HAL
open science

# Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multi-View Reconstruction

Andrei Zaharescu, Edmond Boyer, Radu Horaud

## To cite this version:

Andrei Zaharescu, Edmond Boyer, Radu Horaud. Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multi-View Reconstruction. [Research Report] RR-7136, INRIA. 2009, pp.34. inria-00438358v2

**HAL Id: inria-00438358**

**https://inria.hal.science/inria-00438358v2**

Submitted on 7 Dec 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multi-View Reconstruction*

Andrei Zaharescu — Edmond Boyer — Radu Horaud

**N° 7136**

December 2009

Domaine 4

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7136--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche GRENOBLE - RHÔNE-ALPES

# Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multi-View Reconstruction

Andrei Zaharescu , Edmond Boyer , Radu Horaud

Domaine : Perception, cognition, interaction
Équipe-Projet PERCEPTION

Rapport de recherche n° 7136 — December 2009 — 31 pages

**Abstract:** Triangulated meshes have become ubiquitous discrete-surface representations. In this paper we address the problem of how to maintain the manifold properties of a surface while it undergoes strong deformations that may cause topological changes. We introduce a new self-intersection removal algorithm, TransforMesh, and we propose a mesh evolution framework based on this algorithm. Numerous shape modelling applications use surface evolution in order to improve shape properties, such as appearance or accuracy. Both explicit and implicit representations can be considered for that purpose. However, explicit mesh representations, while allowing for accurate surface modelling, suffer from the inherent difficulty of reliably dealing with self-intersections and topological changes such as merges and splits. As a consequence, a majority of methods rely on implicit representations of surfaces, e.g. level-sets, that naturally overcome these issues. Nevertheless, these methods are based on volumetric discretizations, which introduce an unwanted precision-complexity trade-off. The method that we propose handles topological changes in a robust manner and removes self intersections, thus overcoming the traditional limitations of mesh-based approaches. To illustrate the effectiveness of TransforMesh, we describe several challenging applications: surface morphing and 3-D reconstruction.

**Key-words:** Mesh, surface, manifold mesh, triangulated mesh, mesh evolution, deformable objects, morphing, 3-D reconstruction.

This research was supported by the the EC Marie-Curie program through the VISION-TRAIN project.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

# Une Méthode de Déformation de Maillage par Topologie Adaptative pour l'Evolution de Surfaces, le Morphing et la Reconstruction Multi-Vues

**Résumé :** Les maillages triangulés sont devenus les représentations les plus courantes des surfaces. Dans cet article nous nous intéressons au problème du maintien des propriétés d'une surface, en termes d'une variété, lorsqu'elle subit des déformations pouvant causer des modifications topologiques. Nous introduisons un nouvel algorithme d'élimination d'auto-intersections, TransforMesh, et nous proposons une méthode d'évolution de maillage basée sur cet algorithme. De nombreuses applications utilisent l'évolution d'une surface afin d'améliorer les propriétés de la forme modélisée, comme l'apparence et la précision. Dans ce but, on peut aussi bien utiliser des représentations implicites qu'explicites. Cependant, les représentations explicites de maillages, tout en permettant une modélisation précise, souffrent de la difficulté inhérente à la gestion des changements topologiques. Par conséquent, une grande majorité de méthodes se basent sur des représentations implicites, comme les ensembles de niveau, qui surmontent ces difficultés de manière naturelle. Cependant, ce méthodes utilisent des descripteurs volumiques discrets, ce qui introduit un compromis précision/complexité. La méthode que nous proposons traite les changements topologiques de manière robuste et enlève les auto-intersections du maillage déformé, ce qui résout les limitations bien connues des approches basées sur des maillages. Afin d'illustrer notre méthode et l'algorithme TransforMesh, nous décrivons plusieurs applications, comme le morphing de surfaces et la reconstruction 3-D.

**Mots-clés :** Maillage, surface, variété, triangulation, évolution de maillage, objet déformable, morphing, reconstruction 3-D.

# Contents

**1 Introduction** **4**
1.1 Literature Review . . . . . 4
1.2 Contributions . . . . . 6

**2 Background** **7**

**3 The TransforMesh Algorithm** **9**
3.1 Self Intersections . . . . . 9
3.2 Valid Region Growing . . . . . 10
3.3 Triangle Stiching . . . . . 11

**4 Algorithm Analysis** **13**
4.1 Topological Changes . . . . . 13
4.2 Guarantees . . . . . 13
4.3 Numerical Stability . . . . . 14
4.4 Time Complexity . . . . . 15
4.5 Comparison with a Static Algorithm . . . . . 16
4.6 Implementation Details . . . . . 16

**5 Mesh evolution** **16**

**6 Surface morphing** **17**
6.1 Methodology . . . . . 18
6.2 Complexity Issues and Mesh Discretization . . . . . 19
6.3 Results . . . . . 20

**7 Multi-View 3-D Reconstruction** **21**
7.1 Methodology . . . . . 22
7.2 Results . . . . . 24

**8 Conclusion** **24**

# 1 Introduction

In the process of modeling shapes, several applications resort to surface evolution to improve shape properties. For instance, shape surfaces are evolved so that their appearances are improved, as when smoothing shapes, or so that they best explain given observations as in image based modeling. The interest arises in several fields related to shape modeling: computer vision, computer graphics, medical imaging and visualization among others. Surface evolution is usually formulated as an optimization process that seeks for a surface with a minimum energy with respect to the desired properties. To this aim, surfaces can be represented in different ways, from implicit to explicit representations, and deformed in an iterative way during optimization. Polygonal meshes, while being one of the most widely used representation when modeling shapes, are seldom used in such evolution schemes. The main reasons for that is the inherent difficulty to handle topological changes and self-intersections that can occur during evolution.

In this paper, we introduce an intuitive and efficient algorithm, named TransforMesh, that performs self-intersection removal of triangular meshes, allowing for topological changes, e.g. splits and merges. The method assumes as input a proper oriented mesh – a 2-D compact oriented manifold – which experienced any connectivity preserving deformation. It computes the *outside surface* of the deformed mesh. To illustrate the approach and its interests, we propose a generic surface evolution framework based on TransforMesh and we present two applications: mesh morphing and variational multi-view 3-D reconstruction.

## 1.1 Literature Review

As a result of the large interest for surface evolution in many application domains, numerous surface deformation schemes have been proposed over the last decades. They roughly fall into two main categories with respect to the representation which is considered for surfaces: Eulerian or Lagrangian.

**Eulerian methods** formulate the evolution problem as time variation over sampled spaces, most typically fixed grids. In such a formulation, the surface, also called the interface, is implicitly represented. One of the most successful methods in this category, the *level set method* [45, 47], represents the interface as the zero level of a higher dimensional function. A typical function used is the signed distance of the explicit surface, discretized over the volume. At each iteration the whole implicit function is moved. The explicit surface is recovered by finding the 0-level set of the implicit function. A number of methods have been proposed to extract surfaces from volumetric data [32, 36, 38, 44]. Such an embedding within an implicit function allows to automatically handle topology changes, e.g. merges and/or splits. In addition, such methods allow for an easy computation of geometric properties such as curvatures and benefit from *viscosity solutions* - robust numerical schemes to deal with the evolution. These advantages explain the popularity of level set methods in computer vision [46] as well as in other fields, such as computational fluid dynamics [55] and computer animations of fluids [18]. Nevertheless, implicit representations exhibit limitations resulting from the grid discretization. In particular, the precision/complexity trade-off inherent to the grid has a significant impact on the computational efficiency and the proposed narrow-band solutions [1] or octree

based implementations [39] only partially overcome this issue. In addition, as shown by Enright *et al.* [17], the level set method is strongly affected by mass loss, smearing of high curvature regions and by the inability to resolve very thin parts. Another limitation is that level set methods are not appropriate for tracking surface properties, such as color or texture, which can be desirable in many image-based approaches (i.e. motion tracking). Thus, while providing a solution for the intersection and topological issues within surfaces, implicit representations introduce a new set of issues for which careful solutions need to be crafted.

**Lagrangian methods** propose an approach where surfaces have explicit representations which are deformed over time. Such representations, meshes for instance, present numerous advantages, among which adaptive resolution and compact representation, as well as the ability to directly handle non-geometric properties over the surface, e.g. textures, without the necessity to reconstruct the interface. On the other hand, they raise two major issues when evolved over time, namely self-intersections and topology changes, which make them difficult to use in many practical scenarios. This is why non-intersections and fixed topology were explicitly enforced [26,48]. As a consequence, and in spite of their advantages, they have often been neglected in favor of implicit representations which provide practical solutions to such issues. Nevertheless, solutions have been proposed. McInerney and Terzopoulos [41] introduced topology adaptive deformable curves and meshes, called T-snakes and T-surfaces. However, in solving the intersection problem, the authors use a spatial grid, thus imposing a fixed spatial resolution. In addition, only offsetting motions, i.e. inflating or deflating, are allowed. Another heuristic method was proposed by Lauchaud et al. [37] for mesh deformations. Merges and splits are performed in near boundary cases: when two surface boundaries are closer than a threshold and facing each other, an artificial merge is introduced; a similar procedure is applied for a split, when the two surface boundaries are back to back. Self-intersections are *avoided* in practice by imposing a fixed edge size. A similar method was also proposed by Duan *et al.* [15]. Alternatively, Pons and Boissonat [49] proposed a mesh approach based on a restricted 3-D Delaunay triangulation. A deformed mesh is obtained by triangulating the moved vertices and assuming that the tetrahedra categorization, i.e. inside and outside, remains after the deformation. While being a robust and elegant solution, it nevertheless relies on the assumption that the input mesh is sufficiently dense such that the Delaunay triangulation will not considerably change its layout.

