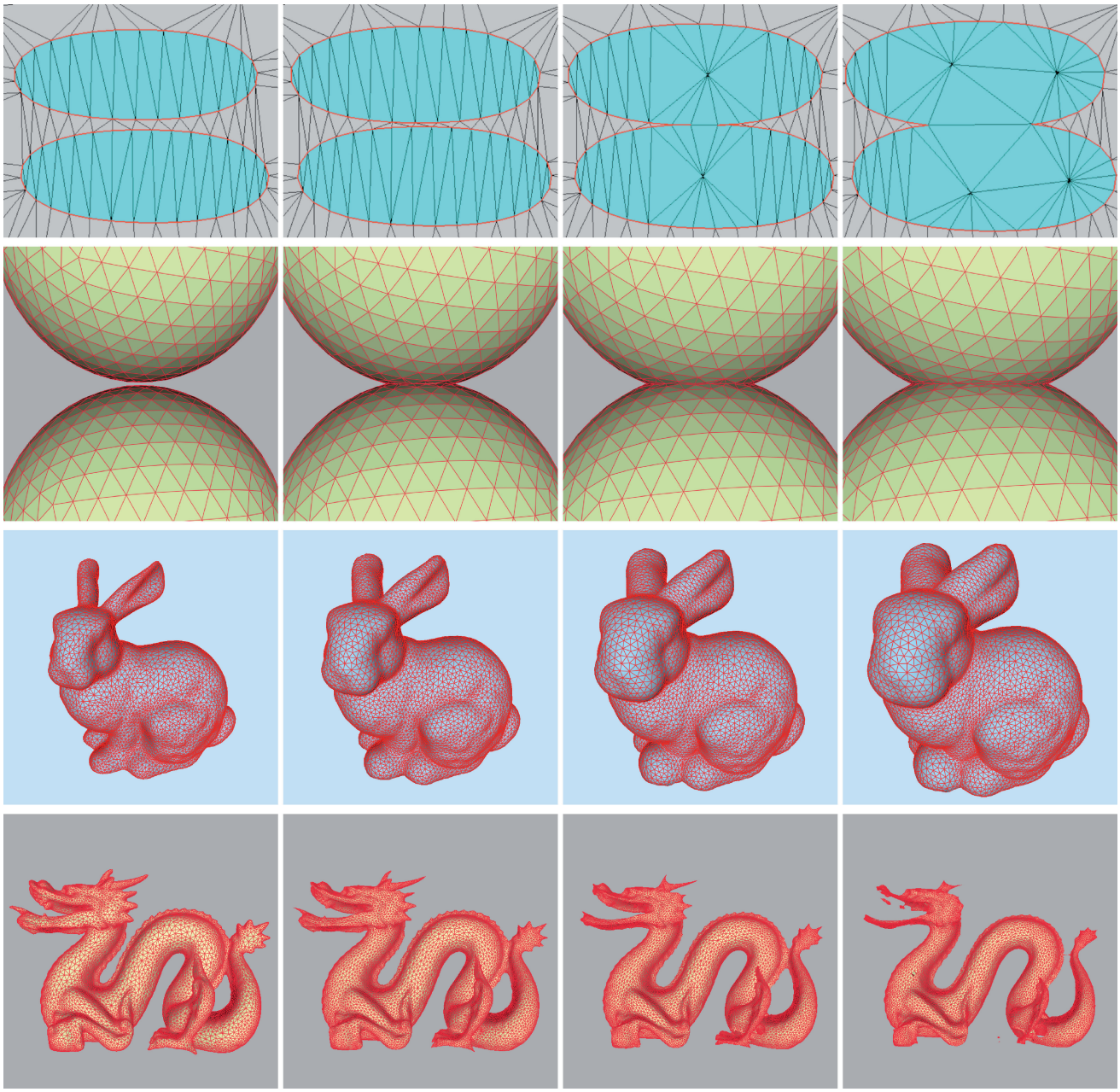


Fig. 5. Interface offsetting, from the top: 2D case; close-up on the area affected by collision of two expanding spheres; Stanford bunny expanding in the normal direction; Dragon model shrinking in the normal direction (the bunny and dragon models are courtesy of the Stanford 3D scanning repository http://graphics.stanford.edu/data/3Dscanrep/).

