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

Andrei Zaharescu, Edmond Boyer, Radu Horaud

Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multiview Reconstruction

Andrei Zaharescu, *Member*, *IEEE*, Edmond Boyer, and Radu Horaud

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 propose a mesh evolution framework based on this algorithm. Numerous shape modeling 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 modeling, 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 3D reconstruction.

Index Terms—Mesh, surface, manifold mesh, triangulated mesh, mesh evolution, deformable objects, morphing, 3D reconstruction.

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 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 representations when modeling shapes, are seldom considered in such evolution schemes. The main reasons for that are the inherent difficulty in handling 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 2D 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 present two applications: mesh morphing and variational multiview 3D reconstruction.

1.1 Literature Review

As a result of the large interest in 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* [1], [2], 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 [3], [4], [5], [6]. Such an embedding within an implicit function allows us 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 [7] as well as in other fields, such as computational

• *The authors are with the INRIA Grenoble Rhone-Alpes, 655 Avenue de l'Europe, 38330 Montbonnot Saint-Martin, France.*
E-mail: {andrei.zaharescu, edmond.boyer, radu.horaud}@inrialpes.fr.

Manuscript received 23 Nov. 2009; revised 30 Mar. 2010; accepted 27 Apr. 2010; published online 1 June 2010.
Recommended for acceptance by S.B. Kang.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2009-11-0777.
Digital Object Identifier no. 10.1109/TPAMI.2010.116.

fluid dynamics [8] and computer animations of fluids [9]. 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 [10] or octree-based implementations [11] only partially overcome this issue. In addition, as shown by Enright et al. [12], 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 are adaptive resolution and compact representation, as well as the ability to directly handle nongeometric 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 nonintersections and fixed topology were explicitly enforced [13], [14]. 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 [15] 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. [16] 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. [17]. Another extension is proposed by Brochu and Bridson [18], with a greater focus on the mesh optimization technique and guarantees. Alternatively, Pons and Boissonat [19] proposed a mesh approach based on a restricted 3D 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. [20] and Jung et al. [21] are also related to our work. The algorithm in [20] 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 [21] 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. [22] 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 exactly matching the extracted isosurface to the original mesh, among others.

In addition to the two above categories, it is worth also mentioning **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) [23], [24] and Boundary Representation (B-Rep) [25], [26]. CSG methods represent shapes as a combination of elementary object shapes based on Boolean operations. Alternatively, B-Rep methods adopt the more natural approach of representing the object boundary using vertices, edges, and facets [27], [28]. Each representation has its advantages. While Boolean operations on CSG objects are straightforward, a lot of computational effort is required to render CSG objects [29], [30]. On the other hand, it is much more difficult to implement Boolean operations on boundary representations (multiresolution surfaces) [31], [32], whereas interactive rendering is trivial. While these methods propose solutions 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 [20], [21] to any topology changes resulting from mesh deformation, including *merges*, *splits*, *hole formations*, and *hole losses*, e.g., Fig. 7. The main contribution is that given an input orientable mesh with self-intersections, the algorithm provides a 2D 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 multiview 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 a topology-adaptive scheme is more general, and hence, better adapts to challenging applications such as

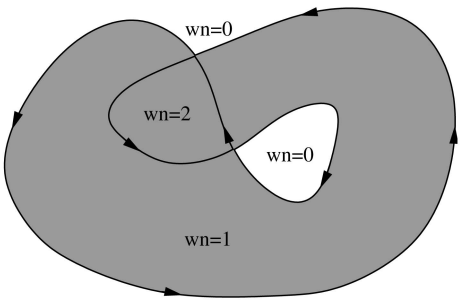


Fig. 1. Interior and exterior regions delimited by an oriented primitive. A point is on the exterior when it belongs to a region with a winding number wn equal to 0, on the interior otherwise.

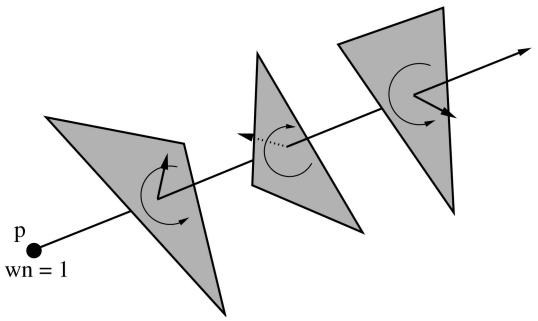


Fig. 2. The winding number at p can be obtained by summing the dot product signs with face normals along any ray from p.

3D reconstruction using multiple images and nonrigid surface tracking.

Recent image-based reconstruction methods [33] 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 paper 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 as 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, as well as two sample applications: mesh morphing in Section 5.2 and 3D reconstruction in Section 5.3. Finally, we conclude in Section 6.

TransforMesh is available as an open-source software package (OSS) under a general public license (GPL).[1] Additional examples are available in the supplemental material, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2010.116.

2 Background

Before we introduce the TransforMesh algorithm, we clarify 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 2D 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 toward 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 a rule simply consists of counting the number of intersections of a ray, emanating from a point, with the delimiting primitives. 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, with positive or negative orientations depending on the sign of the winding number. Fig. 1 depicts this principle in 2D.

To compute this number, two strategies can be followed. A first strategy consists of 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 [34]. The sum will be equal to 0 for a point on the exterior and a multiple of 2π and 4π in 3D for a point on the interior. Another strategy considers a ray from a point and its intersections with the primitive [35]. Each intersection is assigned a value $+1$ or -1 according to the sign of the dot product of the ray direction with the normal to the primitive at the intersection. If this sign is negative, the value is -1 and $+1$ otherwise, see Fig. 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 toward its normal direction and we sum the values -1 and $+1$ obtained at the intersections with

1. http://mvviewer.gforge.inria.fr/.

