TransforMesh : A Topology-Adaptive Mesh-Based Approach to Surface Evolution

Andrei Zaharescu[1], Edmond Boyer[1] and Radu Horaud[1]

INRIA Rhone-Alpes, 655 ave de l'Europe, Montbonnot, 38330, FRANCE,
first-name.last-name@inrialpes.fr

Abstract. Most of the algorithms dealing with image based 3-D reconstruction involve the evolution of a surface based on a minimization criterion. The mesh parametrization, while allowing for an accurate surface representation, suffers from the inherent problems of not being able to reliably deal with self-intersections and topology changes. As a consequence, an important number of methods choose implicit representations of surfaces, e.g. level set methods, that naturally handle topology changes and intersections. Nevertheless, these methods rely on space discretizations, which introduce an unwanted precision-complexity trade-off. In this paper we explore a new mesh-based solution that robustly handles topology changes and removes self intersections, therefore overcoming the traditional limitations of this type of approaches. To demonstrate its efficiency, we present results on 3-D surface reconstruction from multiple images and compare them with state-of-the art results.

1 Introduction

A vast number of problems in the area of image-based 3-D modeling are casted as energy minimization problems where 3-D shapes are optimized such that they best explain image information. More specifically, when interested in performing 3-D reconstruction from multiple images, one typically attempts to recover 3-D shapes by evolving a surface with respect to various criteria, such as photometric or geometric consistencies. Meshes, either triangular, polygonal or even tetrahedral, are one of the most used forms of shape representation. Traditionally, an initial mesh is obtained using some well known method, e.g. bounding boxes or visual hulls, and then deformed over time such that it minimizes an energy, based typically on some form of image similarity measure. There exist two main schools of thought on how to cast the above mentioned problem.

Lagrangian methods propose an intuitive approach where the surface representation, i.e. the mesh, is directly deformed over time. Meshes present numerous advantages, among which adaptive resolution and compact representation, but raise two major issues of concern when evolved over time: self-intersections and topology changes. It is critical to deal with both issues when evolving a mesh over time. In order to answer to this, McInerney and Terzopoulos [1] proposed 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. Lauchaud et al. [2] proposed a heuristic method where merges and splits are perform 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. In addition, Pons et al. [3] recently proposed a method based on a restricted 3-D Delaunay triangulation. Two instances of the mesh are needed: one at time t - a proper mesh (without self intersections and topological issues) used to mark each tetrahedron as belonging to a particular material (inside/outside), and another mesh at another time instance t+1, which "treated" based on the the previous cell categorization. While being a very elegant solution, it nevertheless relies on the assumption that the mesh is sufficiently dense, such that the Delaunay triangulation will not considerably change its layout.

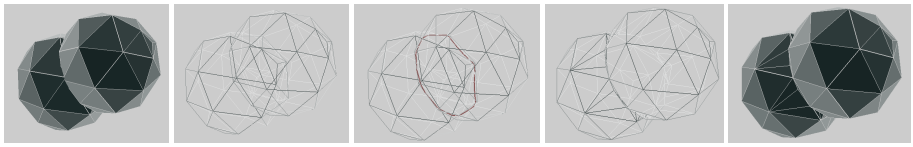
Eulerian methods formulate the problem as time variation over sampled space, most typically fixed grids. In such a formulation, the mesh, also called the interface, is implicitly represented. One of the most successful methods, the *level set method* [4] [5], represents the interface as the zero set of a function. Such an embedding within an implicit function allows one to automatically handle mesh topology changes, i.e. merges, splits. In addition, such methods allow for an easy computation of geometric properties. Nevertheless, this method does not come without its drawbacks. The amount of space required to store the implicit representation is much larger than when storing a mesh parametrization. In addition, at each step of the evolution, the mesh has be recovered from the implicit function. This operation is limited by the grid resolution. Consequently, a mesh with variable resolution cannot be properly preserved by the embedding function.

