Vis Comput (2010) 26: 1393–1406
DOI 10.1007/s00371-010-0416-3

ORIGINAL ARTICLE

A lightweight approach to repairing digitized polygon meshes

Marco Attene

Published online: 10 February 2010

Abstract When designing novel algorithms for geometric processing and analysis, researchers often assume that the input conforms to several requirements. On the other hand, polygon meshes obtained from acquisition of real-world objects typically exhibit several defects, and thus are not appropriate for a widespread exploitation.

In this paper, an algorithm is presented that strives to convert a low-quality digitized polygon mesh to a single manifold and watertight triangle mesh without degenerate or intersecting elements. Differently from most existing approaches that globally resample the model to produce a fixed version, the algorithm presented here attempts to modify the input mesh only locally within the neighborhood of undesired configurations.

After having converted the input to a single combinatorial manifold, the algorithm proceeds iteratively by removing growing neighborhoods of undesired elements and by patching the resulting surface gaps until all the "defects" are removed. Though this heuristic approach is not guaranteed to converge, it was tested on more than 400 low-quality models and always succeeded. Furthermore, with respect to similar existing algorithms, it proved to be computationally efficient and produced more accurate results while using fewer triangles.

Keywords 3D scanning · Self-intersection · Degeneracy · Manifold

M. Attene (✉)
IMATI-GE / CNR, Via De Marini, 6, 16149 Genova, Italy
e-mail: marco.attene@cnr.it

1 Introduction

Significant advances in 3D acquisition technologies have brought a gradual change in the way 3D models are produced and handled, and currently digitized models are becoming more and more widespread. In particular, polygon meshes are becoming a de facto standard in several application contexts, and popular 3D shape repositories [1] are currently dominated by this kind of models. Furthermore, polygon meshes are widely used as *native* representation to encode the surfaces produced by most acquisition technologies such as laser-triangulation 3D scanners.

Although producers of 3D digitizers try to make their tools as flexible as possible, each specific application context has its own requirements that define the class of supported 3D models. In industrial design, for example, several downstream applications assume that the mesh does not contain degenerate, or nearly degenerate, elements. In computer graphics, numerous shape analysis tools expect the input mesh to enclose a solid; such tools typically fail if the mesh has holes, or provide unpredictable results if the input has self-intersections.

Often, a 3D scanning session is considered to be complete when all the views (i.e. the range images) are aligned and merged within a single model [2]. While this is sufficient for mere visualization purposes, at this stage polygon meshes may contain degenerate elements, self-intersecting or overlapping parts, surfaces holes, and a number of other "flaws" that make them not appropriate for a widespread exploitation (see Fig. 1).

In this article, an automatic procedure is presented to remove all the aforementioned flaws and transform a raw digitized mesh into a single manifold and watertight triangle mesh. Two main innovations characterize the proposed approach with respect to existing methodologies. First, it

Fig. 1 The raw model of a chair was produced by merging a set of views acquired through laser scanning (*left*). The same model was processed through the algorithm described in this article to become the watertight surface of a polyhedron (*right*). Raw model courtesy of Aim@Shape

strives to modify the mesh as little as possible, which makes the algorithm less *aggressive* than typical volume-based approaches. Second, it is tailored to treat a specific class of meshes: for these meshes the algorithm leads to better results with respect to those produced by existing CAD-oriented methods, in terms of both visual quality and numerical accuracy.

The main application domain of the repairing approach presented here is constituted of raw digitized solid objects. This choice makes it possible to assume that (1) the resulting mesh should be a single connected manifold bounding a polyhedron, and (2) the sampling density should not vary significantly from one part of the mesh to another.

2 Terminology

Undesirable characteristics of a triangle mesh can be roughly classified as *topological* or *geometrical* defects. For this reason, during the development of the algorithm described here, particular care was taken to maintain a neat separation between connectivity and geometry. In the remainder, some notation adapted from [3] is used, and we denote a triangle mesh as a pair (P, Σ), where P is a set of N point positions $\mathbf{p}_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ with $1 \leq i \leq N$, and Σ is an abstract simplicial complex which contains all the topological information. The complex Σ is a set of subsets of $\{1, \ldots, N\}$. These subsets are called simplices and come in three types: vertices $v = \{i\}$, edges $e = \{i, j\}$, and triangles $t = \{i, j, k\}$, so that any non-empty subset of a simplex of Σ is again a simplex of Σ, e.g., if a triangle is present so are its edges and vertices.

The abstract simplicial complex Σ describes a topology, or connectivity, on P. We refer to P as the *geometry* of the triangle mesh $M = (P, \Sigma)$, while we call *connectivity*, or topology, of M the connectivity defined on P through Σ. We say that M is *combinatorially* manifold iff Σ is a combinatorial manifold [16]. In its turn, Σ is a combinatorial manifold iff all its vertices are manifold, and a vertex of Σ is manifold if its neighborhood is homeomorphic to a disk in the topology of Σ.