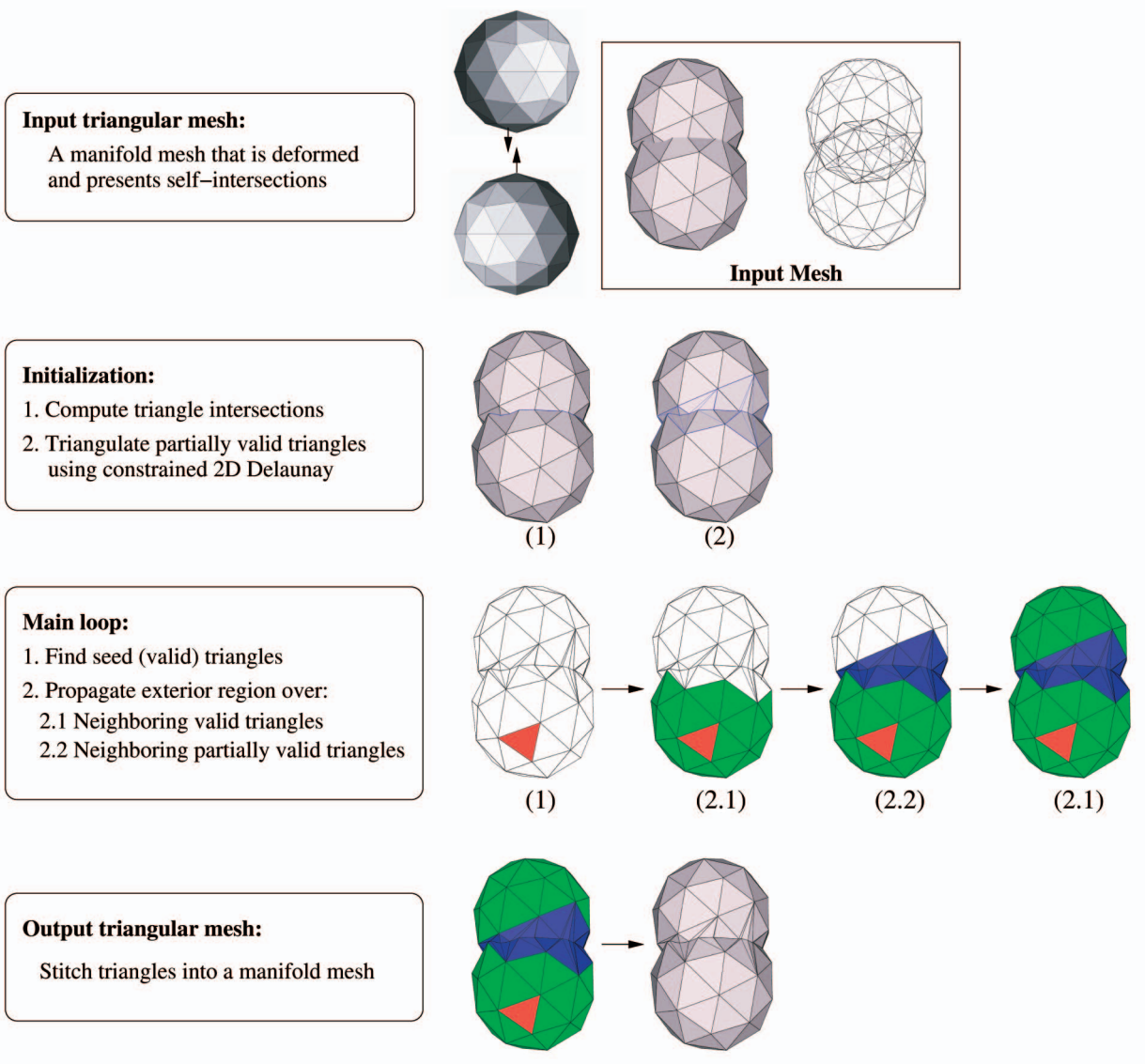


Fig. 3. Overview of TransforMesh.

other faces along the ray. The face is on the exterior when this sum is 0.

We then call 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 subparts, some of which are 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 photoconsistency, 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., 2D 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 of 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. Fig. 3 illustrates the algorithm, and the different steps are detailed in the following.

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 be needed later on 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

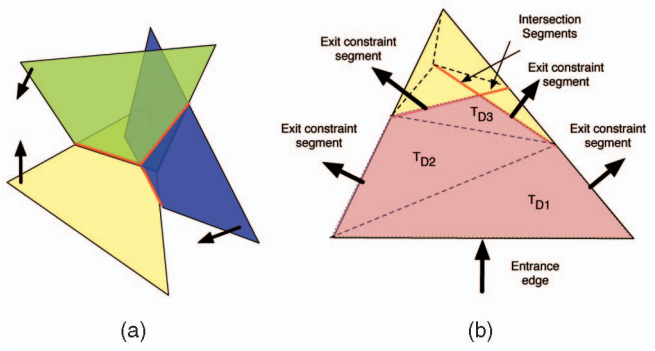


Fig. 4. Partially valid triangle traversal. (a) The intersections with all other triangles are computed for each intersecting triangle. (b) Closeup of the bottom triangle in (a). The local geometry is redefined using a constrained 2D Delaunay triangulation that ensures the presence of the original triangle 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.

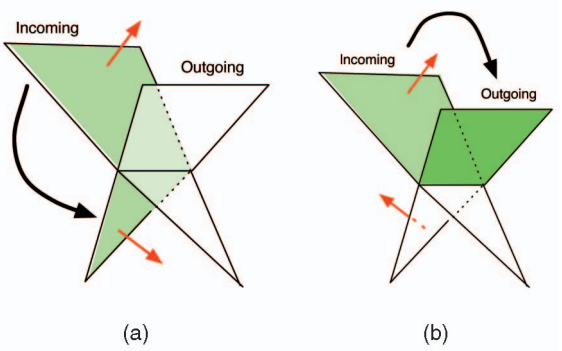


Fig. 5. The two partially valid triangle-crossing cases.

use the fast box intersection method implemented in [36] and described in [37]. 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 [20]. Following this idea, we present here the substeps 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 then details how to traverse the valid subparts of intersection triangles as well as how to cross from one intersecting triangle to the other. The local subparts are triangulated using a constrained 2D Delaunay triangulation. The underlying idea that guides this step is to propagate the normal information from the seed triangles using the local geometry. The algorithm seeks to maintain the orientation of the original surface. When generating the output mesh, the orientation of the valid triangles is preserved. The newly formed subtriangles (partially valid triangles) will inherit the orientations of the parent triangles.

3.2.1 Seed-Triangle Finding

A seed triangle is defined as a nonvisited 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 [38] for the ray-to-triangles intersection test.

3.2.2 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 nonvisited and has no intersections, then it is marked as valid; if the neighboring triangle is nonvisited 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.

3.2.3 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 of the intersection segments between this triangle and all of the other triangles. Let $S_t = \{s_{ti}\}$ represent all of the intersection segments between triangle t and the other triangles. In addition, let $H_t = \{h_{tj} \mid \text{for } j = 1..3\}$ represent the triangle half-edges. A constrained 2D triangulation performed in the triangle plane, using [39], ensures that all segments in both S_t and H_t appear in the new triangular mesh structure and 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 Fig. 4.

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 subtriangles would preserve the watertightness constraint of the manifold. This condition can also be cast as follows: The normals of the two subtriangles should be opposed to each other when the two subtriangles are "folded" on the common edge. A visual representation of the two cases is shown in Fig. 5. The triangles on the other side of the exit constraint edges will be appropriately marked as valid 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 (nonconnected) exterior components. However, each subtriangle formed by the local retriangulation 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.

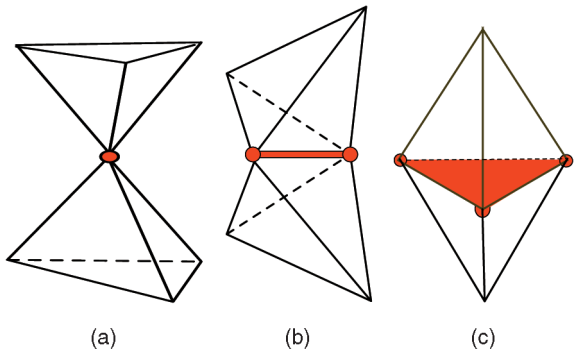


Fig. 6. Special cases encountered while stitching a triangle soup. (a) Singular vertex. (b) Singular edge. (c) Singular face.

3.3 Triangle Stitching

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 3D triangle soup in order to obtain a valid mesh which is manifold. We adopt a method similar in spirit to [40], [41]. In most cases, this is a straightforward 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 constraints and produce locally nonmanifold structures. The special cases, as shown in Fig. 6, arise from performing stitching in places where the original structure should have been maintained. We adopt the naming convention from [40], calling them the singular vertex case, the singular edge case, and the singular face case. All cases are easily identified by performing local operations.