The methods proposed by Aftosmis *et al.* [2] and Jung *et al.* [33] are also related to our work. The algorithm in [2] recovers the outside surface obtained from self-intersecting meshes. The output mesh is obtained by identifying facets, or part of facets, which are on the exterior. The algorithm [33] uses the same idea, applied in the context of mesh offsetting. As explained below in detail, we generalize these approaches to the more general situations of any mesh deformation.

As a **hybrid method**, the recent work of Wojtan et al. [58] is representative, where the topological changes to the mesh are handled by first identifying merging or splitting events at a particular grid resolution, and then locally creating new pieces of the mesh in the affected cells using a standard isosurface creation method. The topologically simplified portions of the mesh are stitched to the rest of the mesh at the cell boundaries. While the authors present very

convincing results, they acknowledge some limitations, such as the restriction to a particular grid cell size, as well as some topological concerns related to matching exactly the extracted isosurface to the original mesh, among others.

In addition to the two above categories it is worth to mention also **Solid modeling methods** that provide practical tools to represent and manipulate surface primitives. Methods in this domain fall into two categories: Constructive Solid Geometry (CSG) [20,30] and Boundary Representation (B-Rep) [7,10]. CSG methods represent shapes as a combination of elementary object shapes based on Boolean operations. Alternatively, B-Rep methods adopt the more natural approach to represent the object boundary using vertices, edges and facets [3, 52]. Each representation has its advantages. While Boolean operations on CSG objects are straightforward, a lot of computational effort is required to render CSG objects [24, 51]. On the other hand, it is much more difficult to implement Boolean operations on boundary representations (multi-resolution surfaces) [8, 43], whereas interactive rendering is trivial. While these methods propose solution for computing Boolean operations of surfaces, to the best of our knowledge they do not deal with any extension needed to address self-intersecting meshes. Generally, the methods are more concerned with the rendering of the resulting geometry than with the generation of correct manifolds in the case of self-intersections.

## 1.2 Contributions

In this paper we propose a novel topology-adaptive self-intersection removal method for triangular meshes as well as an associated efficient algorithm, TransforMesh, with guaranteed convergence and numerical stability. We generalize previous work in this area [2, 33] to any topology changes resulting from mesh deformation, including *merges*, *splits*, *hole formations*, and *hole losses*, e.g. Figure 7. The main contribution is that given an input mesh with self-intersections, the algorithm provides a 2-D compact oriented manifold that represents the *outside skin* of the input mesh. Such an input mesh is typically obtained by applying arbitrary deformations to its vertices, as is often the case with such techniques as surface evolution, surface morphing, or multi-view 3D reconstruction.

The vast majority of the mesh-based surface deformation algorithms available today are based on *topology-preserving* methods. Alternatively, we propose a *topology-adaptive* mesh evolution method that is entirely based on TransforMesh. Such topology-adaptive scheme is more general and hence better adapts to challenging applications such as 3D reconstruction using multiple images and non-rigid surface tracking.

Recent image-based reconstruction methods [53] make use of surface evolution to obtain accurate 3D models. Our approach contributes in this field by providing an efficient unconstrained mesh-based solution that allows for facets of all sizes as well as for topology changes, with the goal of increasing precision without sacrificing complexity. The robustness and flexibility of the proposed framework is also validated in the context of mesh morphing, showing several topologically challenging examples.

The remainder of this article is organized as follows. Section 2 provides some background concepts on which our method resides. Section 3 describes in detail the TransforMesh algorithm. Various aspects of the algorithm, such that the topological changes that it can handle, convergence, numerical stability

and time complexity are detailed in section 4. Section 5 describes the mesh evolution algorithm based on TransforMesh. Mesh morphing is described in section 6. Section 7 describes how our methodology is plugged into the task of 3D reconstruction. Finally, we conclude in section 8.

TransforMesh is available as an open-source software package (OSS) under a general public licence (GPL) at http://mvviewer.gforge.inria.fr/.

## 2 Background

Before we introduce the TransforMesh algorithm we precise the context within which it applies. We assume an initial mesh representing the surface of a real object to be deformed into a self-intersecting input mesh from which the TransforMesh algorithm extracts an output mesh. More precisely, we assume that the initial mesh represents a *compact oriented 2-D manifold* with possibly several components and we expect the output mesh to do the same. Consequently both initial and output meshes should satisfy the following properties: every edge belongs to exactly two flat faces; every vertex is surrounded by a single cycle of edges and faces; faces are oriented and do not intersect except at edges and vertices. The deformation that the initial mesh underwent can then be any transformation that preserves the mesh graph structure, i.e. its connectivity. Hence any vertex displacement field that preserves edges is acceptable. Note that this excludes displacements that fuse neighboring vertices.

The TransforMesh algorithm relies on the identification of outside or exterior faces on the deformed input mesh. An exterior face on an oriented mesh is a boundary face that delimits interior and exterior regions and that is oriented towards an exterior region, i.e. its normal points outward. To further identify regions delimited by the mesh as interior or exterior we need a rule. Traditionally, interior and exterior regions are defined with an *even-odd parity* rule. Such rule simply consists of counting the number of intersections of a ray, emanating from a point, with the delimiting primitive. If this number is odd, the point belongs to an interior region, if not, the point is on the exterior. While efficient, this rule originally applies to simple primitives, e.g. simple closed curves in 2D and closed surfaces in 3D, and does not correctly handle more complex primitives in particular self-intersecting primitives. In that case, the *winding* rule allows regions to be better differentiated by using the primitive's orientation. This appears to be crucial when operating topological changes, such as merge and split, over regions.

The winding number of a point $p$ with respect to an oriented primitive is the number of times the primitive *winds* or cycles around $p$. Cycles are counted positively or negatively depending on their orientations around the point. $p$ is then outside when its winding number is 0, inside otherwise. Figure 1 depicts this principle in 2D.

To compute this number, two strategies can be followed. A first strategy consists in computing the total signed angle, solid angle in 3D, made by a ray from the point under consideration to another point traveling along the primitive [12]. The sum will be equal to 0 for a point on the exterior and a multiple of $2\pi$, $4\pi$ in 3D, for a point on the interior. Another strategy considers a ray from a point and its intersections with the primitive [19]. Each intersection is assigned a value $+1$ or $-1$ according to the sign of the dot product of the ray

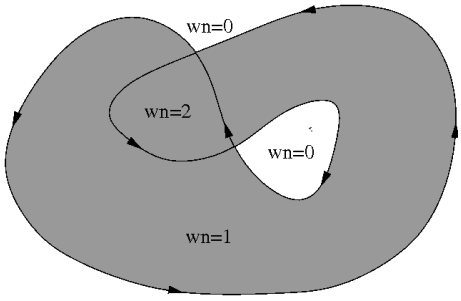



Figure 1: Interior and exterior regions delimited by an oriented primitive. A point is on the exterior when it belongs to a region with a winding number $wn$ equal to 0, on the interior otherwise.

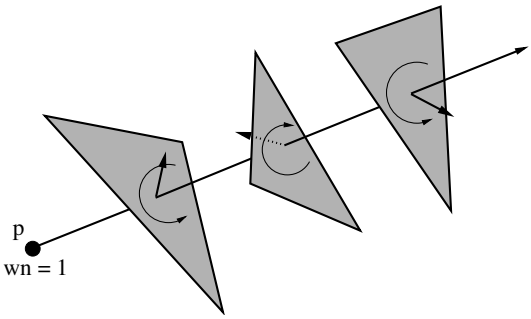



Figure 2: The winding number at $p$ can be obtained by summing the dot product signs with face normals along any ray from $p$.

direction with the normal to the primitive at the intersection. If this sign is negative the value is $-1$ and $+1$ otherwise, see Figure 2. The sum of these values will be 0 only for a point on the exterior. We use this strategy to verify whether a face is on the exterior. We take a ray from the center of the face towards its normal direction and we sum the values $-1$ and $+1$ obtained at the intersections with other faces along the ray. The face is on the exterior when this sum is 0.

We call then a *valid face* a face fully on the exterior without intersections with other faces and a *partially valid face* a face divided by intersections into sub-parts, some of which being on the exterior. Notice that valid faces can be found inside the mesh, as independent connected components may appear inside the mesh as a result of self-intersections. Although these components are valid parts of the resulting mesh, they are usually not considered in evolution processes that do rely on criteria applying on exterior surfaces only, distance or photo-consistency for example.

# 3 The TransforMesh Algorithm

The TransforMesh algorithm removes self-intersection and adapts to topological changes in triangular meshes using an intuitive geometrically-driven solution. In essence, the approach preserves the surface consistency, i.e. 2-D manifoldness, by detecting self-intersections and considering the subset of the original surface that is still *outside*. In order to identify the corresponding faces in the mesh, the method consists in first finding an initial seed face that is fully on the exterior, using the winding rule presented in the previous section, and then propagating the exterior label over the mesh faces by means of region growing. Figure 3 illustrates the algorithm, the different steps are detailed in the following.