In light of this, we propose a mesh-based *lagrangian* approach. To solve for the self-intersections and topology changes issues, we adopt the algorithm proposed by Jung *et al.* [6] in the context of mesh-offseting and extend it to more general situations, as faced with when modeling from multiple images. We have successfully applied our approach to surface reconstruction using multiple cameras. The 3-D reconstruction literature is vast. Recently, Furukawa *et al.* [7] provide some of the most impressive results. They use however a mesh based solution [2] mentioned above that imposes equal face sizes. Pons et al. [8] provide an elegant level-set based implementation. Hernandez et al. [9] provide an in-between solution: they choose an explicit mesh representation, however they immerse the mesh within a vector field with forces proportional to the image consistencies at given places within a grid. Self-intersections and topology changes are not handled. Other approaches are also surveyed in the recent overview on multi-view stereo reconstruction algorithms [10]. Our contribution with respect to these methods is to provide a purely mesh based solution that does not constrain meshes and allows faces of all sizes as well as topology changes, with the goal of increasing precision without sacrificing complexity when dealing with surface evolution problems.

The rest of the paper is organized as follows. Section 2 introduces the algorithm that handles self-intersection and topology changes. In section 3, we present its application to the 3-D surface reconstruction problem. Section 4 shows results on well known data sets and makes comparisons with state-of-the-art approaches, before concluding in section 5.

2 TransforMesh - A Topology-Adaptive Self-Intersection Removal Mesh Solution

As stated earlier, the main limitations that prevent many applications from using meshes are self-intersections and topology changes, which frequently occur when evolving meshes. In this paper, we show that such limitations can be overcome using a very intuitive geometrically-driven solution. In essence, the approach preserves the mesh consistency by detecting self-intersections and considering the subset of the original mesh surface that is *outside* with respect to the mesh orientation. A toy example is illustrated in Figure 1. Our method is inspired by the work of [6] proposed in the context of mesh-offsetting or mesh expansions. We extend it to the general situation of identifying a valid mesh, i.e. a manifold, from a self-intersecting one with possible topological issues. The currently proposed algorithm has the great advantage of dealing with topological changes *naturally*, much in the same fashion as the Level-Set based solutions, casting it as a viable solution to surface evolutions with meshes. The only requirement is that the input mesh is the result of the deformation of an oriented and valid mesh. The following sections detail the sequential steps of the algorithm.



(a) Input (b) Input (c) Intersections (d) Output (e) Output

Fig. 1. A toy example of TransforMesh at work.

2.1 Pre-treatment

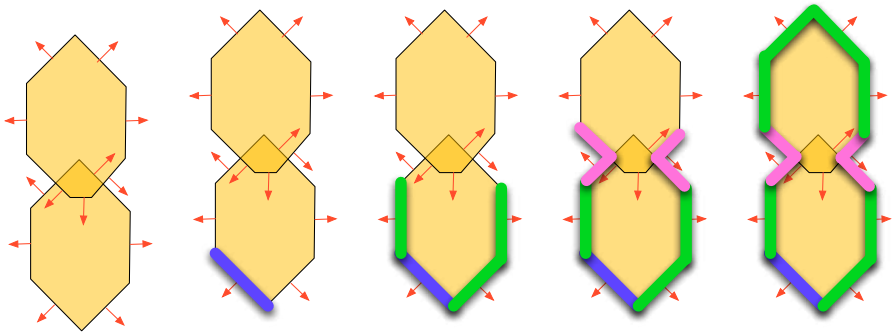
Most mesh-related algorithms suffer from numerical problems due to degenerate triangles. Such degenerate triangles are mainly triangles having area close to zero. In order to remove those triangles, two operations are performed: edge collapse and edge flip.

2.2 Self-intersections

The first step of the algorithm consists of identifying self-intersections, i.e. edges along which triangles of the mesh intersect. In practice, one would have to perform $n^2/2$ checks to see if two triangles intersect, which can become quite 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 [11] and described in [12]. The complexity of the method is $O(n\,log^d(n)+k)$ for the running time and $O(n)$ for the space occupied, where n is the number of triangles, d the dimension (3 in the 3-D case), and k the output complexity, i.e., the number of pairwise intersections of the triangles.