3.3.1 Singular Vertex Case (Fig. 6a)

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. To detect these cases, the algorithm 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 nonvisited neighbors that are also incident to v (neighboring triangles are chosen based on the available mesh connectivity); the process is repeated until all of the neighboring facets are processed; if by doing so, we exhausted all of the neighboring facets, vertex v is nonsingular; otherwise, it is singular, so a copy of it is created and added to all the remaining nonvisited facets. The process is repeated until all of the incident facets are visited.

3.3.2 Singular Edge Case (Fig. 6b)

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.

3.3.3 Singular Triangle Case (Fig. 6c)

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.

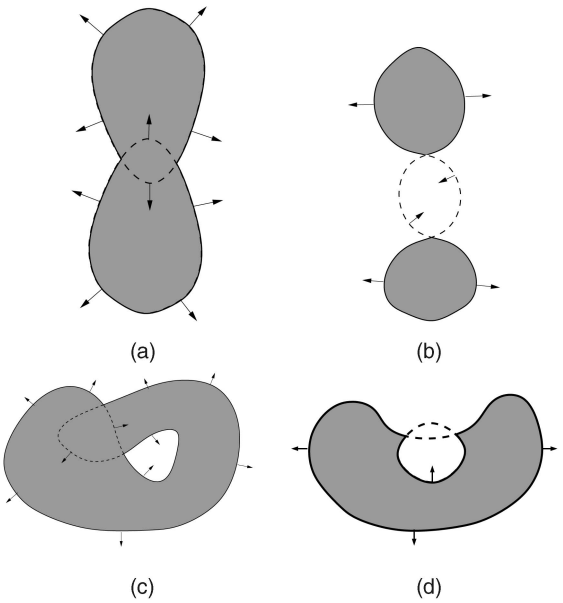


Fig. 7. Topological changes (2D simplified view). Regions delimited by nonexterior faces (dashed lines) are eliminated by the algorithm. (a) Merge. (b) Split. (c) Hole formation. (d) Hole loss.

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.

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 that it correctly handles 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 Fig. 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 Fig. 10 for examples). Also note that regions delimited by nonexterior faces, as shown by dashed lines in Fig. 7, are eliminated by the algorithm.

The partially valid triangle crossing technique described earlier in Section 3.2 and detailed in Fig. 5 ensures a *natural* handling of these topological changes that plagued most of the mesh approaches until now. The *merge* case scenario, as shown in Fig. 7a, 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, hence defining an inverted inside region with a negative winding number. Such a case is depicted in Fig. 7b, in a mesh morphing scenario, where the initial surface has a 1-connected component and the destination has 2-connected components.

The two other examples, hole formation and loss, are less frequent. While handled by the algorithm, as mentioned earlier, inside valid faces, e.g., Fig. 7c, are usually not considered in the surface evolution processes described here.

4.2 Guarantees

Given that the input mesh is a 2D compact oriented manifold that has been deformed by a motion field and assuming exact computations (see Section 4.3), TransforMesh will recover 2D compact oriented manifold components. The 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 2D manifold correctness is guaranteed by identifying and correcting all of the possible 2D manifold neighborhood violations when performing triangle stitching (singular vertex, singular edge, and singular facet).

The algorithm preserves the geometry and orientation 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 and when disambiguating the boundary cases. Boundary cases are defined as nontypical cases. For example, in a triangle-triangle intersection test, the typical cases are when the intersection is a 2D segment or when there is no intersection at all, whereas the boundary cases are when the intersection is a point, a 2D polygon, or a line segment on one of the triangle edges.

The boundary cases are disambiguated using the *simulation of simplicity* technique of virtual perturbations [42]. It involves inducing a small vertex perturbation locally, which will force the boundary triangle-triangle intersections into one of the classical cases.

Mesh offsetting was used as a way to perturb the original mesh, where each vertex is moved along its normal by a small step. While it is impossible to guarantee that a perturbation will completely eliminate the boundary cases, the algorithm detects the reoccurrence of such a boundary condition and it applies another perturbation. Note that any other mesh perturbation can be used.

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. *Simulation of simplicity* is complementary to the approach proposed by Mäntylä in [43], where all of the possible boundary scenarios are handled explicitly, making the method more suitable for applications where exact intersections are required (i.e., Boolean operations with CAD models).

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 due to the fast box intersection method used, described in [37] and implemented in [36]. 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 percent 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 multithreading, 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 Strategy

Alternatively, one could use the winding test, described in Section 2, in order to replace the propagation step described in Section 3.2. Instead of growing the valid region outside, it would test all of the existing triangles and subtriangles obtained from local Delaunay triangulations and only choose the triangles that reside on the exterior, after which it would proceed to the final triangle stitching step. However, this static scheme would take considerably longer time since it requires the same initial time $O(n \log^3 n)$ to compute all of the triangle intersections and local Delaunay triangulations, followed by the additional time required for the valid triangle test, which is not negligible.

4.6 Extension to Open Surfaces

Consider surfaces with holes, where a hole on a mesh is a closed ring of half-edges. The region growing step presented before still applies, assuming an initial outside face. In order to correctly identify such a face using the winding rule, holes are temporally closed, just for the seed-triangle finding step. Any hole filling method could be used for that purpose; however, we use, in practice, a simple scheme where all of the half-edges along a hole contour are connected to a newly added vertex. Since the input mesh is the deformation of an oriented manifold and unless the newly added vertices make all faces interior, the algorithm will be able to find an initial exterior face from which the outside label can be normally propagated on the original faces. Note that the situation where all faces are labeled as interior happens when the added vertex is located on the back of all faces. This will seldom happen in general and can be, in any case, detected. Such an extension allows one to deal with open surfaces and it also enables local self-intersections removal on very large meshes.

Algorithm: Generic Mesh Evolution with 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;
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;
   b) *Vertex Valence Optimization*: improve the quality of the triangles;
   c) *Smoothing*: improve the mesh based on a smooth surface prior.

Fig. 8. The generic mesh evolution algorithm using TransforMesh.

4.7 Implementation Details

In our implementation, we used Computational Geometry Algorithms (CGALs) C++ library [44], which provides *guaranteed*, robust, and efficient implementations for various algorithms. We have used the following CGAL modules: N-dimensional fast box intersections, 2D constrained Delaunay triangulation, AABB trees, triangular meshes, and support for exact arithmetic kernels. As a preprocessing step, the triangle degeneracies are eliminated (see Section 5).

5 Mesh Evolution and Applications

In this section, a mesh evolution algorithm is introduced based on TransforMesh. Two applications using this framework are presented: mesh morphing and multiview 3D reconstruction, allowing us to test various configurations.

5.1 Mesh Evolution

A number of methods that deal with deformable surfaces exist in the literature, such as Kenneth Brakke's Evolver,[2] Wojtan and Turk's visco-elastic simulator [45], or the work of Celniker and Gossard on deformable surfaces [46]. Nevertheless, the aforementioned methods are all mesh-based topology preserving. This might or might not be 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 Fig. 8 and detailed below. More implementation details follow.

5.1.1 Algorithm

Within each evolution iteration, there are four steps. First, a velocity vector field $\vec{\mathcal{F}}$ is computed for each vertex of the mesh $\mathcal{M}$. This step is application specific. Second, the mesh is deformed using the computed velocity vector field $\vec{\mathcal{F}}$ and a small time step t. Third, 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 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 substeps are being performed: adaptive remeshing, vertex valence optimization, and smoothing. These four main steps are repeated until the mesh has reached the desired final state, also application specific.