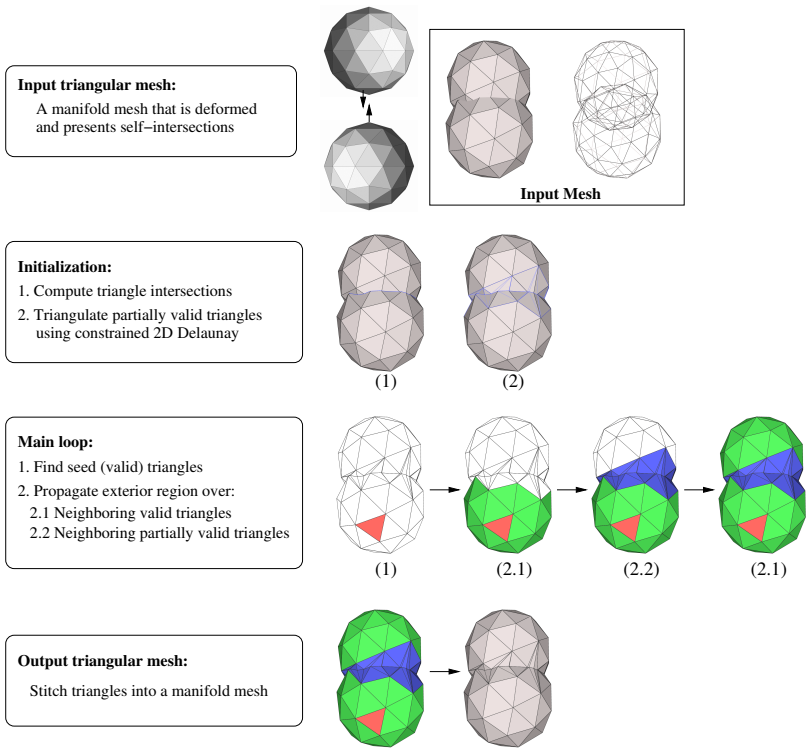



Figure 3: Overview of TransforMesh.

## 3.1 Self Intersections

The first step of the algorithm consists of identifying self-intersections, i.e. edges along which triangles of the mesh intersect.

This information will later on be needed in the computations, since it delimits the outside regions. In the general situation, one would have to perform $O(n^2)$ checks, with $n$ the number of triangles, to verify all triangle intersections, which can become expensive when the number of facets is large. In order to decrease the computational time, we use a bounding box test to determine which

bounding boxes (of triangles) intersect, and only for those perform a triangle intersection test. We use the fast box intersection method implemented in [34] and described in [61]. The complexity of the method is then $O(n\ log^3(n))$.

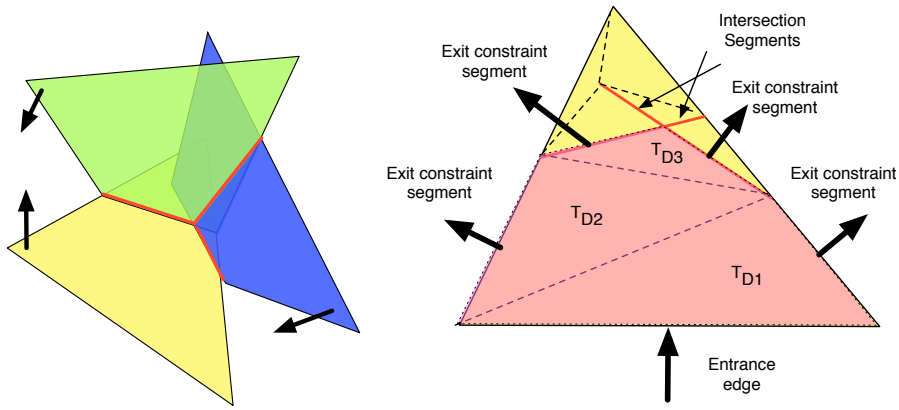
## 3.2 Valid Region Growing

The second step of the algorithm consists of identifying exterior triangles in the mesh. A valid region growing approach is used to propagate validity labels on triangles that composed the *outside* of the mesh. Alternatively, it can be viewed as a "painting" procedure, as it was originally described in [2]. Following this idea, we present here the sub-steps of the region-growing procedure. First, in the *Seed-triangle finding* step, valid triangles are sought as starting triangles without intersections that reside on the exterior. In the next *Valid triangle expansion* step this information is propagated by expanding on neighboring valid triangles until triangles with intersections are reached. The *Partially valid triangle traversal* step details then how to traverse the valid sub-parts of intersection triangles as well as how to cross from one intersecting triangle to the other. The local sub-parts are triangulated using a constrained 2-D Delaunay triangulation. The underlying idea that guides this step is to propagate the normal information from the seed triangles using the local geometry.

**Seed-triangle finding** A seed-triangle is defined as a non-visited valid triangle, found using the winding rule previously introduced. In other words, a seed-triangle is a triangle that is guaranteed to be on the exterior. This triangle is crucial, since it constitutes the starting point for the valid region growing. If found, the triangle will be marked as valid; otherwise, we assume that all outside triangles are identified and the algorithm jumps to the next stage (section 3.3). We have adopted the efficient AABB tree implementation described in [4] for the ray-to-triangles intersection test.

**Valid triangle expansion** Region growing over valid triangles is simply performed by checking neighbors of a valid triangle and stopping on the intersections: if the neighboring triangle is non-visited and has no intersections, then it is marked as valid; if the neighboring triangle is non-visited and has intersections, then it is marked as partially valid together with the entrance segment and direction, corresponding in this case to an oriented half-edge.

**Partially-Valid triangle traversal** In this step proper processing of regions containing intersections is ensured, with local geometry being generated. Let $t$ be a partially valid triangle as marked during the valid triangle expansion step. We have previously calculated all the intersection segments between this triangle and all the other triangles. Let $S_t = \{s_{ti}\}$ represent all the intersection segments between triangle $t$ and the other triangles. In addition, let $H_t = \{h_{tj}|\text{for } j = 1..3\}$ represent the triangle half-edges. A constrained 2-D triangulation performed in the triangle plane, using [27], ensures that all segments in both $S_t$ and $H_t$ appear in the new triangular mesh structure and that propagation can be achieved in a consistent way. A fill-like traversal is performed from the entrance half-edge to adjacent triangles, stopping on constraint edges, as depicted in Figure 4.




(a) Triangle intersections (b) Partially valid triangle traversal

Figure 4: Partially valid triangle traversal. (a)The intersections with all other triangles are computed for each intersecting triangle. (b) close-up of the bottom triangle in (a). The local geometry is re-defined using a constrained 2-D Delaunay triangulation that ensures the presence of the original triangle edges and the intersection segments. The traversal starts at the entrance edge and stops on constraint edges thus marking $T_{D1}$, $T_{D2}$ and $T_{D3}$ as valid.

Choosing the correct side of continuation of the "fill" like region growing when crossing from a partially valid triangle to another is a crucial aspect in ensuring a natural handling of topological changes. The correct orientation is chosen such that, if the original normals are maintained, the two newly formed sub-triangles would preserve the water-tightness constraint of the manifold. This condition can also be casted as follows: the normals of the two sub-triangles should be opposing each other when the two sub-triangles are "folded" on the common edge. A visual representation of the two cases is shown in Figure 5. The triangles on the other side of the exit constraint edges will be marked as valid appropriately, based on whether they contain any intersections or not.

Note that it is possible to visit a partially valid triangle multiple times, depending on whether there are multiple isolated (non-connected) exterior components. However, each sub-triangle formed by the local re-triangulation is only visited once. The simplest example to image is a cross, formed out of two intersecting parallelepipeds. There will be intersecting triangles appearing on both sides.

## 3.3 Triangle Stiching

The region growing algorithm described previously will iterate until there are no more unmarked triangles to visit. At this stage, what remains to be done is to stitch together the 3-D triangle soup ($\mathcal{G}$ queue) in order to obtain a valid mesh which is manifold. We adopt a method similar in spirit to [25,54]. In most cases this is a straight forward operation, which consists of identifying the common vertices and edges between facets, followed by stitching. However, there are three special cases, in which performing a simple stitching will violate the mesh

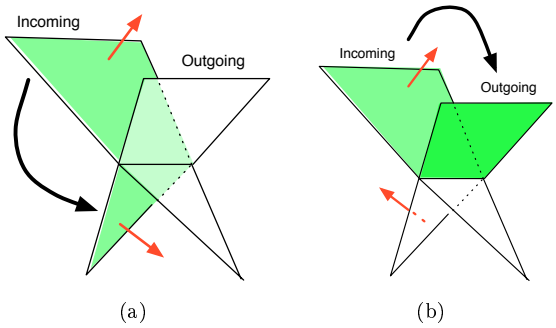



Figure 5: The 2 partially valid triangle crossing cases.

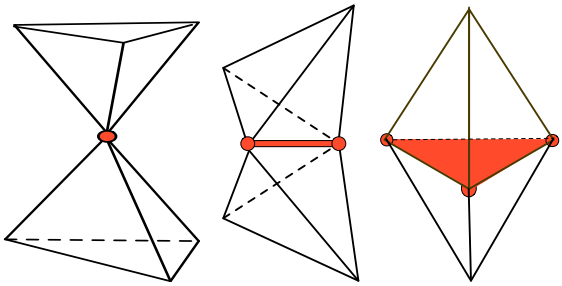
constraints and produce locally non-manifold structures. The special cases, shown in Figure 6, arise from performing stitching in places where the original structure should have been maintained. We adopt the naming convention from [25], calling them the singular vertex case, the singular edge case and the singular face case. All cases are easily identified by performing local operations.

**Singular vertex case** (Figure 6(a)). A vertex is shared by two or more different regions. In this case, the manifold property stating that for each manifold point, there is a single neighborhood, does not hold. The algorithm to detect these cases proceeds simply by checking that all facets incident to a vertex are within one neighborhood. The steps are: starting from a facet of $v$, mark it visited and do the same with its non-visited neighbors that are also incident to $v$ (neighboring triangles are chosen based on the available mesh connectivity); the process is repeated until all the neighboring facets are processed; if by doing so we exhausted all the neighboring facets, vertex $v$ is non singular, otherwise it is singular, so a copy of it is created and added to all the remaining non-visited facets. The process is repeated until all the incident facets are visited.