2.3 Valid region growing

The second step of the algorithm consists in identifying *valid* triangles in the mesh. To this purpose, a valid region growing approach is used to propagate validity labels on triangles that composed the *outside* of the mesh.



(a) Init (b) Seed-triangle (c) Valid Queue (d) Partial Queue (e) End

Fig. 2. Valid Region Growing (2-D simplified view). The selected elements are in bold.

Initialization. Initially, all the triangles are marked as non-visited. This corresponds to Figure 2(a). Three queues are maintained: one named $\mathcal{V}$, of valid triangles, i.e. triangles outside and without intersections; one named $\mathcal{P}$ of partially valid triangles, i.e. only part of the triangle is outside, and finally one named $\mathcal{G}$, where all the valid triangles will be stored until stitched together into a new mesh (in section 2.4).

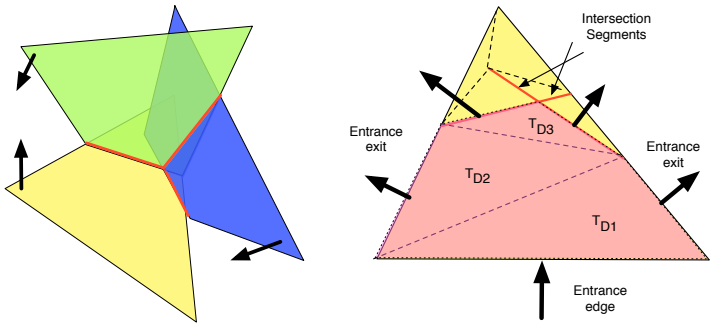
All that follows is performed within an outer loop, while there still exists a valid seed triangle.

Seed-triangle Finding. A seed-triangle is defined as a non-visited triangle without intersections whose normal does not intersect any other triangle of the same connected component. This corresponds to Figure 2(b). In other words, a seed-triangle is a triangle that is guaranteed to be on the exterior. This triangle is crucial, since we will start our valid region growing from such a triangle. If found, the triangle will be added to $\mathcal{V}$ and marked as valid; otherwise, the algorithm will jump to the next stage (section 2.4).

The next two-steps are performed within a inner loop until both $\mathcal{V}$ and $\mathcal{P}$ are empty.

Valid Queue Processing. While $\mathcal{V}$ is not empty, pop a triangle t from the queue, add it to $\mathcal{G}$ and for each neighbouring triangle $N(t)$ perform the following: if $N(t)$ is non-visited and has no intersections, then add it to $\mathcal{V}$; if $N(t)$ is non-visited and has intersections, then add it to $\mathcal{P}$ together with the entrance segment and direction, corresponding in this case to the oriented half-edge. (see Figure 2(c)).

Partially-Valid Queue Processing. While $\mathcal{P}$ is not empty, pop a triangle t from the queue, together with the entrance half-edge f_t. Also, 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 $H_t = \{h_{tj} | \text{where } j = 1..3\}$ represent the triangle half-edges. A constrained 2-D triangulation is being performed in the triangle plane, using [13], to ensure that all segments in both S_t and H_t appear in the new 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 3. A crucial aspect to ensure a natural handling of topological changes is on choosing the correct side of continuation of the "fill" like region growing when crossing a partially valid triangle. The correct orientation is chosen such that, if the original normals are maintained, the two newly formed sub-triangles would preserve the watertightness constraint of a manifold. This condition can also be casted as following: 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 4.



(a) Triangle Intersections (b) Partial Triangle Traversal

Fig. 3. Partial Triangle Traversal.