5.1.2 Implementation Details

In practice, during the second step, 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 α 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. 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 sizes close to zero. In practice, this is obtained through edge swap, edge split, or edge collapse operations. Edge collapses are only performed if not violating the manifold constraint. Additionally, connected components where all edges are smaller than e_1 and that have a volume less than $\pi/6\, e_1^3$ are also removed. The vertex valence optimization step performs edge swaps in an attempt to ensure an overall vertex valence of 6 [47]. 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 degrees 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 [48], i.e., the discrete Laplace-Beltrami operator, Δv for each vertex v of the mesh. Furthermore, the mesh is smoothed using $v \rightarrow v - \beta \Delta v$. If smoothing will artificially shrink small components and remove surface details, then the Laplace-Beltrami operator could be used twice, as proposed in [49], by taking into account higher order surface information: $v \rightarrow v - \beta_1 \Delta v + \beta_2 \Delta \Delta v$. Alternatively, if no smoothing is necessary, β can be set to 0. These mesh optimization steps ensure that degenerate triangles, that is, triangles with zero area, are properly handled and eliminated. Degenerate triangles can affect the output accuracy of some of the geometric calculations, such as triangle normal estimation, triangle-triangle intersection tests, or Delaunay triangulations. Note, nevertheless, that TransforMesh uses local perturbations in order to eliminate any potential left-over boundary cases.

5.1.3 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

2. http://www.susqu.edu/facstaff/b/brakke/evolver/evolver.html.

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 reduce the number of intersections.

5.1.4 Remeshing

The remeshing step is important and should theoretically occur at each time step, due to the fact that some regions can become undersampled in areas where the speed vector field is divergent or oversampled 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.

We give below two mesh evolution examples, one for mesh morphing in Section 5.2, demonstrating the ability of the algorithm to handle complex surface evolutions, and the other one for multiview 3D reconstruction in Section 5.3. 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 Fig. 8.

5.2 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 toward a destination surface S_B. This will allow us to thoroughly test various cases of topology changes. Surface morphing has been widely described in the literature. We will adopt the method proposed by Breen and Whitaker [50]. We will summarize the reasoning that leads the surface evolution equation.

5.2.1 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 γ_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 γ_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 $\mathbf{N}(x)$:

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

The evolution strategy described above will converge to a local minimum. Given the source surface 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.

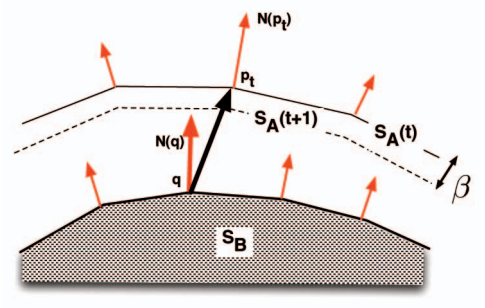


Fig. 9. Mesh Morphing evolution step. The surface S_A evolves from time t to time $t+1$ toward S_B. If for point $p \in S_A$, the closest point in S_B is q, then the point p will evolve along its normal with a magnitude of $(p-q) \cdot N(p)$, thresholded by a maximum user set evolution magnitude.

5.2.2 Complexity Issues and Mesh Discretization

In the general case, in order to calculate an exact distance function γ_B, one would have to consider the distance from a query point to 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 3D distance fields. For a recent survey, consult [51]. As per [51], 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 its 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 Fig. 9. If for a point $p \in S_A$, the closest point from S_B is q, the evolution equation for point p is

$$\gamma_B(p) = (q-p) \cdot N(p), \quad (2)$$

where γ 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 Fig. 8. The distance and sign from a

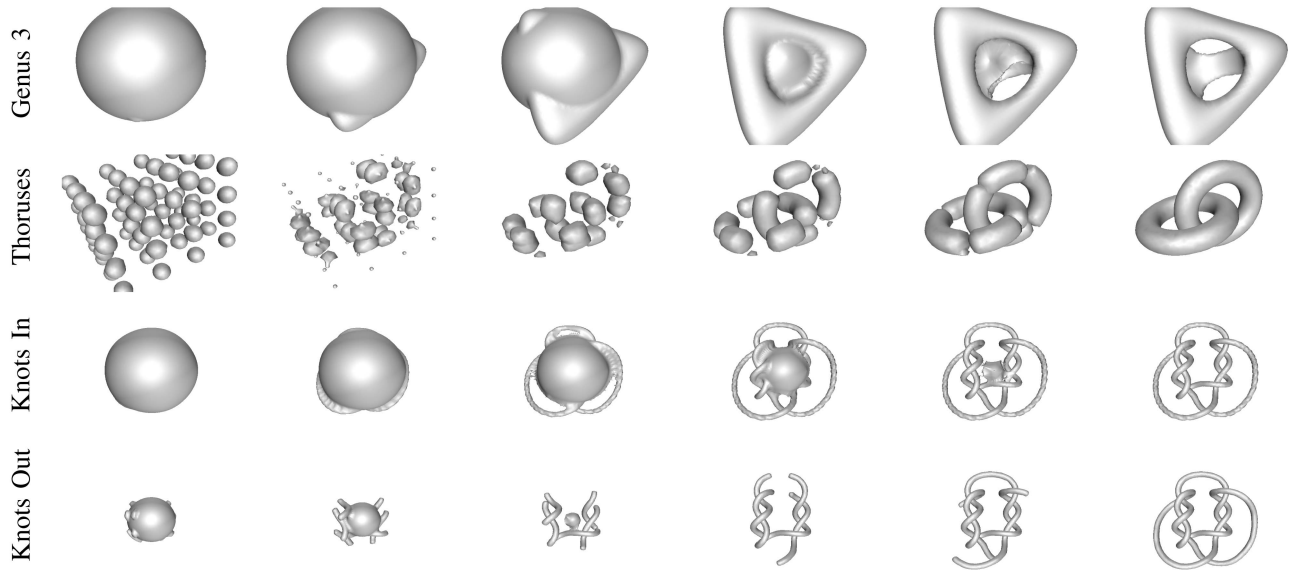


Fig. 10. Mesh morphing examples. Different steps for various test cases. Each row corresponds to a test case. The first column represents the first iteration, whereas the last column represents the last iteration.

query point are computed on the fly, as supposed to being stored in a distance field 3D 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 [52] 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 [51].

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 of 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 3D points. This would allow one to morph an initial mesh S_A toward a set of oriented 3D points P_B. If orientation information is not available, it can be estimated from neighboring points using principal component analysis [53]. Alternatively, in the context of multiview stereo, it can be obtained via a minimization scheme [54].

5.2.3 Results

In Fig. 10, we present results obtained with four test cases, entitled "Genus 3," "Thoruses," "Knots In," and "Knots Out." As 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 and 1.6 seconds, depending on the number of facets and 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 Fig. 8 within which we casted the current mesh morphing algorithm, we considered $t = 1$ for the time step, $\alpha = 0.2$ 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 $e_2 = 1.5 \cdot e_{avg}$.

TABLE 1
Mesh Morphing Statistics for Different Data Sets

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

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.

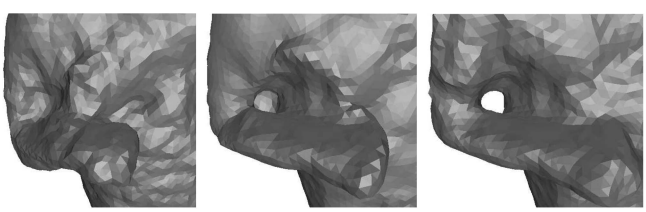


Fig. 11. Example of topological changes during mesh morphing in surface tracking. The source surface S'_t, shown to the left, is the result of a deformation of the original mesh S_t at time t such that it closely matches the mesh S_{t+1} at $t+1$, shown to the right. The mesh morphing process ensures the proper handling of topological changes (i.e., the whole formation in the arm region).