**Singular edge case** (Figure 6(b)). An edge is shared by two or more different regions, hence the manifold property does not hold. Such cases are detected and repaired by the singular vertex detection step, which will correctly identify and duplicate the two vertices that form the singular edge.

**Singular triangle case** (Figure 6(c)). A triangle is shared by two or more different regions, hence the manifold property does not hold. Such cases are detected and repaired by the singular vertex detection step, which will correctly identify and duplicate the three vertices that form the singular triangle.

Given that the original input mesh does not contain any of the above singular simplex scenarios, they rarely occur in practice. Note however that there are situations where creases are formed on the mesh, usually when inverting mesh regions, that can degenerate into singular cases.



(a) Singular vertex (b) Singular edge (c) Singular face

Figure 6: Special cases encountered while stitching a triangle soup.

# 4 Algorithm Analysis

Having introduced the algorithm in the previous section, we discuss in this section some of its most important aspects, including the handling of topological changes, the guarantee to obtain a valid mesh given a valid input mesh, the numerical stability and the time complexity.

## 4.1 Topological Changes

A nice feature of the algorithm is to correctly handle topological changes that result from the modification of the local geometry, i.e. faces that appear and disappear. We consider compact surfaces and in the general case, topological changes that can occur are: merge, split, hole formation and hole loss. They are depicted in Figure 7. Note that in 3D hole cases correspond to situations where a connected component is inside another connected component and that topological changes where handles appear or disappear are covered by the merge and split cases (see Figure 11 for examples). The partially valid triangle crossing technique described earlier in Section 3.2 and detailed in Figure 5 ensures a *natural* handling of these topological changes that plagued most of the mesh approaches until now. The *merge* case scenario, shown in Figure 7(a), coincides in spirit with the *union* Boolean set operation $\cup^*$. Less intuitive is the *split* operation, which will typically occur during a mesh evolution process, when certain parts will thin out up to the moment when some triangles from opposite sides will cross each other. Such a case is depicted in Figure 7(b), in a mesh morphing scenario, where the initial surface has 1 connected component and the destination 2 connected components. The 2 other examples, hole formation and loss, are less frequent. While handled by the algorithm, we do not account for them in practice since, as mentioned earlier, inside valid faces are usually not considered in surface evolution processes.

## 4.2 Guarantees

Given that the input mesh is a 2-D compact oriented manifold that has been deformed by a motion field and assuming exact computations (see section 4.3), TransforMesh will recover 2-D compact oriented manifold components. The

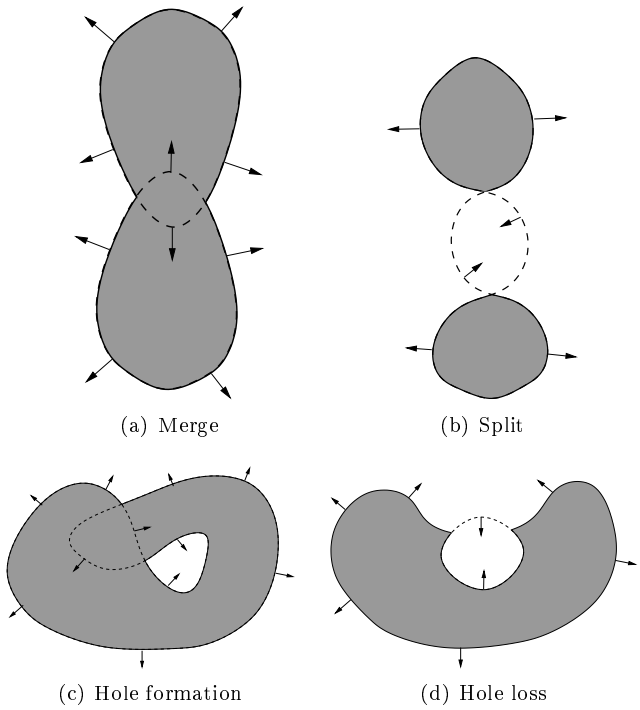



Figure 7: Topological changes (2-D simplified view).

number of components depends on the number of seed triangles detected. The algorithm will always *finish*, because it does not revisit already traversed subparts. In addition, it is guaranteed to always find the *exterior surface*, since it starts from a valid seed triangle, thus on the exterior, and it always rests that way, by propagating the normal information. The computed output is *manifold* by construction, since it traverses a valid input manifold and accounts for the manifold violations with the degenerate cases. It is *compact*, since the original input surface has no border and the algorithm does not build any, i.e. there is always a way outside a triangle intersection.

In addition, the 2-D manifold correctness is guaranteed by identifying and correcting all the possible 2-D manifold neighborhood violations when performing triangle stitching (singular vertex, singular edge and singular facet).

The algorithm preserves the geometry of the input mesh, with the exception of the self-intersection areas, where local triangulations redefine the geometry.

## 4.3 Numerical Stability

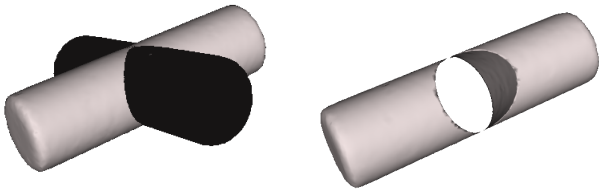
The numerical stability is critical, in order to be able to guarantee that the output is valid. It is ensured by using exact arithmetic predicates when computing intersections, as well as disambiguating the boundary cases. The special boundary cases between two intersecting triangles are:

1. the intersection reduces to a point;

2. the intersection is a segment that lies on one of the original triangle edges;

3. the intersection is a 2-D polygon, i.e. the two triangles are co-planar.

We disambiguate the above boundary cases using the *simulation of simplicity* technique of virtual perturbations [16]. It involves inducing a small vertex perturbation locally, which will force the triangle-triangle intersection into one of the classical cases: either the two triangles do not intersect or their intersection is a segment that does not lie on one of the original triangle edges. In practice, when such boundary situations occur, mesh offsetting is performed for that purpose. The choice of using the *simulation of simplicity* technique to handle boundary cases is motivated by the targeted application, mesh evolution, where such boundary situations rarely occur and where the explicit handling of all special cases would penalize the algorithm. We note, however, that there are applications, e.g. CAD applications, where such situations can occur, as shown in Figure 8. In that case, one could advantageously consider the approach proposed by Mäntylä [40], which treats all the possible scenarios explicitly, to disambiguate boundary cases. Other remarks are in order:



(a) Input (b) Output

Figure 8: Two inverted cylinders example. Examples of a boundary case testing numerical stability and the solutions provided by the algorithm.

- The first boundary case -when the intersection is a point- can alternatively be dealt with by considering the *regularized* intersection operation $\cap^*$, thus assuming that there is no intersection between the triangles.
- Others numerical stability issues related to fixed numerical precision, e.g. newly computed intersection segments might not be strictly co-planar with the original triangles, are handled by using exact geometrical predicates. See 4.6 for implementation details.
- In practice, triangles having near-zero area are also eliminated, should they occur, by using two mesh operations: edge collapse and edge flip.

## 4.4 Time Complexity

The overall time complexity of the algorithm depends on the number $n$ and relative sizes of facets and it is of $O(n \log^3(n))$ expected time (the average case). This complexity is dominated by the number of operations required to determine intersections. Each triangle requires $O(\log^3 n)$ tests, thanks to the fast box intersection method used, described in [61] and implemented in [34]. The complexity of the method is $O(n\ log^d(n) + k)$ for the running time and $O(n)$ for the

space occupied, $d$ the dimension (3 in the current case), and $k$ the output complexity, i.e., the number of pairwise intersections of the triangles. In practice, more than 80% of the running time is spent computing the self-intersections. Typically, the running time for performing the self-intersections test is under 1 second for a mesh with $50,000$ facets on a 2.6 GHz Intel Core2Duo, with no multi-threading, and where exact arithmetic is used for triangle intersections and where the self-intersections are in the range of 100.

### 4.5 Comparison with a Static Algorithm

Alternatively, one could use the valid triangle test, described in Section 2, in order to devise another static algorithm, which will test all the existing triangles and sub-triangles obtained from local Delaunay triangulations. The method will only choose the triangles that reside on the exterior, after which it will proceed to the final triangle stitching step.

However, this static algorithm would take considerably longer time, since it requires the same initial time $O(n \log^3 n)$ to compute all the triangle intersections and local Delaunay triangulations, followed by the additional time required for the valid triangle test, which is not negligible. Nevertheless, it should be noticed that this static algorithm is much simpler to implement than the proposed one and could be of interest in some applications, e.g. for validation purposes.

### 4.6 Implementation Details

In our implementation we have made use of CGAL (Computational Geometry Algorithms) C++ library [9], which provides *guaranteed* implementations for various algorithms. We have used the following CGAL modules: N-dimensional fast box intersections, 2-D constrained Delaunay triangulation, AABB trees, triangular meshes and support for exact arithmetic kernels.