Although low-quality tetrahedra occasionally appear in the DSC mesh, dihedral angles from outside 6°–171° range constitute less than 0.1% of all dihedral angles in the tetrahedral mesh throughout all iterations (see Table II). This is sufficiently good for some applications, including those where the DSC mesh is not used for FEM computation, as well as FEM simulations for visualization purposes only, however, high accuracy FEM simulations might require a narrower range of dihedral angles.

Single iteration time might depend on the size of the displacement. If the displacements are large compared to the interface mesh edge size, several collisions might need to be resolved and bringing all interface vertices to their final positions might require more than just one or two executions of the main loop in Algorithm 1. Compare the results for the bunny example in Table II: in the offsetting example the velocity vector length does not vary much from vertex to vertex, and for sufficiently small time steps, the average iteration

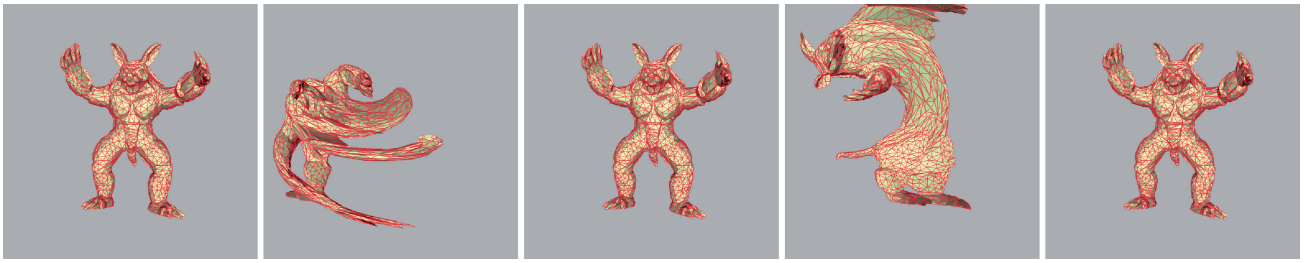


Fig. 6. The full cycle of the Enright deformation for decimated Armadillo model, at $t = 0$, 0.5, 1, 1.5, and 2 (the interface comes back to the original shape at $t = 1$ and $t = 2$). The armadillo model is courtesy of the Stanford 3D scanning repository.

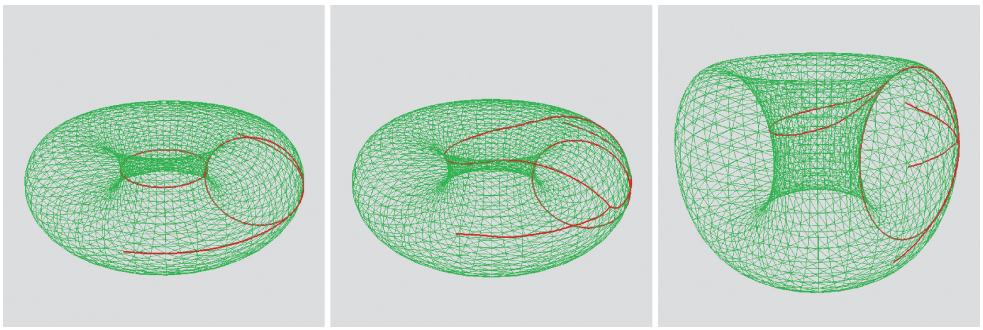


Fig. 7. 2D DSC-based cut locus construction algorithm results for tori: left and middle: cut loci of a standard torus of revolution with circular generator $\mathbf{r}_1(u, v) = ((2+\cos(v))\cos(u), (2+\cos(v))\sin(u), \sin(v))$, for points $(u, v) = (0, 0)$ and $(0, 0.1\pi)$); middle and right: cut locus of a standard torus of revolution with elliptic generator $\mathbf{r}_2(u, v) = ((2 + \cos(v))\cos(u), (2 + \cos(v))\sin(u), 2\sin(v))$, for a point $(u, v) = (0, 0.1\pi)$.