TABLE 2
Middleburry 3D Reconstruction Results

	Temple Ring		Temple Sparse Ring		Dino Ring		Dino Sparse Ring	
	Acc.	Compl.	Acc.	Compl.	Acc.	Compl.	Acc.	Compl.
Pons *et al.* [56]	0.60mm	99.5%	0.90mm	95.4%	0.55mm	99.0%	0.71mm	97.7%
Furukawa and Ponce [54]	0.47mm	99.6%	**0.63mm**	**99.3%**	**0.28mm**	**99.8%**	**0.37mm**	**99.2%**
Hernandez and Schmitt [14]	0.52mm	99.5%	0.75mm	95.3%	0.45mm	97.9%	0.60mm	98.52%
Vu et al.[57]	**0.45mm**	**99.8%**			0.53mm	99.7%		
TransforMesh	0.55mm	99.2%	0.78mm	95.8%	0.42mm	98.6%	0.45mm	**99.2%**

Accuracy: the distance d in millimeters that brings 90 percent of the result R within the ground truth surface G. Completeness: the percentage of G that lies within 1.25 mm of R.

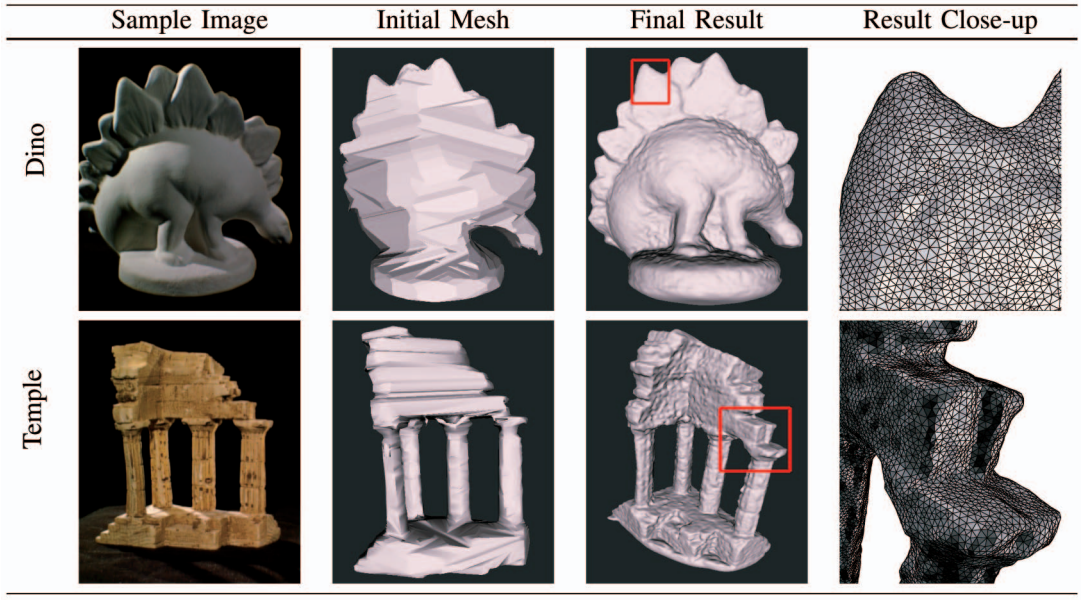


Fig. 12. Reconstruction results for the Middleburry multiview data set (dino case and temple case).

Additional results of mesh morphing are presented in Fig. 11, with meshes obtained from 3D reconstructions from multiple cameras, in the context nonrigid surface tracking [55].

5.3 Multiview 3D Reconstruction

In this section, we explain how our method fits into the multiview/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 [14], [33], [56], [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 algorithms, such as TransforMesh, provide a more flexible solution that allows to better resolve for local details using topological changes.

In [56], the multiview 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., Fig. 14. Surface evolution based on a level set formulation is proposed in [56]. Our contribution to this class of reconstruction methods is to extend such surface evolution approaches to meshes that allow us to focus on the shape's surface instead of a bounding volume.

The method described below was applied both to visual hulls, e.g., [58], and to sparse point-based 3D data, e.g., [59]. 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 [60], for example.

5.3.1 Methodology

The initial meshed surface corresponds to an extended bounding box obtained using image silhouettes and a geometric approach that involves cone intersections in 3D, i.e., [61]. 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 photoconsistent, i.e., its projections in the images should have similar photometric information [62].

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. [56]. 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 [56],

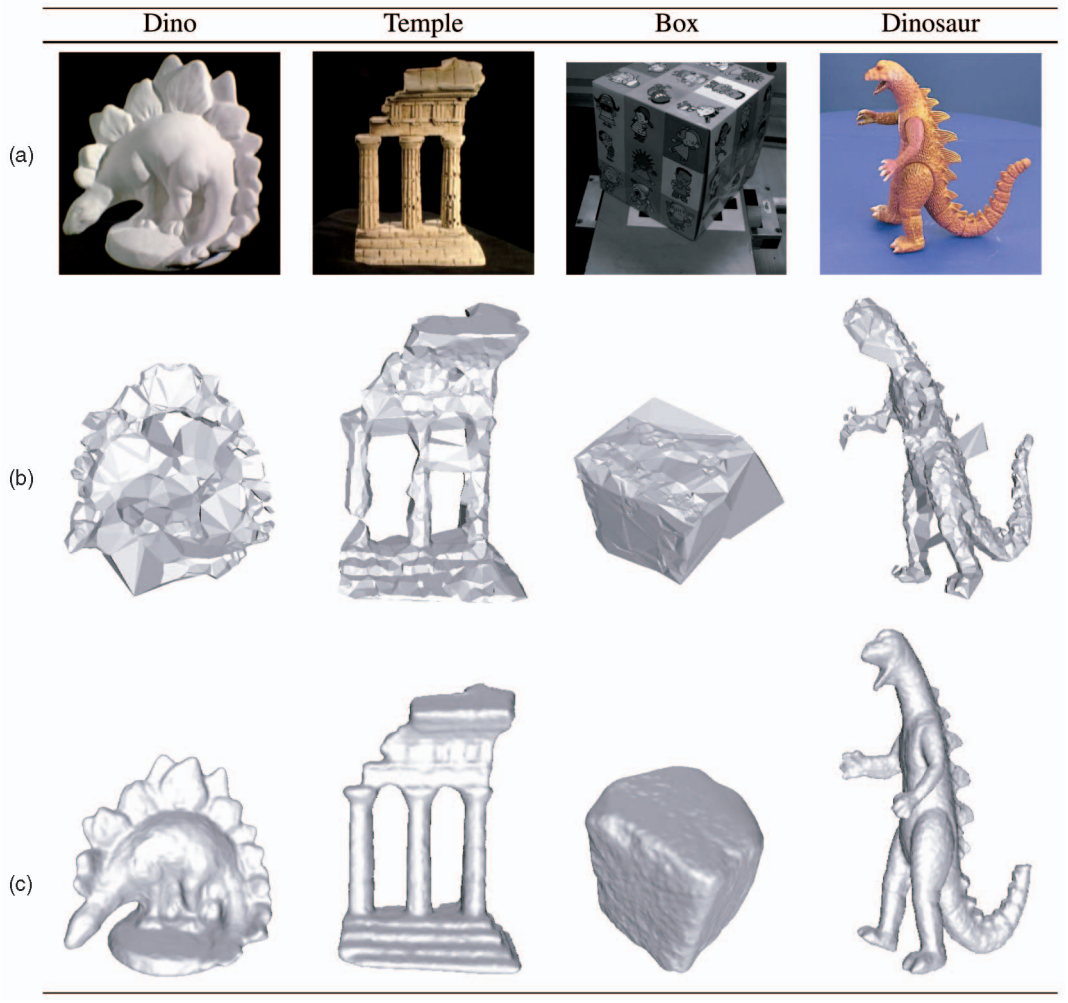


Fig. 13. Additional dense reconstruction results. (a) Sample input image. (b) A rough mesh obtained using PowerCrust [60] from the sparse 3D points, reconstructed using [59]. (c) The final dense reconstruction after surface evolution.