## 5 Mesh evolution

A number of methods exist in the literature that deal with deformable surfaces, such as Kenneth Brakke's Evolver [1], Wojtan and Turk's visco-elastic simulator [59] or the work of Celniker and Gossard on deformable surfaces [13]. Nevertheless, the above mentioned methods are all mesh-based topology preserving. This might be or not a desired feature of the algorithm, depending on the target application. It is our goal, in the current section, to introduce an intuitive *generic mesh evolution paradigm* that is topology adaptive, based on TransforMesh. The main steps of the algorithm are presented in Figure 9. Within each evolution iteration, there are four steps. Firstly, a velocity vector field $\vec{\mathcal{F}}$ is computed for each vertex of the mesh $\mathcal{M}$. This step is application specific. Secondly, the mesh is deformed using the computed velocity vector field $\vec{\mathcal{F}}$ and a small time step $t$, thresholded by a maximum movement $\alpha \cdot e_{avg}(v)$, where $\alpha$ is a user-set threshold (typically between 0.1-0.3) and $e_{avg}(v)$ represents the local average edge length for a vertex $v$. Thirdly, TransforMesh is invoked in order to clean the potential self-intersections and topological problems introduced by the second step. The fourth step involves mesh optimization, with the goal of

[1] http://www.susqu.edu/facstaff/b/brakke/evolver/evolver.html

ensuring good mesh properties. Ideally, a mesh should consist of triangles as close to equilateral as possible, which allows for better computations of local mesh properties, e.g. curvatures and normals. To this purpose, a number of sub-steps are being performed: adaptive remeshing, vertex valence optimization and Laplacian smoothing. The adaptive remeshing step ensures that all edges are within a safety zone interval $(e_1, e_2)$, which is user-defined. This prevents edges from reaching close to zero sizes. In practice, this is obtained through edge collapse or edge swap operations. The vertex valence optimization step performs edge swaps in an attempt to ensure an overall vertex valence of 6 [35]. Vertex valence is defined as the number of edges shared by a vertex. The ideal vertex valence of 6 is desirable because, assuming that the manifold is generally locally planar, it is equivalent to obtaining 60° for each of the sharing triangle angles, thus optimizing for equilateral triangles. Alternatively, the vertex valence can also be improved by performing edge swaps only if it increases the minimum angle of either triangle adjacent to the edge. The Laplacian smoothing is attained by computing the discrete mesh Laplacian [14, 42], i.e. the discrete Laplace-Beltrami operator, $\Delta v$ for each vertex $v$ of the mesh. Furthermore, the mesh is smoothed using $v \rightarrow v - \beta \Delta v$. These four main steps are repeated until the mesh has reached the desired final state, also application specific.

We will present below two examples, one for mesh morphing in section 6, demonstrating the ability of the algorithm to handle complex surface evolutions, and the other one for multi-view 3-D reconstruction in section 7. In both cases, the application specific information is detailed in order to compute the vector fields $\vec{\mathcal{F}}_{morphing}$ and $\vec{\mathcal{F}}_{reconstruction}$, which plug directly within the generic mesh evolution framework presented in Figure 9.

**Choosing the correct time step.** The currently presented mesh evolution approach does not make any assumptions about choosing the right time-step. This parameter is entirely application specific. The TransforMesh algorithm does not have any information about the temporal component. It is therefore entirely up to the user to choose a meaningful time-step $t$ which will capture all the temporal dynamics. The only measure proposed in the generic evolution algorithm is to threshold the maximum vertex movement to $\alpha \cdot e_{avg}(v)$, in order to prevent both large jumps and to reduce the number of intersections.

**Remeshing.** The remeshing step is important and should theoretically occur at each timestep, due to the fact that some regions can become under-sampled in areas where the speed vector field is divergent or over-sampled in areas where the speed vector field is convergent. More importantly, intersections can generate poorly shaped triangles, which would probably have an impact on the local numerical process applied to the mesh that produces the vector field.

# 6 Surface morphing

A straightforward mesh evolution application of our algorithm is surface morphing, that is starting from a source surface $S_A$ and evolving it towards a destination surface $S_B$. This will allow us to test thoroughly various cases of topology changes. Surface morphing has been widely described in the literature. We will adopt the method proposed by Breen and Whitaker [11]. We will summarize the reasoning that leads the surface evolution equation.

**Generic Mesh Evolution using TransforMesh**

While Not Finished

1. **Compute Velocity Vector Field** $\vec{\mathcal{F}}$ of velocities for each vertex of the mesh $\mathcal{M}$, using application specific information.
2. **Evolve Mesh** $\mathcal{M}$ using the vector field $\vec{\mathcal{F}}$ and a small time-step $t$, thresholded by a maximum $\alpha \cdot e_{avg}(v)$, where $\alpha$ is a user-set threshold (typically set between 0.1-0.3) and $e_{avg}(v)$ is the local average edge length for a vertex $v$.
3. **Invoke TransforMesh** on $\mathcal{M}$ in order to clean self-intersections and topological problems
4. **Mesh Optimization**
   a) *Adaptive Remeshing*: ensures that all edges are within a safety zone interval, i.e. $\forall e \in \mathcal{M}, e \in [e_1, e_2]$, by performing edge swaps or edge collapses.
   b) *Vertex Valence Optimization*: perform edge swaps such that each vertex will be shared by 6 triangles [35].
   c) Perform *Mesh Laplacian Smoothing*: each vertex $v$ is updated by $v \rightarrow v - \beta\Delta v$, where $\Delta v$ represents the discrete mesh Laplacian [14, 42].

Figure 9: Generic mesh evolution algorithm using TransforMesh.

## 6.1 Methodology

A metric that quantifies how much two surfaces overlap is defined (source surface $S_A$ and destination surface $S_B$). A natural choice of such a metric is the signed distance function $\gamma_B$ of the destination mesh $S_B$', defined as in the level set literature as being negative inside the shape $S_B$, zero on the surface, and positive on the exterior. By considering the volume integral $\mathcal{M}_{S_B}(S_A)$ of any surface $S_A$ with respect to $\gamma_B$ (thus $S_B$), one can see that it will achieve the maximum when the two surfaces overlap. By taking the first variation of the metric $\mathcal{M}_{S_B}(S_A)$ with respect to the surface $S_A$ and a small displacement field and differentiating with respect to the vector field, one obtains the following evolution equation using a hill climbing strategy for each vertex $x$ along its normal $\mathbf{N}(x)$:

$$\vec{\mathcal{F}}_{morphing} = \frac{\partial S}{\partial t} = -\gamma_B(x)\mathbf{N}(x) \tag{1}$$

The evolution strategy described above will converge to a local minimum. Given the surface of departure $S_A$ and the destination surface $S_B$, $S_A$ will correctly find all the connected components of $S_B$ that are included in the original surface $S_A$. If $S_A$ represents a surface outside the destination surface $S_B$, $S_A$ will converge to an empty surface. We keep this result in mind when choosing the initial surface $S_A$.

## 6.2 Complexity Issues and Mesh Discretization

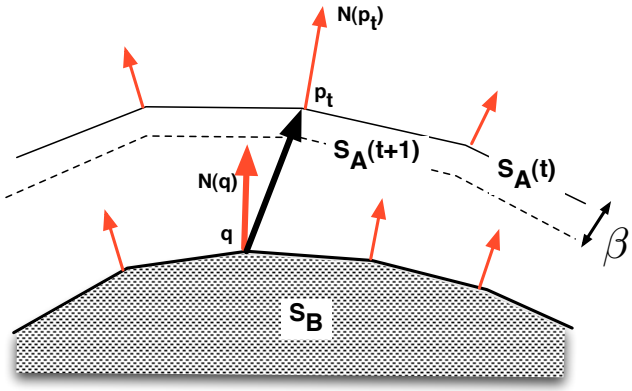



Figure 10: Mesh Morphing evolution step. The surface $S_A$ evolves from time $t$ to time $t+1$ towards $S_B$. If for point $p \in S_A$, the closest point in $S_B$ is $q$, then the point $p$ will evolve along its normal with a magnitude of $(p-q) \cdot N(p)$, thresholded by a maximum user set evolution magnitude.

In the general case, in order to calculate an exact distance function $\gamma_B$, one would have to consider the distance from a query point to the each of the facets of the mesh (representing the surface $S_B$), keeping the closest distance. This process will take $O(N_F)$, where $N_F$ represents the number of facets. This is a fairly expensive computation, which will have to be performed at each iteration throughout the evolution for every vertex.

There exists a large number of methods for computing 3-D distance fields. For a recent survey, please consult [31]. As per [31], the methods can be classified according to two criteria. According to the first criterion, they can be:

- **Chamfer methods**, where the new distance of a voxel is computed from the distances of its neighbors by adding values from a distance template;
- **vector methods** where each voxel stores a vector to its nearest surface point and the vector at an unprocessed voxel is computed from the vectors at its neighbors by means of a vector template and
- **Eikonal solvers**, where the distance of a voxel is computed by a first or second order estimator from the distances of it's neighbors.

According to the second criterion, the distances can be propagated throughout the volume in a:

- **sweeping scheme**, when the propagation starts in one corner of the volume and proceeds in a voxel-by-voxel, row-by-row fashion to the opposite end, typically requiring multiple passes in different directions, or in a
- **wavefront scheme**, when the distances are propagating from the initial surface in the order of increasing distances until all voxels are covered.

For our testing purposes, we propose an approximation/heuristic using the distance to the closest vertex point, as illustrated in Figure 10. If for a point $p \in S_A$, the closest point from $S_B$ is $q$, the evolution equation for point $p$ is:

$$\gamma_B(p) = (q - p) \cdot N(p), \tag{2}$$

where $\gamma$ was introduced in (1). Note that the vector magnitude will be thresholded to a maximum of $\alpha \cdot e_{avg}(p)$, as per step 2 of the generic mesh evolution algorithm, described in Figure 9. The distance and sign from a query point are computed on the fly, as supposed to being stored in a distance field 3-D grid. The computation time is reduced drastically due to the use of proper search structures. The search time for the nearest neighbor is $O(log(N_V))$, where $N_V$ represents the number of vertices. There is an initial overhead of $O(N_V log(N_V))$ of building the search tree. In practice, we have used the implementation of [28] available in CGAL. Note that if the target surface $S_B$ contains a good enough mesh resolution, this approximation is very close to the true signed distance function. Also, if the accuracy of distance field computation is of concern, more exact implementations could be adopted [31].