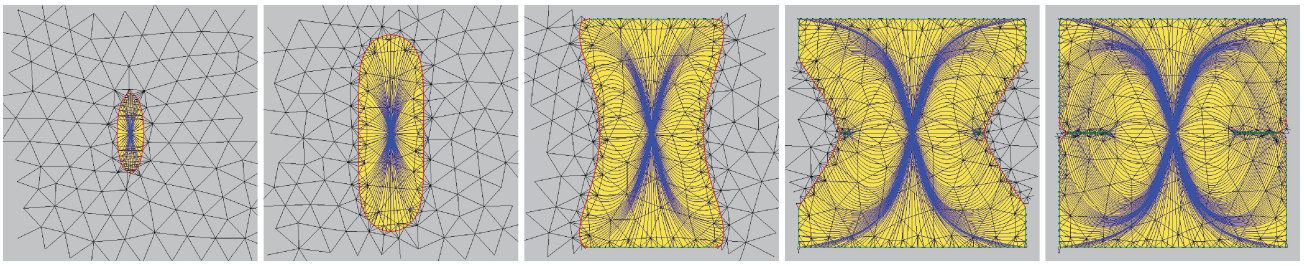


Fig. 8. A few steps of the cut locus construction algorithm for a standard torus of revolution. An equidistant front (highlighted in red) is propagating along geodesics (highlighted in blue) in the (u, v)-plane. Whenever the front collides with itself it stops (highlighted in green).

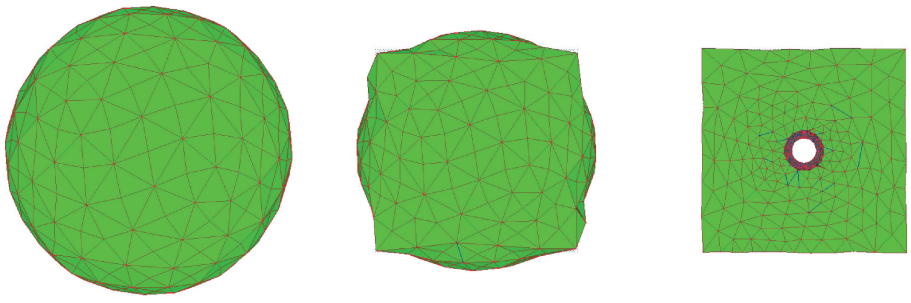


Fig. 9. Reconstruction of an artificial point cloud: a box with a cylindrical tunnel.

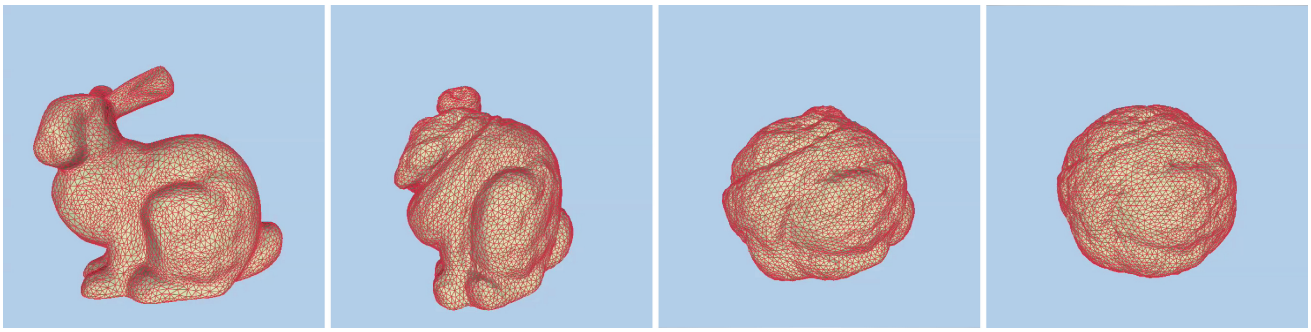


Fig. 10. An example of fluid simulation using 3D DSC: Stanford bunny deformed by surface tension forces alone (the bunny model is courtesy of the Stanford 3D scanning repository http://graphics.stanford.edu/data/3Dscanrep/).