based on the normalized cross correlation. The evolution equation is, in this case:

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

In [56], 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 3D grid. As a side effect, all of the facets of the recovered surface are of a maximum size, set by the discretization grid cell. 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 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 Fig. 8 requires a number of parameters to be set in advance. In the case of 3D 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

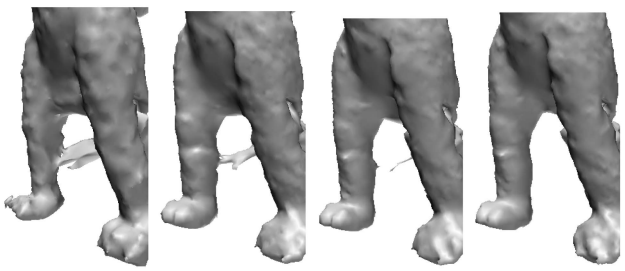


Fig. 14. Example of topological changes during 3D reconstruction for the dinosaur sequence, introduced in Fig. 13. The start-up surface, obtained from triangulated 3D points via PowerCrust [60], contains several topological errors (i.e., the extra branch connecting the dinosaur's limbs). They are corrected during the surface evolution, as shown in the rightmost image.

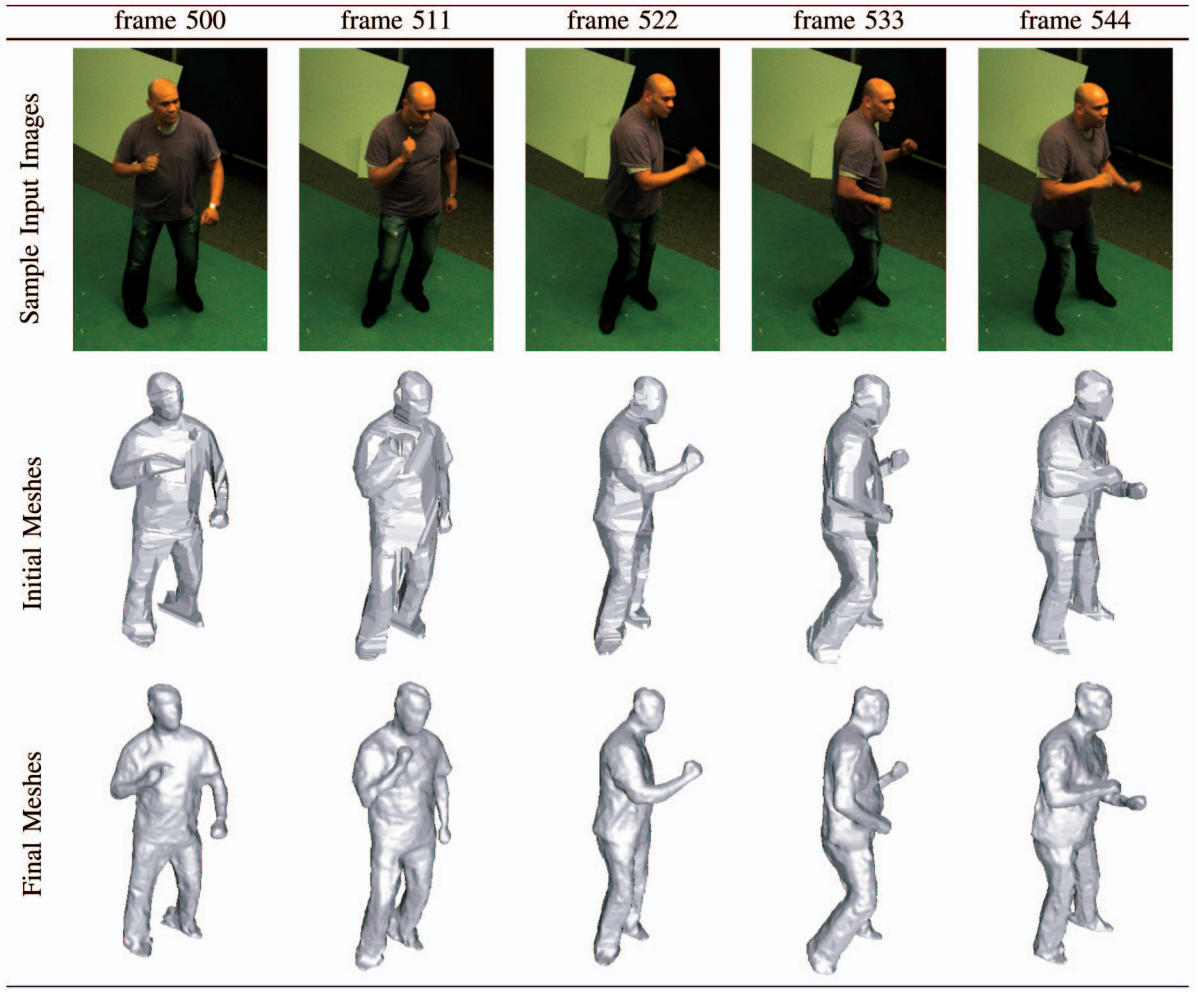


Fig. 15. Results for the Man dance sequence from the INRIA-PERCEPTION group. At each time step, the mesh is reconstructed from 34 cameras.

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.

5.3.2 Results

We have tested the mesh evolution algorithm with the data sets provided by the Multiview Stereo evaluation site [33].[3] The ground truth is obtained from laser scans. Comparative and detailed results are extracted from the Middlebury Web site and are presented in Table 2. The table includes results from Furukawa and Ponce [54], Pons et al. [56], Vu et al. [57], and Hernández and Schmitt [14]; all of these methods yield state-of-the-art results. The differences between all of these methods are very small, ranging between 0.01 and 0.1 mm. Some of our reconstruction results are shown in Figs. 12 and 13. While Vu et al. [57] used the same energy functional as part of their 3D reconstruction pipeline, their improved results are mostly due to the fact that the mesh regularization term takes photoconsistency into account.

Fig. 13 shows the results obtained with our method when starting with very rough meshes that correspond to coarse triangulations obtained from a sparse set of 3D points. An example of how TransforMesh handles topological changes is shown in Fig. 14. This figure shows a typical evolution scenario where there are more "topological problems" at the beginning. As the algorithm converges, self-intersections rarely occur.