In a triangle mesh $M = (P, \Sigma)$, for each simplex $\sigma = \{i_1, \ldots, i_n\}$, the set $|\sigma| \subset \mathbb{R}^3$ whose points can be defined as linear convex combinations of the points $\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_n}$ is called the *geometric realization* of σ, and $|M| = \bigcup_{\sigma_i \in \Sigma} |\sigma_i|$ is the geometric realization of M [3]. Thus, the geometric realization is a set of points of $\mathbb{R}^3$ for which an Euclidean topology exists, and we say that $|M|$ is manifold iff the neighborhood of each point in $|M|$ is homeomorphic to a disk. Throughout the remainder of this paper we say that M is *geometrically* manifold, or manifold in the Euclidean sense, if $|M|$ is manifold with respect to the Euclidean topology. Note that a triangle mesh may be manifold in the combinatorial sense and not in the Euclidean one, for example when the mesh self-intersects. Also, a geometrically manifold mesh may not be combinatorially manifold. To obtain such a model, for example, start from a triangle mesh which is both combinatorially and geometrically manifold (e.g. a triangulated sphere), pick an edge $e = \{i, j\}$, add a new triangle $t = \{i, j, k\}$ and set $\mathbf{p}_k := \mathbf{p}_j$. If we relax the requirement of homeomorphism with a disk to the weaker condition of homeomorphism with a disk **or** with a half-disk, we say that M is manifold with boundary, which holds both in the Euclidean and in the combinatorial sense. We define an orientation of an edge as an ordering of its two vertices. Furthermore, we call an orientation of a triangle an equivalence class of ordering of its vertices where $(v_1, v_2, v_3) \backsim (v_{\tau(1)}, v_{\tau(2)}, v_{\tau(3)})$ are equivalent orderings if the parity of the permutation τ is even. Two triangles sharing an edge e are consistently oriented if they induce different orientations on e. A triangle mesh is orientable iff all its triangles can be oriented consistently.

3 Related work

Algorithms to adapt polygonal meshes to particular application contexts are comprehensively described in the literature [11], and specific approaches to remove topological and geometrical defects have been proposed. Algorithms for mesh repairing can be classified into two main categories: surface-based and volume-based methods. Surface-based algorithms try to remove the defects by modifying the input only locally; these methods are not invasive, as they act only where necessary, but unfortunately they typically fail on complex configurations or require the user to interact to resolve ambiguities. Volume-based algorithms use the input to define a new (implicit) surface which is eventually tessellated to produce the output mesh; these methods are typi-

cally much more robust and can treat a wide range of configurations, but unfortunately they introduce modifications in all the parts of the surface, regardless of the presence or absence of defects.

3.1 Surface-based methods

Among surface-based methods, some approaches are designed to fix topological flaws, while some others are more general and attempt to fix the geometry as well. The first subclass of *topological* methods is motivated by the fact that for several applications it is sufficient that the mesh can be encoded through data structures optimized for manifold meshes. Moreover, in several scenarios the acquisition process produces meshes which are *mostly* combinatorial manifolds, in the sense that only a small percentage of vertices are singular. This fact prompted the development of several algorithms that slightly modify the connectivity to edit the singularities without changing anything far from them. To achieve this goal, a widely used approach consists of decomposing non-manifold meshes into simpler parts, splitting at those elements (vertices, edges, facets, etc.) where singularities occur [19, 30]. The result of such a decomposition is a collection of singularity-free components.

In [30] a method is proposed to convert a non-manifold set of triangles to a set of manifold surface meshes. First, non-manifold edges (i.e. edges having more than two incident faces) are identified, and each edge having $2k$ incident faces is split into k manifold edges, so that if these edges are bent by a small amount in the appropriate direction, the resulting shape will not have self-intersections. This representation is called an *edge-manifold* representation, and may still contain isolated non-manifold vertices. Then, to guarantee a manifold topology, one has to identify and duplicate properly the non-manifold vertices. A strategy is suggested to produce a minimum number of vertex duplications [30].

In a similar setting, [19] introduces a strategy based on two high level operations: cutting and stitching. The *cutting* operation involves identifying non-manifold edges and cutting the surface along such edges. Two strategies are available for cutting: a global method, operating on all the surface elements, which is appropriate for cuts covering a large portion of the surface, and a local strategy, operating only on a set of marked vertices and edges, which is more efficient in case of a small number of marked elements. The result of the cutting operation is a manifold surface that may contain boundary edges. Hence, a *stitching* operation is performed, which involves joining two boundary edges while guaranteeing that the surface has a manifold topology. There are two greedy strategies for stitching: *pinching* attempts to simply zip boundary edges created during the cutting operation, while *snapping* attempts to stitch along boundaries other than those, and reduces the number of connected components of the surface. Differently from [30], the method in [19] does not address geometric issues such as self-intersecting surfaces.