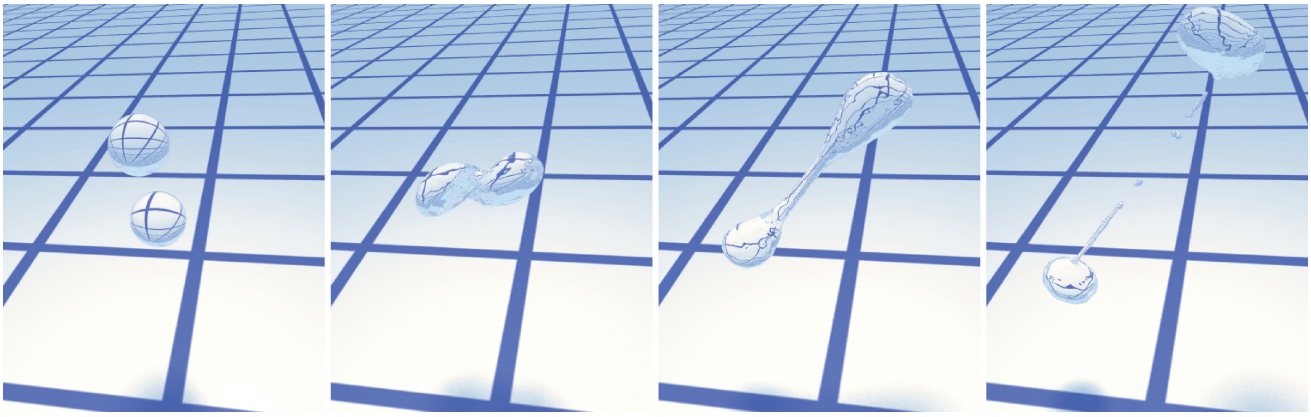


Fig. 11. An example of fluid simulation using 3D DSC: two droplets colliding obliquely (the interface mesh was subdivided once using Loop subdivision and ray traced as a postprocess).

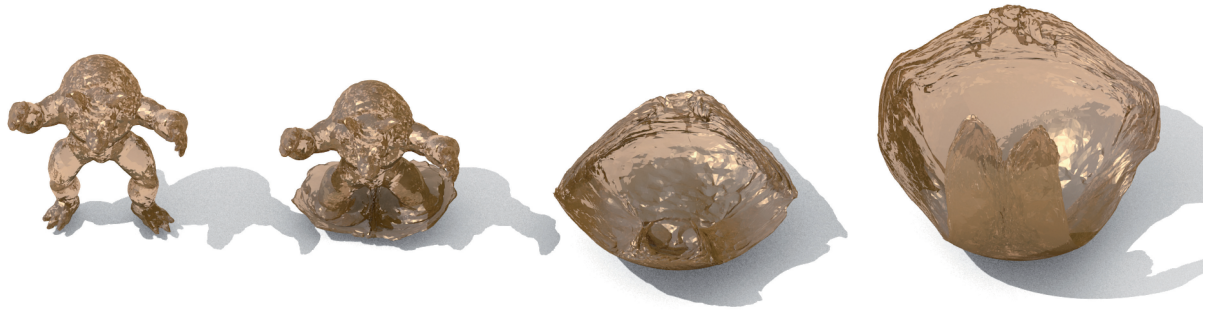


Fig. 12. An example of viscous fluid simulation using 3D DSC: Armadillo splashing inside a spherical bowl (the interface mesh was subdivided once using Loop subdivision and ray traced as a postprocess). The armadillo model is courtesy of the Stanford 3D scanning repository.

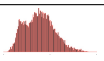
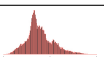
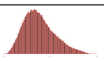
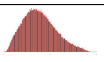
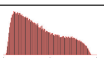
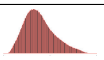
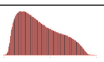
time is as low as 5 seconds, while in the mean curvature motion example the displacement values can be very large in the high curvature regions (where the triangles are also appropriatelly smaller) resulting in almost three times higher average iteration time.

On average, for similar flows, the computation time seems to scale linearly with respect to the number of tetrahedra in the mesh. All experiments were run on 64-bit Intel® Core® i7 CPU X980 @ 3.33 GHz, 24GB RAM).

5. CONCLUSIONS AND FUTURE WORK

Our results show that the deformable simplicial complex can be an interesting addition to an existing deformable models toolbox. It shares the main advantage of the level set method: robust topological adaptivity, while suffering from little to no numerical diffusion and naturally supports adaptive resolution. Likewise, the DSC method's intrinsic collision detection mechanism makes topology

Table II. Statistics for the Simple Geometric Flows

example	#surf. triangles	#total tets	#inside tets	min/max dih. angles	dih. angle distrib.	time per iteration
GENUS2	1536	10609	2963	7.38°–164.42°		
rotation	1552	6781	2864	4.83°–164.36°		0.32 s
TWO SPHERES	3240	17773	6509	7.01°–163.61°		
offsetting	2910	11166	6037	2.87°–171.69°		0.81 s
BUNNY	13932	88659	40763	3.86°–172.46°		
offsetting	13444	57257	32477	1.15°–177.93°		5.07 s
mean curv. flow	12060	61609	27776	5.12°–170.95°		13.94 s
DRAGON	87840	518043	249142	4.55°–172.7°		
offsetting	65528	355321	156149	0.52°–178.4°		36.22 s

For each experiment we present mesh data before and after the simulation. From the left to the right, the following statistics are shown: the number of the interface triangles; the total number of tetrahedra in the DSC mesh; the number of the tetrahedra marked *inside*; minimum and maximum dihedral angles among the inside tetrahedra; the histogram of dihedral angles distribution in the inside mesh; average iteration time.

control natural and simple, and the domain can have topology other than a disk, as demonstrated by the cut locus application. Moreover, the DSC mesh can be used for finite element computations.