In the case of sufficient sampling, the current approximation will return a vertex belonging to the closest triangle where the true projection would be. Thus, the error bound is the distance between the vertex and the projection. In practice, however, we do not use the actual distance, but its sign, in order to establish the direction of the evolution. This makes the current approximation fit for our purpose. Alternatively, one could easily verify all the incident triangles to the closest vertex to establish the true distance function, if the application requires it, keeping in mind that the sufficient sampling condition still applies.

The current heuristic only makes use of the mesh vertices of $S_B$, together with their associated normals. This has the great advantage of being able to be applied in the current formulation, not only to meshes, but also to oriented 3-D points. This would allow one to morph an initial mesh $S_A$ towards a set of oriented 3-D points $P_B$. If orientation information is not available, it can be estimated from neighboring points using principal component analysis [29]. Alternatively, in the context of multi-view stereo, it can be obtained via a minimization scheme [23].

## 6.3 Results

In Figure 11 we present results obtained with four test cases, entitled "Genus 3", "Thoruses", "Knots In" and "Knots Out". As it can be observed, the algorithm successfully deals with merge and split operations as well as handling multiple connected components. The average computation time per iteration on a 2.6 GHZ Intel Core2Duo processor varies between 0.2 to 1.6 seconds, depending on the number of facets and on the number of intersections. More detailed statistics are presented in Table 1.

In terms of parameter settings with respect to the generalized mesh evolution framework depicted in Figure 9 within which we casted the current mesh morphing algorithm, we considered $t = 1$ for the timestep, $\alpha = 0.2$ the average edge size $e_{avg}$ for maximum movement amplitude and $\beta = 0.1$ for the smoothing term. Additionally, the original meshes had a constant mesh resolution. Hence, we set the edge thresholds to $e_1 = 0.7 \cdot e_{avg}$ and $e_2 = 1.5 \cdot e_{avg}$.

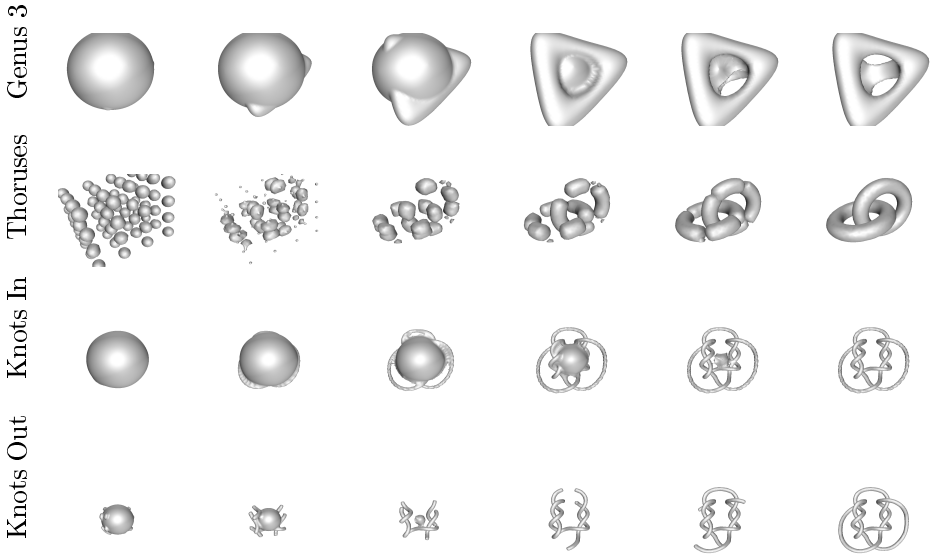



Figure 11: Mesh morphing examples. Different steps for various test cases. Each column corresponds to a test case. The first row represents the first iteration, whereas the last row represents the last iteration.

| Dataset | Genus 3 | Thoruses | Knots In | Knots Out |
|---|---|---|---|---|
| Iterations | 54 | 37 | 119 | 430 |
| # Facets | 4764.14 | 6296.33 | 13244.25 | 3873.11 |
| # Intersections | 33.88 | 22.67 | 101.52 | 4.86 |
| Time (TransforMesh) | 0.65 sec | 0.81 sec | 1.63 sec | 0.18 sec |
| Time (total) | 1.42 sec | 1.78 sec | 3.58 sec | 0.89 sec |

Table 1: Mesh morphing statistics for different datasets. The reported values presented in the bottom four rows represent average values accumulated across the iterations. The running time is recorded on a 2.6 GHz Intel Core2Duo processor.

Additional results of mesh morphing are presented in Figure 12, with meshes obtained from 3-D reconstructions from multiple cameras, in the context non-rigid surface tracking [56].

# 7 Multi-View 3-D Reconstruction

In this section we explain how our method fits into the multi-view/image-based 3D reconstruction pipeline. The problem of reconstructing an object from images gathered with a large number of cameras has received a lot of attention in the recent past [26, 50, 53, 57, 57]. It is interesting to notice that, until recently, there were only a handful of mesh-based solutions to the surface reconstruction problem. This is mainly due to the topological problems raised by existing mesh-evolution methods. In particular, topological-preserving approaches are ill adapted to the problem of surface reconstruction. Topological-adaptive al-

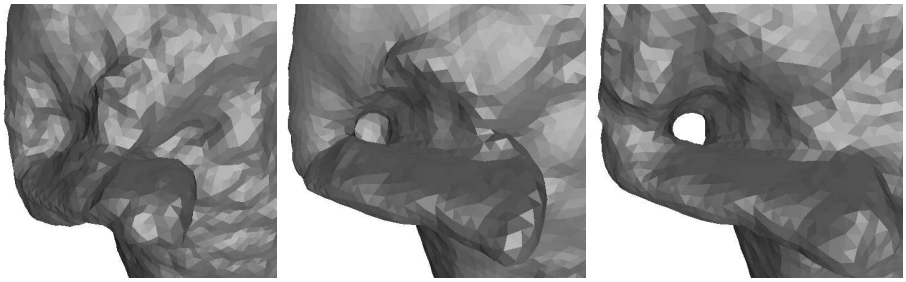


Figure 12: Example of topological changes during mesh morphing in surface tracking. The source surface $S'_t$, shown to the left, is the result of a deformation of the original mesh $S_t$ at time $t$ such that it matches closely the mesh $S_{t+1}$ at $t+1$, shown to the right. The mesh morphing process ensures the proper handling of topological changes (i.e. the whole formation in the arm region).

gorithms, such as TransforMesh provide a more flexible solution that allows to better resolve for local details using topological changes.

In [50] the multi-view reconstruction problem is cast into an energy minimization problem using photometric constraints. It is well known that topological changes may take place during the minimization process, e.g., Figure 15. Surface evolution based on a level-set formulation is proposed in [50]. Our contribution to this class of reconstruction methods is to extend such surface evolution approaches to meshes that allow to focus on the shape's surface instead of a bounding volume.

The method described below was applied both to visual hulls, e.g., [21] and to sparse point-based 3-D data, e.g., [60]. The former representation constitutes the initial mesh that needs be improved using photometric information from the available images. The latter representation can be easily turned into a rough mesh using [6] for example.

## 7.1 Methodology

The initial meshed surface corresponds to an extended bounding box obtained using image silhouettes and a geometric approach that involves cone intersections in 3-D, i.e., [22]. Such a mesh is only a coarse approximation of the observed surface. One main limitation of visual hull approaches is that they do not recover concave regions. The initial surface can be improved by considering photometric information in the images. The underlying principle is that, with a correct geometry, and under the Lambertian surface assumption, the mesh should be photo-consistent, i.e., its projections in the images should have similar photometric information [5].

The photometric constraints are casted into an energy minimization framework, using a similarity measure between pairs of cameras that are close to each other, as proposed by Pons *et al.* [50]. The problem is solved in practice via gradient descent. $E_{img}$ is the derivative of the local photoconsistency term, in the normal direction, that can be computed using several methods. To compute such a derivative, we use one of the most efficient approaches [50], based on the normalized cross-correlation. The evolution equation is in this case:

$$\vec{\mathcal{F}}_{reconstruction} = \frac{\partial S}{\partial t} = E_{img}(x)\mathbf{N}(x). \tag{3}$$

In [50] the surface evolution is implemented within the level-set framework. We extended it to meshes using the TransforMesh algorithm. The level-set solution performs surface evolution using a coarse-to-fine approach in order to escape from local minima. Traditionally, in level-set approaches, the implicit function that embeds the surface $S$ is discretized evenly on a 3-D grid. As a side-effect, all the facets of the recovered surface are of approximately equal triangle size. In contrast, mesh based approaches do not impose such a constraint and allow facets of all sizes on the evolving surface. This is particularly useful when starting from rough surface estimates, such as visual hulls, where the initial mesh contains triangles of all dimensions. In addition, the dimension of visual facets appears to be a relevant information since regions where the visual reconstruction is less accurate, i.e. concave regions on the observed surface, are described by bigger facets on the visual hull. Thus, we adopt an approach in which bigger triangles are processed first, until they are stabilized, then the whole process is repeated at a finer scale.