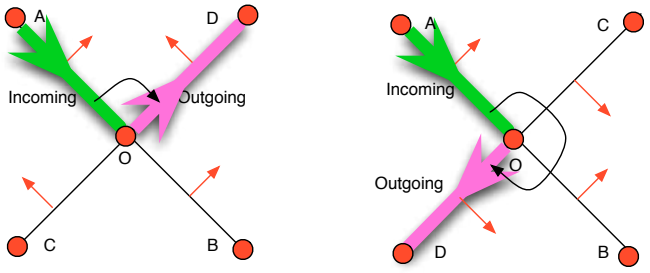


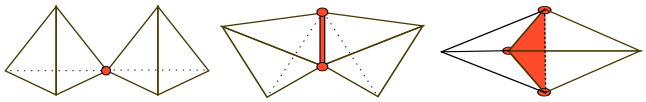
Fig. 4. Partial Triangle Crossing Cases (side view).

2.4 Triangle Stitching

The region growing algorithm described previously will iterate until all the triangles have been selected. In the chosen demo example, this corresponds to Figure 2(e). At this stage, what remains to do 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 [14, 15]. In most cases this is a straight forward operation, reduced to identifying the common vertices and edges, followed by stitching. However, there are three special cases, in which performing a simple stitching will violate the mesh constraints and produce locally non-manifold structures. The special cases, shown in Figure 5, arise from performing stitching in places where the original structure should have been maintained. We adopt the naming convention from [14], calling them the singular vertex case, the singular edge case and the singular face case. All cases are easy to identify only from local operations and are identified after all the stitching has been performed.

In the singular vertex case, in order to detect whether vertex v is singular, we adopt the following algorithm: mark all the facets incident to the vertex v as non-visited. Then start from a facet of v, mark it visited and do the same with its non-visited neighbours that are also incident to v (neighbour as chosen based on half-edges). The process is repeated until all the neighbouring facets are processed. If by doing so we exhausted all the neighbouring 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. In order to detect a singular edges e, all we have to do is count the number of triangles that share that edge. If it is bigger than 2, we have a singular edge case and two additional vertices and a new edge will be added to account for it.

In practice, only the singular vertex case appears in real triangular mesh test cases. Such an example has been created to illustrate such a scenario and it is shown in Figure 6.



(a) Singular Vertex (b) Singular Edge (c) Singular Face

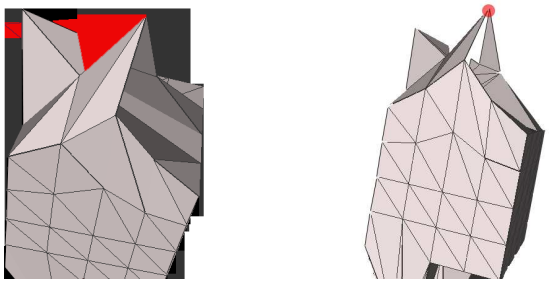
Fig. 5. Special case encountered while stitching a triangle soup.

2.5 Topological Changes

The partial-triangle crossing technique described earlier ensures a *natural* handling of the topological changes (splits and joins) that plagued the mesh approaches until now. Representative cases are illustrated in Figure 7.

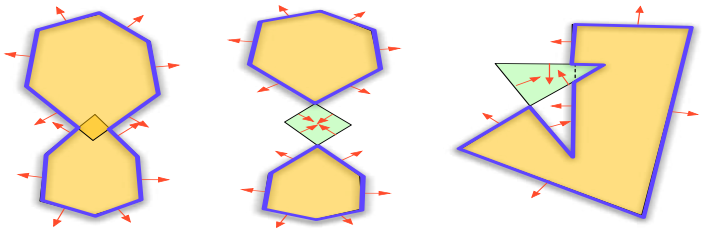
3 Using TransforMesh to perform mesh evolutions.

Our original motivation in developing a mesh self-intersection removal algorithm was to perform mesh evolutions, in particular when recovering 3-D shapes from multiple



(a) Original Mesh. (b) Resulting mesh.

Fig. 6. An example of how a singular vertex occurs in a typical self-intersection removal situation, due to an inverted triangle (marked in red).