While the speed of the DSC is comparable to other tetrahedral mesh-based methods [Wicke et al. 2010], it is slower than the level set method. However, there is an untapped potential for parallelizing computations in the DSC method which we intend to explore. Moreover, for some applications, we can improve speed by performing operations only locally where changes to the interface occur. Note also that in many cases FEM computations take significantly more time than interface advection.

In order to use the DSC in FEM simulations requiring more accuracy, we would first have to further improve the DSC mesh quality (in terms of both extreme dihedral angles and mesh refinement), possibly by introducing Klinger's and Shewchuk's vertex insertion algorithm [Klingner and Shewchuk 2007; Wicke et al. 2010] into the repertoire of tetrahedral mesh operations used in the DSC algorithm. Furthermore, the results could benefit from higher-order time step integration and from computing exact collision time and using it to determine the ordering of vertex displacements in a single DSC iteration.

The DSC method in its current form allows for nonmanifold configurations in the surface mesh. While this sort of generality can be desirable, some applications require the surface mesh to be 2-manifold. In order to enforce that, we could try to construct a tunnel-like structure around a nonmanifold vertex or edge, whenever it is introduced, however, we have yet to analyse this problem.

We would also like to continue working on the applications mentioned in Section 4.

ACKNOWLEDGMENTS

We are grateful to the following people for resources, discussions and suggestions: François Anton, Jeppe Revall Frisvad, Asger Nyman Christiansen, Tomé Santos Silva (Technical University of Denmark), Kenny Erleben (University of Copenhagen), Adam Bargteil (University of Utah), Robert Bridson and Carl Ollivier-Gooch (University of British Columbia) as well as anonymous reviewers.

REFERENCES

BÆRENTZEN, J. A. 2010. Introduction to GEL. http://www2.imm.dku.dk/projects/GEL/intro.pdf.

BARGTEIL, A. W., GOKTEKIN, T. G., O'BRIEN, J. F., AND STRAIN, J. A. 2006. A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph. 25,* 1.

BRIDSON, R. 2008. *Fluid Simulation*. A. K. Peters, Ltd., Natick, MA.

BROCHU, T. AND BRIDSON, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM J. Sci. Comput. 31,* 4, 2472–2493.

CHEW, L. 1997. Guaranteed-Quality delaunay meshing in 3d (short version). In *Proceedings of the 13th Annual Symposium on Computational Geometry*. ACM, New York. 391–393.

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys. 183,* 1, 83–116.

ERLEBEN, K., MISZTAL, M. K., AND BÆRENTZEN, J. A. 2011. Mathematical foundation of the optimization-based fluid animation method. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. 101–110.

FLORIANI, L. D., HUI, A., PANOZZO, D., AND CANINO, D. 2010. A dimension-independent data structure for simplicial complexes. In *Proceedings of the 19th International Meshing Roundtable*.

FREITAG, L. A., JONES, M., AND PLASSMANN, P. 1995. An efficient parallel algorithm for mesh smoothing. In *Proceedings of the 4th International Meshing Roundtable*. 103–112.

FREITAG, L. A. AND OLLIVIER-GOOCH, C. 1997. Tetrahedral mesh improvement using swapping and smoothing. *Int. J. Numer. Methods Engin. 40*, 3979–4002.

GLIMM, J., GROVE, J. W., LI, X. L., SHYUE, K.-M., ZENG, Y., AND ZHANG, Q. 1995. Three dimensional front tracking. *SIAM J. Sci. Comput. 19*, 703–727.

HANSEN, M. F., BÆRENTZEN, J. A., AND LARSEN, R. 2009. Generating quality tetrahedral meshes from binary volumes. In *Proceedings of the VISAPP Conference*.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *Proceedings of the ACM SIGGRAPH Conference*. 19–26.

JIAO, X. 2007. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys. 220,* 2, 612–625.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing (SGP '06)*. Eurographics Association, 61–70.