The mesh evolution algorithm depicted in Figure 9 requires a number of parameters to be set in advance. In the case of 3-D reconstruction we used the following parameter settings in all our examples: $t = 0.001$ for the time step, $\alpha = 0.1$ and $e_{avg}$ for the maximum movement amplitude and $\beta = 0.1$ for the smoothing term. The meshes have an adaptive mesh resolution. As mentioned earlier, we ran the algorithm at different scales, starting from scale $s_{max}$ to $s_{min} = 1$ in $\lambda = \sqrt{2}$ decrements. For each scale $s_i$, the input images and camera matrices are downscaled accordingly. The appropriate edge size interval is set to $e_1 = edgeSize(1,1)$ $e_{2i} = edgeSize(5,i)$, where $edgeSize(p_1,p_2)$ is a function that computes the desired edge size such that it has $p_1$ pixels using images at scales $p_2$. The initial scale $s_{max}$ is computed such that the largest edges of the initial mesh measure 5 pixels when projected into the images at scale $s_{max}$. When the finer scale is reached, new iterations are run by decreasing $e_2$ from $edgeSize(5,1)$ to $edgeSize(2,1)$ in $\lambda = \sqrt{2}$ decrements.

| | Temple Ring | | Temple Sparse | | Dino Ring | | Dino Sparse | |
|---|---|---|---|---|---|---|---|---|
| | Acc. | Compl. | Acc. | Compl. | Acc. | Compl. | Acc. | Compl. |
| [50] | 0.60 | 99.5% | 0.90 | 95.4% | 0.55 | 99.0% | 0.71 | 97.7% |
| [23] | 0.47 | 99.6% | **0.63** | **99.3%** | **0.28** | **99.8%** | **0.37** | **99.2%** |
| [26] | 0.52 | 99.5% | 0.75 | 95.3% | 0.45 | 97.9% | 0.60 | 98.52% |
| [57] | **0.45** | **99.8%** | | | 0.53 | 99.7% | | |
| Us | 0.55 | 99.2% | 0.78 | 95.8% | 0.42 | 98.6% | 0.45 | **99.2%** |

Table 2: Middleburry 3-D Reconstruction Results. Accuracy (expressed in milimeters): the distance $d$ (in mm) that brings 90% of the result $R$ within the ground-truth surface G. Completeness: the percentage of $G$ that lies within 1.25mm of $R$.

### 7.2 Results

We have tested the mesh evolution algorithm with the datasets provided by the Multi-View Stereo evaluation site [53] [2]. The ground-truth is obtained from laser-scans. Comparative and detailed results are extracted from the Middlebury website and are presented in Table 2. The table includes results from Furukawa and Ponce [23], Pons *et al.* [50], Vu *et al.* [57] and Hernandez and Schmitt [26]; all these methods yield state-of-the-art results. The differences between all these methods are very small, ranging between $0.01mm$ to $0.1mm$. Some of our reconstruction results are shown in Figure 13 and Figure 14. While Vu et al. [57] used the same energy functional as part of their 3-D reconstruction pipeline, their improved results are mostly due to the fact that the mesh regularization term takes into account photo-consistency.

Figure 14 shows the results obtained with our method when starting with very rough meshes that correspond to coarse triangulations obtained from a sparse set of 3-D points. An example of how TransforMeshhandles topological changes is shown in Figure 15. This figure shows a typical evolution scenario where there are more "topological problems" at the beginning; As the algorithm converges, self-intersections barely occur.

Finally, Figure 16 shows results obtained with the Man-dance sequence publicly available from the Multiple-video database of the PERCEPTION group at INRIA [3].

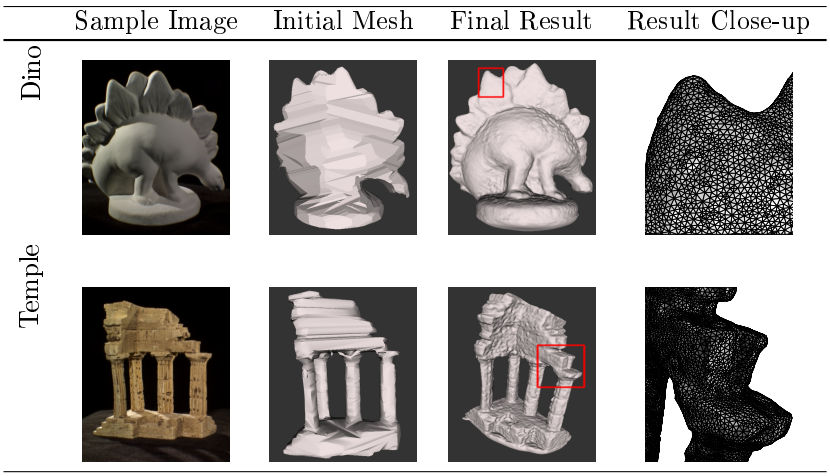



Figure 13: Reconstruction Results for the Middleburry multiview dataset (dino case and temple case)

## 8 Conclusion

In this paper, we proposed a geometry-driven self-intersection removal algorithm for triangular meshes, able to handle topological changes in an intuitive and efficient way. Our main contribution with respect to the existing mesh-evolution

[2] http://vision.middlebury.edu/mview/
[3] http://4drepository.inrialpes.fr/

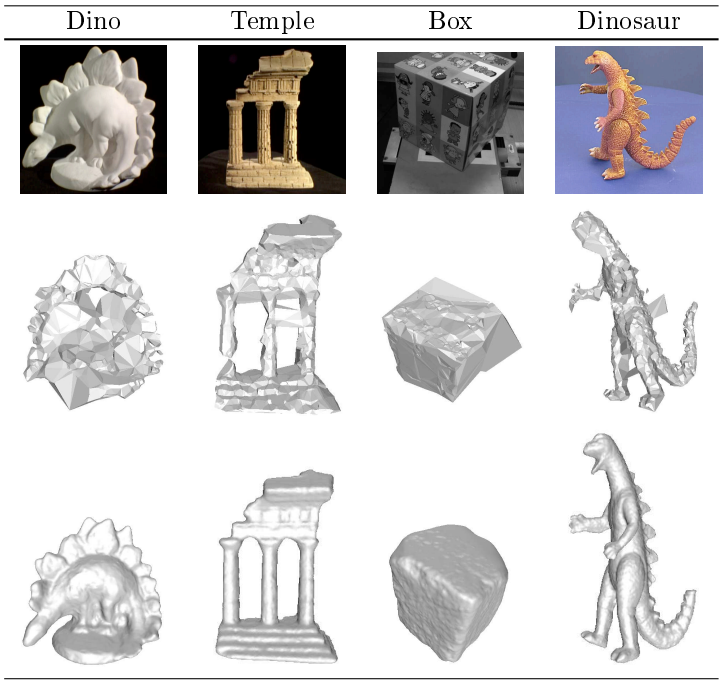



Figure 14: Additional dense reconstruction results. First Row: Sample Input Image; Second Row: A rough mesh obtained using using PowerCrust [6] from the sparse 3-D points, reconstructed using [60]; Third Row: the final dense reconstruction after surface evolution.

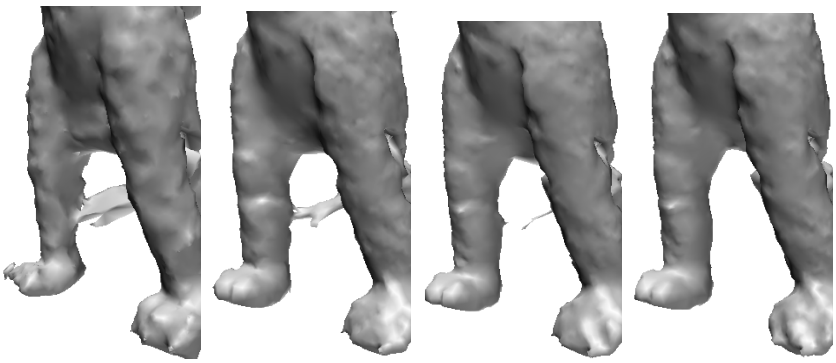


Figure 15: Example of topological changes during in 3-D reconstruction for the dinosaur sequence, introduced in Figure 14. The start-up surface, obtained from triangulated 3-D points via PowerCrust [6] contains several topological errors (i.e. the extra branch connecting the dinosaur's limbs). They are corrected during the surface evolution, ash shown in the right most image.

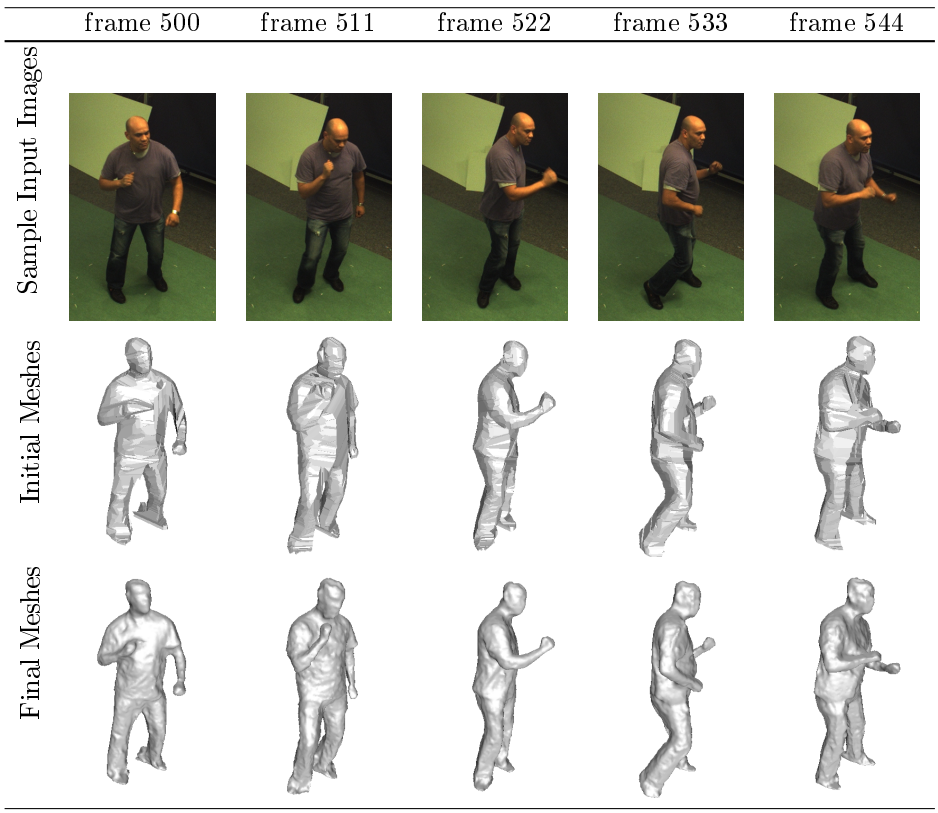



Figure 16: Results for the Man-dance sequence from INRIA. At each time-step the mesh is reconstructed from 34 cameras.

methods is to provide a purely geometric mesh-based solution that is correct, that does not constrain meshes and that allows for facets of all sizes as well as for topological changes.