(a) Join Case (b) Split Case (c) Inside Carving Case

Fig. 7. Different topological changes examples (2-D simplified view). The outline of the final surface obtained after self-intersection removal is outlined in bold blue.

calibrated images. As stated earlier, few efforts have been put in mesh-based solutions for such 3-D surface reconstruction problem, mostly due to the topological issues raised by mesh evolutions. However, meshes allow one to focus on the region of interest in space, namely the shape's surface and, as a result, lower the complexities and lead to better precisions with respect to volumetric approaches. In this section we present the application of TransforMesh to the surface reconstruction problem. Often such a problem is casted as an energy minimization over a surface. We decided to start from exact visual hull reconstructions, obtained using [16] and further improve the mesh using photometric constraints by means of an energy functional described in [8].

The photometric constraints are casted as an energy minimization problem using similarity measure between pairs of cameras that are close to each other. We denote by $S \subset \mathbb{R}^3$ the 3-D surface. Let $I_i : \Omega_\mathrm{i} \subset \mathbb{R}^2 \rightarrow \mathbb{R}^d$ be the image captured by camera i (d=1 for grayscale and d=3 for color images). The perspective projection of camera i is represented as $\Pi_\mathrm{i} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$. Since the method uses visibility, consider S_i as part of surface S visible in image i. In addition, the back-projection of image from camera i onto the surface is represented as $\Pi_\mathrm{i}^{-1} : \Pi_\mathrm{i}(\mathrm{S}) \rightarrow \mathrm{S_i}$.

Armed with the above notation, one can compute a similarity measure M_{ij} of the surface S as the the similarity measure between image I_i and the reprojection of image I_j into the other camera i via the surface S. Summing across all the candidate stereo pairs, one can write:

$$\mathcal{M}(S) = \sum_i \sum_{j \neq i} \mathcal{M}_{ij}(S) \tag{1}$$

$$\mathcal{M}_{ij}(S) = M|_{\Omega_i \cap \Pi_i(S_j)} \left(I_i, I_i \circ \Pi_i \circ \Pi_{j,S}^{-1} \right) \tag{2}$$

Finally, the surface evolution equation at a point x is given by:

$$\frac{\partial S}{\partial t} = \left[-\lambda_1 E_{smooth} + E_{img} \right] \mathbf{N} \tag{3}$$

where E_{smooth} depends on the curvature H (see [7]), $\mathbf{N}$ represents the surface normal and E_{img} is a photoconsistency term that is a summation across pairs of cameras which depends upon derivatives of the similarity measure $\mathcal{M}$, of the images I, of the projection matrices Π and on the distance x_z (see [8] for more details).

In the original paper [8], the surface evolution equation was implemented within the Level-Set framework. We extend it to meshes using the TransforMesh algorithm described in the previous section. The original solution performs surface evolution using a coarse to fine approach in order to escape 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 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 visual hulls, for which 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 a coarse to fine approach in which bigger triangles are moved until they stabilize, followed by dimension reduction via an edge-splits. The algorithm iterates at a smaller scale until the desired smallest edge size is obtained. Therefore, the algorithm uses a multi-scale approach, starting from scale s_{max} to $s_{min} = 1$ in $\lambda_2 = \sqrt{2}$ increments using $\Delta t = 0.001$ as the timestep. A vertex can maximally move 10% of the average incoming half-edges. The original surface, obtained from a visual hull, is evolved using equation (3), where the cross correlation was used as a similarity measure and $\lambda_1 = 0.3$. Every 5 iterations TransforMesh is performed, in order to remove the self-intersections and allow for any topological changes.

4 Results

We have tested the mesh evolution algorithm with the datasets provided by the Multi-View Stereo Evalutation site [10] (http://vision.middlebury.edu/mview/) and our results are comparable with state-of-the-art: we rank in the top 1-3 (depending on the data set and ranking criteria chosen) and the results are within sub-milimeter accuracy. Detailed results are extracted from the website and presented in Table 1 (consult the website for detailed info and running times). We have also included results by Furukawa et al. [7], Pons et al. [8] and Hernandez et al. [9], considered to be the state of the art. The

differences between all methods are very small, ranging between $0.01mm$ to $0.3mm$. Some of our reconstruction results are shown in Figure 8.