Finally, Fig. 15 shows results obtained with the Man dance sequence publicly available from the Multiple-video database of the PERCEPTION group at INRIA.[4]

6 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. We provided both a detailed description of Transfor-

3. http://vision.middlebury.edu/mview/.

4. http://4drepository.inrialpes.fr/.

Mesh as well as an in-depth analysis of its convergence and performances (numerical stability and time complexity).

The TransforMesh algorithm was plugged into a generic mesh evolution framework, thus allowing us to address two challenging problems within a topology-adaptive approach: surface morphing and multiview image-based 3D reconstruction. Our main contribution with respect to the existing mesh evolution 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.

In the case of surface morphing, we showed that TransforMesh can deal with challenging topological cases.

The 3D 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 3D points. The 3D reconstruction results are of comparable quality with state-of-the-art methods recently developed by computer vision researchers.

ACKNOWLEDGMENTS

The authors thank Jean-Philippe Pons and Renaud Keriven for providing the source code for the gradient computation needed by multiview 3D reconstruction. This research was supported by the EC's Marie-Curie program through the VISIONTRAIN project.

REFERENCES

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

[2] S. Osher and J. Senthian, "Front Propagating with Curvature Dependent Speed: Algorithms Based on the Hamilton-Jacobi Formulation," *J. Computational Physics,* vol. 79, no. 1, pp. 12-49, 1988.

[3] W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics,* vol. 21, no. 4, pp. 163-169, July 1987.

[4] L.P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature Sensitive Surface Extraction from Volume Data," *Proc. ACM SIGGRAPH,* pp. 57-66, 2001.

[5] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual Contouring of Hermite Data," *Proc. ACM SIGGRAPH,* 2002.

[6] Y. Ohtake, A. Belyaev, and A. Pasko, "Dynamic Mesh Optimization for Polygonized Implicit Surfaces with Sharp Features," *The Visual Computer,* vol. 19, pp. 115-126, 2003.

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

[8] M. Sussman, P. Smereka, and S. Osher, "A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow," *J. Computational Physics,* vol. 114, no. 1, pp. 146-159, 1994.

[9] D. Enright, S. Marschner, and R. Fedkiw, "Animation and Rendering of Complex Water Surfaces," *Proc. ACM SIGGRAPH,* pp. 736-744, 2002.

[10] D. Adalsteinsson and J. Senthian, "A Fast Level Set Method for Propagating Interfaces," *J. Computational Physics,* vol. 118, no. 2, pp. 269-277, 1995.

[11] F. Losasso, R. Fedkiw, and S. Osher, "Spatially Adaptive Techniques for Level Set Methods and Incompressible Flow," *Computers and Fluids,* vol. 35, no. 10, pp. 995-1010, 2006.

[12] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, "A Hybrid Particle Level Set Method for Improved Interface Capturing," *J. Computational Physics,* vol. 183, no. 1, pp. 83-116, 2002.

[13] J.-J. Park, T. McInerney, D. Terzopoulos, and M.-H. Kim, "A Non-Self-Intersection Adaptive Deformable Surface for Complex Boundary Extraction from Volumetric Images," *Computer and Graphics,* vol. 25, pp. 421-440, 2001.

[14] C.E. Hernández and F. Schmitt, "Silhouette and Stereo Fusion for 3D Object Modeling," *Computer Vision and Image Understanding,* vol. 96, no. 3, pp. 367-392, 2004.

[15] T. McInerney and D. Terzopoulos, "T-Snakes: Topology Adaptive Snakes," *Medical Image Analysis,* vol. 4, no. 2, pp. 73-91, 2000.

[16] J.-O. Lachaud and B. Taton, "Deformable Model with Adaptive Mesh and Automated Topology Changes," *Proc. Fourth Int'l Conf. 3D Digital Imaging and Modeling,* 2003.

[17] Y. Duan, L. Yang, H. Qin, and D. Samara, "Shape Reconstruction from 3D and 2D Data Using PDE-Based Deformable Surfaces," *Proc. European Conf. Computer Vision,* vol. 3, pp. 238-251, 2004.

[18] T. Brochu and R. Bridson, "Robust Topological Operations for Dynamic Explicit Surfaces," *SIAM J. Scientific Computing,* vol. 31, no. 4, pp. 2472-2493, 2009.

[19] J.-P. Pons and J.-D. Boissonnat, "Delaunay Deformable Models: Topology Adaptive Meshes Based on the Restricted Delaunay Triangulation," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* June 2007.

[20] M. Aftosmis, M. Berger, and J. Melton, "Robust, Efficient Cartesian Mesh Generation for Component-Based Geometry," AIAA Paper 97-0196, citeseer.ist.psu.edu/500259.html, 1997.

[21] W. Jung, H. Shin, and B.K. Choi, "Self-Intersection Removal in Triangular Mesh Offsetting," *Computer-Aided Design and Applications,* vol. 1, nos. 1-4, pp. 477-484, 2004.

[22] C. Wojtan, N. Thurey, M. Gross, and G. Turk, "Deforming Meshes That Split and Merge," *Proc. ACM SIGGRAPH,* 2009.

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

[24] P.M. Hubbard, "Constructive Solid Geometry for Triangulated Polyhedra," Technical Report CS-90-07, Dept. of Computer Science, Brown Univ., citeseer.ist.psu.edu/hubbard90constructive.html, Jan. 1990.

[25] B.G. Baumgart, "Geometric Modeling for Computer Vision," PhD dissertation, Stanford Univ., 1974.

[26] I.C. Braid, R.C. Hillyard, and I.A. Stroud, "Stepwise Construction of Polyhedra in Geometric Modelling," *Math. Methods in Computer Graphics and Design,* Academic Press, 1978.

[27] A. Agrawal and A. Requicha, "A Paradigm for the Robust Design of Algorithms for Geometric Modeling," *Computer Graphics Forum,* vol. 13, no. 3, pp. 33-44, 1994.

[28] J. Rossignac and A. Requicha, "Solid Modeling," *Encyclopedia of Electrical and Electronics Eng.,* John Wiley and Sons, 1999.

[29] A. Rappoport and S. Spitz, "Interactive Boolean Operations for Conceptual Design of 3D Solids," *Proc. ACM SIGGRAPH,* pp. 269-278, 1997.

[30] J. Goldfeather, J.P.M. Hultquist, and H. Fuchs, "Fast Constructive-Solid Geometry Display in the Pixel-Powers Graphics System," *Proc. ACM SIGGRAPH,* pp. 107-116, July 1986.

[31] H. Biermann, D. Kristjansson, and D. Zorin, "Approximate Boolean Operations on Free-Form Solids," *Proc. ACM SIGGRAPH,* pp. 185-194, 2001.

[32] A.L.N. Litke and P. Schröder, "Trimming for Subdivision Surfaces," technical report, Caltech, 2000.

[33] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 519-526, http://vision.middlebury.edu/mview/, 2006.

[34] P. Carvalho and P. Cavalcanti, "Point in Polyhedron Testing Using Spherical Polygons," *Graphics Gem V,* ch. II, pp. 42-49, Academic Press, 1995.

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

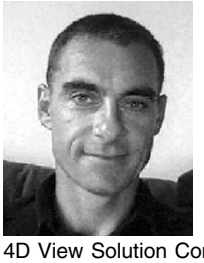
[36] L. Kettner, A. Meyer, and A. Zomorodian, "Intersecting Sequences of dD Iso-Oriented Boxes," *CGAL-3.2 User and Reference Manual,* C.E. Board, ed., http://www.cgal.org/Manual/3.2/doc_html/cgal_manual/packages.html#Pkg:BoxIntersectionD, 2006.