The TransforMesh algorithm was plugged into a generic mesh-evolution framework, thus allowing to address two challenging problems within a topology-adaptive approach: surface morphing and multi-view image-based 3-D reconstruction.

We provided both a detailed description of the proposed algorithm as well as an in-depth analysis of its convergence and performances (numerical stability and time complexity). In the case of surface morphing, we showed that TransforMesh can deal with challenging topological cases.

The 3-D reconstruction method that we described and which is based on mesh evolution is extremely versatile. The method recovers a correct discrete surface geometry starting from very coarse approximations, such as visual hulls or sparse sets of 3-D points. The 3-D reconstruction results are of comparable quality with state-of-the-art methods recently developed by computer vision researchers.

## Acknowledgments

We thank Jean-Philippe Pons and Renaud Keriven for providing the source code for the gradient computation needed by multi-view 3D reconstruction.

## References

[1] D. Adalsteinsson and J. Senthian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[2] M. Aftosmis, M. Berger, and J. Melton. Robust and efficient cartesian mesh generation for component-based geometry. In *AIAA Paper 97-0196.*, 1997.

[3] A. Agrawal and A. Requicha. A paradigm for the robust design of algorithms for geometric modeling. *Computer Graphics Forum*, 13(3):33–44, 1994.

[4] P. Alliez, S. Tayeb, and C. Wormser. Aabb tree. In CGAL Editorial Board, editor, *CGAL User and Reference Manual.* 3.5 edition, 2009.

[5] S. M. S. amd C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173, 1999.

[6] N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform,. *Computational Geometry: Theory and Applications*, 19(2-3):127–153, 2001.

[7] B. G. Baumgart. *Geometric Modeling for Computer Vision.* PhD thesis, Standford University, 1974.

[8] H. Biermann, D. Kristjansson, and D. Zorin. Approximate boolean operations on free-form solids. In *Proceedings of SIGGRAPH*, pages 185–194, 2001.

[9] C. E. Board. *CGAL-3.2 User and Reference Manual*, 2006.

[10] I. C. Braid, R. C. Hillyard, and I. A. Stroud. Stepwise construction of polyhedra in geometric modelling. *Mathematical Methods in Computer Graphics and Design*, 1978.

[11] D. E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Transaction on Visualization and Computer Graphics*, 7(2):173–192, 2001.

[12] P. Carvalho and P. Cavalcanti. Point in polyhedron testing using spherical polygons. In *Graphics Gem V*, chapter II.2, pages 42–49. Academic Press, 1995.

[13] G. Celniker and D. Gossard. Deformable curve and surface finite-elements for free-form shape design. In *Computer Graphics*, volume 25, pages 257–266, 1991.

[14] H. Delingette, M. Herbert, and K. Ikeuchi. Shape representation and image segmentation using deformable surfaces. *Image and Vision Computing*, pages 132–145, 1992.

[15] Y. Duan, L. Yang, H. Qin, and D. Samara. Shape reconstruction from 3D and 2D data using pde-based deformable surfaces. In *Proceedings of European Conference on Computer Vision*, volume 3, pages 238–251, 2004.

[16] H. Edelsbrunner and E. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[17] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[18] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of SIGGRAPH*, pages 736–744, 2002.

[19] J. Foley, A. V. Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice, second edition*. Addison Wesley, 1996.

[20] J. D. Foley, A. van Dam, S. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, 1990.

[21] J.-S. Franco and E. Boyer. Exact polyhedral visual hulls. In *Proceedings of the British Machine Vision Conference*, volume 1, pages 329–338, September 2003.

[22] J. S. Franco and E. Boyer. Efficient polyhedral modeling from silhouettes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(3):414–427,, March 2009.

[23] Y. Furukawa and J. Ponce. Accurate, dense and robust multi-view stereopsis. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page to appear, 2009.

[24] J. Goldfeather, J. P. M. Hultquist, and H. Fuchs. Fast constructive-solid geometry display in the pixel-powers graphics system. In *Proceedings of SIGGRAPH*, volume 20, pages 107–116, 7 1986.

[25] A. Gueziec, G. Taubin, F. Lazarus, and B. Horn. Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE Transaction on Visualization and Computer Graphics*, 7(2):136–151, 2001.

[26] C. E. Hernández and F. Schmitt. Silhouette and stereo fusion for 3-D object modeling. *Computer Vision and Image Understanding*, 96(3):367–392, 2004.

[27] S. Hert and M. Seel. dD convex hulls and delaunay triangulations. In C. E. Board, editor, *CGAL-3.2 User and Reference Manual*. 2006.

[28] G. R. Hjaltason and H. Samet. Ranking in spatial databases. *Symposium on Large Spatial Databases*, pages 83–95, 1995.

[29] H. Hoppe, T. DeRose, T. Duchamp, J. MCDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of SIGGRAPH*, 1992.

[30] P. M. Hubbard. Constructive solid geometry for triangulated polyhedra. Technical Report CS-90-07, Department of Computer Science, Brown University, 1 1990.

[31] M. W. Jones, J. A. Bærentzen, and M. Sramek. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Compute Graphics*, 12(4):581–599, July/August 2006.

[32] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In *Proceedings of SIGGRAPH*, 2002.

[33] W. Jung, H. Shin, and B. K. Choi. Self-intersection removal in triangular mesh offsetting. *Computer-Aided Design and Applications*, 1(1-4):477–484, 2004.

[34] L. Kettner, A. Meyer, and A. Zomorodian. Intersecting sequences of dD iso-oriented boxes. In C. E. Board, editor, *CGAL-3.2 User and Reference Manual*. 2006.

[35] L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Proceedings of Eurographics*, pages 249–260, 2000.

[36] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH*, pages 57–66, 2001.

[37] J.-O. Lachaud and B. Taton. Deformable model with adaptive mesh and automated topology changes. In *Proceedings of the Fourth International Conference on 3-D Digital Imaging and Modeling*, 2003.

[38] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[39] F. Losasso, R. Fedkiw, and S. Osher. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids*, 35(10):995–1010, 2006.

[40] M. Mäntylä. Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics*, 5(1):1–29, 1986.

[41] T. McInerney and D. Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, 2000.

[42] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential geometry operators for triangulated 2-dimensional manifolds. In *Proceedings of VisMath*, 2002.

[43] A. L. Nathan Litke and P. Schröder. Trimming for subdivision surfaces. Technical report, Caltech, 2000.

[44] Y. Ohtake, A. Belyaev, and A. Pasko. Dynamic mesh optimization for polygonized implicit surfaces with sharp features. *The Visual Computer*, 19:115–126, 2003.

[45] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer, 2003.

[46] S. Osher and N. Paragios. *Geometric Level Set Methods in Imaging, Vision, and Graphics.* Springer, 2003.

[47] S. Osher and J. Senthian. Front propagating with curvature dependent speed: algorithms based on the Hamilton-Jacobi formulation. *Journal of computational Physics*, 79(1):12–49, 1988.

[48] J.-J. Park, T. McInerney, D. Terzopoulos, and M.-H. Kim. A non-self-intersection adaptive deformable surface for complex doundary extraction from volumetric images. *Computer & Graphics*, 25:421–440, 2001.

[49] J.-P. Pons and J.-D. Boissonnat. Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Minneapolis, USA, Jun 2007.

[50] J.-P. Pons, R. Keriven, and O. Faugeras. Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. *International Journal of Computer Vision*, 72(2):179 – 193, 2007.

[51] A. Rappoport and S. Spitz. Interactive boolean operations for conceptual design of 3-D solids. In *Proceedings of SIGGRAPH*, pages 269–278, 1997.

[52] J. Rossignac and A. Requicha. *Encyclopedia of Electrical and Electronics Engineering*, chapter Solid Modeling. John Wiley and Sons, 1999.

[53] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 519–526, 2006.

[54] H. Shin, J. C. Park, B. K. Choi, Y. C. Chung, and S. Rhee. Efficient topology construction from triangle soup. In *Proceedings of the Geometric Modeling and PRocessing*, 2004.

[55] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146–159, 1994.

[56] K. Varanasi, A. Zaharescu, E. Boyer, and R. P. Horaud. Temporal surface tracking using mesh evolution. In *Proceedings of European Conference on Computer Vision*, 2008.

[57] H. Vu, R. Keriven, P. Labatut, and J.-P. Pons. Towards high-resolution large-scale multi-view stereo. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Jun 2009.

[58] C. Wojtan, N. Thurey, M. Gross, and G. Turk. Deforming meshes that split and merge. In *Proceedings of SIGGRAPH*, 2009.

[59] C. Wojtan and G. Turk. Fast viscoelastic behavior with thin features. In *Proceedings of SIGGRAPH*, 2008.

[60] A. Zaharescu and R. P. Horaud. Robust factorization methods using a gaussian/uniform mixture model. *International Journal of Computer Vision*, March 2009.

[61] A. Zomorodian and H. Edelsbrunner. Fast software for box intersection. *International Journal of Compational Geometry and Applications*, 12(1-2):143–172, 2002.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399