Dataset / Paper	Temple Ring		Temple Sparse Ring		Dino Ring		Dino Sparse Ring	
	Acc.	Compl.	Acc.	Compl.	Acc.	Compl.	Acc.	Compl.
Pons et al. [8]	0.60mm	99.5%	0.90mm	95.4%	0.55mm	99.0%	0.71mm	97.7%
Furukawa et al. [7]	0.55mm	99.1%	0.62mm	99.2%	0.33mm	99.6%	0.42mm	99.2%
Hernandez et al. [9]	0.52mm	99.5%	0.75mm	95.3%	0.45mm	97.9%	0.60mm	98.52%
Our results	0.55mm	99.2%	0.78mm	95.8%	0.42mm	98.6%	0.45mm	99.2%

Table 1. 3-D Rec. Results. Accuracy: 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.

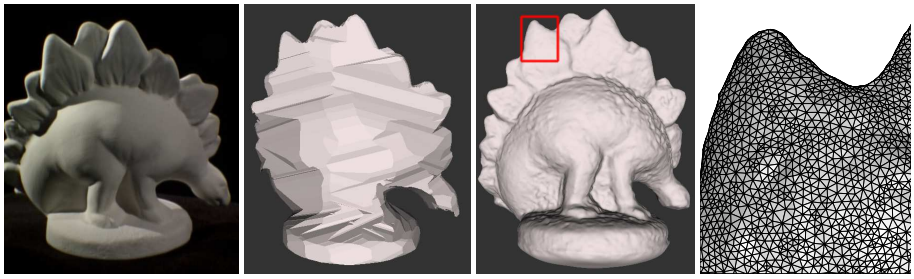
The algorithm reaches a good solution without the presence of a silhouette term in the evolution equation. In a typical evolution scenario, there are a more self-intersections at the beginning, but, as the algorithm converges, intersections rarely occur. Additionally, in the temple case, we performed a test where we have started from one big sphere as the startup condition, in order to check whether the topological split operation performs properly. Proper converges was obtained. We acknowledge that TransforMesh was not put to a thorough test using the current data sets, which might leave the reader suspicious about special cases in which the method could fail. We have implemented a mesh morphing algorithm in order to test the robustness of the method. We have successfully morphed meshes with significantly different topology from the surface of departure. Results will be detailed in another publication.

Implementation Notes. In our implementation we have made extensive use of CGAL (Computational Geometry Algorithms Library) [17] C++ library, which provides *excellent* implementations for various algorithms, among which the n-dimentional fast box intersections, 2-D constrained Delaunay triangulation, triangular meshes and support for exact arithmetic kernels. The running times of TransforMesh depend greatly on the number of self-intersections, since more than 80% of the running time is spent performing them. Typically, the running time for performing the self-intersections test is under 1 second for a mesh with $50,000$ facets, where exact arithmetic is used for triangle intersections and the self-intersections are in the range of 100.

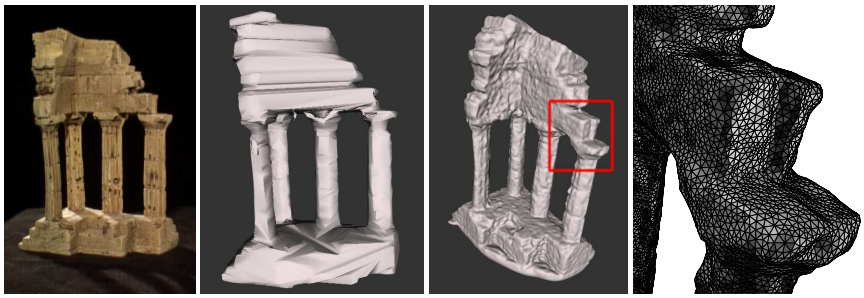
5 Conclusion

We have presented a fully geometric efficient *Lagrangian* solution for triangular mesh evolutions able to handle topology changes *gracefully*. We have tested our method in the context of multi-view stereo 3-D reconstruction and we have obtained top ranking results, comparable with state-of-the-art methods in the literature. Our contribution with respect to the existing methods is to provide a purely geometric mesh-based solution that does not constrain meshes and that allows for facets of all sizes as well as for topology changes.