KLINGNER, B. M. AND SHEWCHUK, J. R. 2007. Agressive tetrahedral mesh improvement. In *Proceedings of the 16th International Meshing Roundtable*. 3–23.

LACHAUD, J. AND MONTANVERT, A. 1999. Deformable meshes with automated topology changes for coarse-to-fine three-dimensional surface extraction. *Med. Image Anal. 3,* 2, 187–207.

LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. *ACM Trans. Graph. 25,* 3.

MÄNTYLÄ, M. 1988. *An Introduction to Solid Modeling*. Computer Science Press.

MCINERNEY, T. AND TERZOPOULOS, D. 2000. T-snakes: Topology adaptive snakes. *Med. Image Anal. 4,* 2, 73–91.

MISZTAL, M. K., BÆRENTZEN, J. A., ANTON, F., AND ERLEBEN, K. 2009. Tetrahedral mesh improvement using multi-face retriangulation. In *Proceedings of the 18th International Meshing Roundtable*. 539–556.

MISZTAL, M. K., BÆRENTZEN, J. A., ANTON, F., AND MARKVORSEN, S. 2011. Cut locus construction using deformable simplicial complexes. In *Proceedings of the 8th International Symposium on Voronoi Diagrams in Science and Engineering*.

MISZTAL, M. K., BRIDSON, R., ERLEBEN, K., BÆRENTZEN, J. A., AND ANTON, F. 2010. Optimization-Based fluid simulation on unstructured meshes. In *Proceedings of the 7th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS10)*.

OSHER, S. J. AND FEDKIW, R. P. 2002. *Level Set Methods and Dynamic Implicit Surfaces* 1st Ed. Springer.

PARTHASARATHY, V. N., GRAICHEN, C. M., AND HATHAWAY, A. F. 1994. A comparison of tetrahedron quality measures. *Finite Elements Anal. Des. 15,* 3, 255–261.

PINKALL, U. AND POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Exper. Math. 2,* 1, 15–36.

PONS, J.-P. AND BOISSONNAT, J.-D. 2007a. Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–8.

PONS, J.-P. AND BOISSONNAT, J.-D. 2007b. A lagrangian approach to dynamic interfaces through kinetic triangulation of the ambient space. *Comput. Graph. Forum 26,* 2, 227–239.

SAKAI, T. 1996. *Riemannian Geometry*. Translations of Mathematical Monographs, vol. 149, American Mathematical Society.

SHEN, C., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2004. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH Conference*. ACM Press, 896–904.

SHEWCHUK, J. R. 1998. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the 14th Annual Symposium on Computational Geometry*. ACM, New York. 86–95.

SHEWCHUK, J. R. 2002. What is a good linear finite element? Interpolation, conditioning, anisotropy, and quality measures. http://www.cs.berkeley.edu/jrs/papers/elemj/pdf.

SI, H. 2004. Tetgen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, v1.3 user's manual. Tech. rep., WIAS.

THÜREY, N., WOJTAN, C., GROSS, M., AND TURK, G. 2010. A multiscale approach to mesh-based surface tension flows. In *ACM SIGGRAPH 2010 Papers*. ACM, New York, 1–10.

TURK, G. AND O'BRIEN, J. F. 2002. Modelling with implicit surfaces that interpolate. *ACM Trans. Graph. 21,* 4, 855–873.

WICKE, M., RITCHIE, D., KLINGNER, B. M., BURKE, S., SHEWCHUK, J. R., AND O'BRIEN, J. F. 2010. Dynamic local remeshing for elastoplastic simulation. In *Proceedings of the ACM SIGGRAPH'10 Conference*. 49: 111.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2009. Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 Papers*. ACM, 76.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2010. Physics-Inspired topology changes for thin fluid features. In *ACM SIGGRAPH 2010 Papers*. ACM, New York, 1–8.

ZAHARESCU, A., BOYER, E., AND HORAUD, R. 2007. Transformesh: A topology-adaptive mesh-based approach to surface evolution. In *Proceedings of the Computer Vision Conference*. 166–175.

ZAHARESCU, A., BOYER, E., AND HORAUD, R. P. 2011. Topology-adaptive mesh deformation for surface evolution, morphing, and multi-view reconstruction. *IEEE Trans. Pattern Anal. Mach. Intell. 33,* 4, 823–837.

Received October 2010; revised December 2011; accepted January 2012