[37] A. Zomorodian and H. Edelsbrunner, "Fast Software for Box Intersection," *Int'l J. Computational Geometry and Applications,* vol. 12, nos. 1/2, pp. 143-172, 2002.

[38] P. Alliez, S. Tayeb, and C. Wormser, "AABB Tree," *CGAL User and Reference Manual,* third ed., CGAL Editorial Board, ed., http://www.cgal.org/Manual/3.5/doc_html/cgal_manual/packages.html#Pkg:AABB_tree, 2009.

[39] S. Hert and M. Seel, "dD Convex Hulls and Delaunay Triangulations," *CGAL-3.2 User and Reference Manual*, C.E. Board, ed., http://www.cgal.org/Manual/3.2/doc_html/cgal_manual/packages.html#Pkg:ConvexHullD, 2006.
[40] A. Gueziec, G. Taubin, F. Lazarus, and B. Horn, "Cutting and Stitching: Converting Sets of Polygons to Manifold Surfaces," *IEEE Trans. Visualization and Computer Graphics*, vol. 7, no. 2, pp. 136-151, Apr.-June 2001.
[41] H. Shin, J.C. Park, B.K. Choi, Y.C. Chung, and S. Rhee, "Efficient Topology Construction from Triangle Soup," *Proc. Geometric Modeling and Processing*, 2004.
[42] H. Edelsbrunner and E. Mücke, "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms," *ACM Trans. Graphics*, vol. 9, no. 1, pp. 66-104, 1990.
[43] M. Mäntylä, "Boolean Operations of 2-Manifolds through Vertex Neighborhood Classification," *ACM Trans. Graphics*, vol. 5, no. 1, pp. 1-29, 1986.
[44] C.E. Board, *CGAL-3.2 User and Reference Manual*, http://www.cgal.org/Manual/3.2/doc_html/cgal_manual/index.html, 2006.
[45] C. Wojtan and G. Turk, "Fast Viscoelastic Behavior with Thin Features," *Proc. ACM SIGGRAPH*, 2008.
[46] G. Celniker and D. Gossard, "Deformable Curve and Surface Finite-Elements for Free-Form Shape Design," *Computer Graphics*, vol. 25, pp. 257-266, 1991.
[47] L. Kobbelt, T. Bareuther, and H.-P. Seidel, "Multiresolution Shape Deformations for Meshes with Dynamic Vertex Connectivity," *Proc. Eurographics*, pp. 249-260, 2000.
[48] M. Meyer, M. Desbrun, P. Schröder, and A.H. Barr, "Discrete Differential Geometry Operators for Triangulated 2-Dimensional Manifolds," *Proc. Workshop Visualization and Math.*, 2002.
[49] H. Delingette, M. Herbert, and K. Ikeuchi, "Shape Representation and Image Segmentation using Deformable Surfaces," *Image and Vision Computing*, vol. 10, pp. 132-145, 1992.
[50] D.E. Breen and R.T. Whitaker, "A Level-Set Approach for the Metamorphosis of Solid Models," *IEEE Trans. Visualization and Computer Graphics*, vol. 7, no. 2, pp. 173-192, Apr.-June 2001.
[51] M.W. Jones, J.A. Bærentzen, and M. Sramek, "3D Distance Fields: A Survey of Techniques and Applications," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 4, pp. 581-599, July/Aug. 2006.
[52] G.R. Hjaltason and H. Samet, "Ranking in Spatial Databases," *Proc. Symp. Large Spatial Databases*, pp. 83-95, 1995.
[53] H. Hoppe, T. DeRose, T. Duchamp, J. MCDonald, and W. Stuetzle, "Surface Reconstruction from Unorganized Points," *Proc. ACM SIGGRAPH*, 1992.
[54] Y. Furukawa and J. Ponce, "Accurate, Dense, and Robust Multi-View Stereopsis," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1362-1376, Aug. 2010.
[55] K. Varanasi, A. Zaharescu, E. Boyer, and R.P. Horaud, "Temporal Surface Tracking Using Mesh Evolution," *Proc. European Conf. Computer Vision*, 2008.
[56] J.-P. Pons, R. Keriven, and O. Faugeras, "Multi-View Stereo Reconstruction and Scene Flow Estimation with a Global Image-Based Matching Score," *Int'l J. Computer Vision*, vol. 72, no. 2, pp. 179-193, 2007.
[57] H. Vu, R. Keriven, P. Labatut, and J.-P. Pons, "Towards High-Resolution Large-Scale Multi-View Stereo," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 2009.
[58] J.-S. Franco and E. Boyer, "Exact Polyhedral Visual Hulls," *Proc. British Machine Vision Conf.*, vol. 1, pp. 329-338, Sept. 2003.
[59] A. Zaharescu and R.P. Horaud, "Robust Factorization Methods Using a Gaussian/Uniform Mixture Model," *Int'l J. Computer Vision*, vol. 81, no. 3, pp. 240-258 , Mar. 2009.
[60] N. Amenta, S. Choi, and R. Kolluri, "The Power Crust, Unions of Balls, and the Medial Axis Transform," *Computational Geometry: Theory and Applications*, vol. 19, nos. 2/3, pp. 127-153, 2001.
[61] J.S. Franco and E. Boyer, "Efficient Polyhedral Modeling from Silhouettes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 414-427, Mar. 2009.
[62] S.M. Seitz and C.R. Dyer, "Photorealistic Scene Reconstruction by Voxel Coloring," *Int'l J. Computer Vision*, vol. 35, no. 2, pp. 151-173, 1999.

Andrei Zaharescu received the BSc and MSc degrees in computer science from York University, Toronto, Canada, in 2002 and 2004, respectively, and the PhD degree in computer science from the Institut National Polytechnique de Grenoble, France, in 2008. He is currently working in the industry in the area of computer vision, dealing with background subtraction and 2D tracking methods. His research interests include camera calibration, background subtraction, sparse and dense 3D reconstruction, geometric mesh processing, and 2D and 3D tracking. He is a member of the IEEE and the IEEE Computer Society.



Edmond Boyer received the PhD degree from the Institut National Polytechnique de Lorraine, France, in 1996. He is an associate professor at Grenoble Universities, France. He started his professional career as a research assistant in the Department of Engineering, University of Cambridge, United Kingdom. He joined INRIA Grenoble in 1998. His fields of competence cover computer vision, computational geometry, and virtual reality. He is a cofounder of the 4D View Solution Company in the domain of spatiotemporal modeling. His current research interests are in 3D dynamic modeling from images and videos, motion capture and recognition from videos, and immersive and interactive environments.

Radu Horaud received the BSc degree in electrical engineering, the MSc degree in control engineering, and the PhD degree in computer science from the Institut National Polytechnique de Grenoble, France. He holds the position of director of research with the Institut National de Recherche en Informatique et Automatique (INRIA), Grenoble Rhône-Alpes, Montbonnot, France, where he has been the head of the PERCEPTION team since 2006. His research interests include computer vision, machine learning, multisensory fusion, and robotics. He is an area editor of the *Elsevier Computer Vision and Image Understanding*, a member of the advisory board of the *Sage International Journal of Robotics Research*, and a member of the editorial board of the *Kluwer International Journal of Computer Vision*. He was a program cochair of the Eighth IEEE International Conference on Computer Vision (ICCV '01).