Acknowledgements This research was supported by the VISIONTRAIN RTN-CT-2004-005439 Marie Curie Action within the European Community's Sixth Framework Programme. This paper reflects only the author's views and the Community is not liable for



(a) Dino - Input (b) Dino - Start (c) Dino - Results (d) Dino - Closeup



(e) Temple - Input (f) Temple - Start (g) Temple-Results (h) Temple-Closeup

Fig. 8. Reconstruction results obtained in the temple and in the dino case (the gradient code is a courtesy of Jean-Phillippe Pons).

any use that may be made of the information contained therein. We would also like to thanks Jean-Phillippe Pons *et al.* for allowing us to re-use the calculation of the gradient code.

References

1. McInerney, T., Terzopoulos, D.: T-snakes: Topology adaptive snakes. Medical Image Analysis **4**(2) (2000) 73–91
2. Lachaud, J.O., Taton, B.: Deformable model with adaptive mesh and automated topology changes. In: Proceedings of the Fourth International Conference on 3-D Digital Imaging and Modeling. (2003)
3. Pons, J.P., Boissonnat, J.D.: Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. In: IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis, USA (2007) 1–8
4. Osher, S., Fedkiw, R.: Level Set Methods and Dynamic Implicit Surfaces. Springer (2003)
5. Osher, S., Senthian, J.: Front propagating with curvature dependent speed: algorithms based on the Hamilton-Jacobi formulation. Journal of computational Physics **79**(1) (1988) 12–49
6. Jung, W., Shin, H., Choi, B.K.: Self-intersection removal in triangular mesh offsetting. Computer-Aided Design and Applications **1**(1-4) (2004) 477–484
7. Furukawa, Y., Ponce, J.: Accurate, dense and robust multi-view stereopsis. In: CVPR. (2007)
8. Pons, J.P., Keriven, R., Faugeras, O.: Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. International Journal of Computer Vision **72**(2) (2007) 179 – 193

9. Hernandez, C.E., Schmitt, F.: Silhouette and stereo fusion for 3-D object modeling. Computer Vision and Image Understanding **96**(3) (2004) 367–392
10. Seitz, S.M., Curless, B., Diebel, J., Scharstein, D., Szeliski, R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Volume 1. (2006) 519–526
11. Kettner, L., Meyer, A., Zomorodian, A.: Intersecting sequences of dD iso-oriented boxes. In Board, C.E., ed.: CGAL-3.2 User and Reference Manual. (2006)
12. Zomorodian, A., Edelsbrunner, H.: Fast software for box intersection. International Journal of Compational Geometry and Applications (12) (2002) 143–172
13. Hert, S., Seel, M.: dD convex hulls and delaunay triangulations. In Board, C.E., ed.: CGAL-3.2 User and Reference Manual. (2006)
14. Gueziec, A., Taubin, G., Lazarus, F., Horn, B.: Cutting and stitching: Converting sets of polygons to manifold surfaces. IEEE Transaction on Visualization and Computer Graphics **7**(2) (2001) 136–151
15. Shin, H., Park, J.C., Choi, B.K., Chung, Y.C., Rhee, S.: Efficient topology construction from triangle soup. In: Proceedings of the Geometric Modeling and PRocessing. (2004)
16. Franco, J.S., Boyer, E.: Exact polyhedral visual hulls. In: British Machine Vision Conference. Volume 1. (2003) 329–338
17. Board, C.E.: CGAL-3.2 User and Reference Manual. (2006)