In order to fix geometric flaws, Borodin and colleagues [9] remove artifacts such as surface gaps and cracks using a vertex-edge contraction operator and a progressive boundary decimation algorithm. Unfortunately, this method is tailored to fix tessellated CAD models, and typically fails when trying to fix meshes produced by modern acquisition hardware that often contain complex holes bounded by several curves.

Filling complex holes in meshes is a long-standing problem, and several solutions have been proposed, starting from plain triangulation algorithms [5], up to more elaborated approaches that guarantee a certain continuity of the normal field [25], even by using radial basis function interpolators [12].

Based on the observation that tiny handles and tunnels are often generated during the reconstruction of a mesh from raw data, in [21] an algorithm is proposed to locate such *topological noise* and locally re-triangulate the mesh in order to reduce its genus.

Aside from the scientific literature, most commercial systems already provide algorithms to fix specific mesh problems. Among its repairing features, *Geomagic* [17] provides several hole-filling algorithms that can also be launched automatically depending on specific parameters of the holes. Besides automatic hole-filling, *Polyworks* [22] also provides manual tools to deal with really tricky problems. One of the most comprehensive tools for mesh repairing is *Rapidform* [23], which can automatically remove crossing, non-manifold and other degenerate polygons, and fill holes intelligently based on surrounding curvature. The common missing feature, however, is an integrated approach that combines all the algorithms to produce a clean polyhedron out of a raw merge of range images, possibly without requiring user interaction. Though several algorithms are provided by commercial systems, they are tailored for specific flaws, and usually a sequential run of these algorithms is not sufficient (see Sect. 4.2.4).

3.2 Volume-based methods

In some cases the input mesh exhibits a significant amount of flaws, and surface-based approaches become inefficient and do not always succeed. Such *polygon soups* can be more effectively fixed through volume-based methods which, after having converted the mesh to a volumetric representation, produce a completely new mesh approximating the input.

Earlier approaches use a BSP tree to represent the original surface mesh [26]. The requirement of significant computational resources for this approach, however, prompted

the design of a more efficient algorithm in which the input polygon soup is coded through an octree to produce the output-closed manifold [24]. Besides its efficiency in terms of speed and memory consumption, this method also preserves geometric details and sharp features of the original mesh. A major drawback is given by the fact that the output may still contain topological singularities and, as most volume-based approaches, both [26] and [24] are likely to completely remove thin structures.

In [27] an algorithm is proposed to convert the input to a uniformly sampled distance field which is processed and eventually polygonized to produce a simplified version of the input. If the main goal is to *repair* the input, it must be considered that a global resampling is employed, and thus important features such as sharp creases may be spoiled by the processing.

The volume-based paradigm is also used for a specific kind of mesh repair called *mesh completion* [28]. In these approaches the volume is represented by a pair of graphs representing the interior and the exterior of the model. Depending on the desired topology of the repaired model, these approaches fill mesh gaps rather efficiently and effectively. Similarly, volumetric diffusion has also been used in [14] to fill complex holes.

An octree representation is also used by Bischoff and Kobbelt [7] to remove combinatorial and geometrical singularities from tessellated CAD models. This approach is specifically designed for CAD models, and expects the input to be a set of tessellated patches, each without self-intersections. Moreover, possible mesh gaps are filled only if they are smaller than a user-defined threshold; in order to avoid introducing unnecessary distortion while closing all the gaps, such a threshold may need to differ from one part of the model to another, and this makes it difficult to automatize the process in order to build a watertight manifold mesh.

The method presented in [8] uses an octree to completely resample an arbitrary polygon mesh, so that the result is guaranteed to be the boundary of a solid object which stays within a user-prescribed distance from the original mesh. Though providing strong guarantees, this method suffers from the typical drawback of volume-based approaches, that is, it inserts a distortion even in those parts of the surface that do not contain any flaw.

4 Repairing process

The automatic repairing procedure described here is performed in two successive phases: *topology reconstruction* and *geometry correction*. Both the stages have been designed with the objective of modifying the mesh as little as possible.

4.1 Topology reconstruction

This stage of the algorithm aims to convert the set of input polygons into a single combinatorially manifold and oriented triangle mesh. Note that in this section only the connectivity of the input is taken into account, and the goal of the algorithm described here is to construct a single combinatorial manifold without boundary.

Most graphic formats widely used to share 3D models (OFF, VRML, PLY, etc.) encode surface meshes through *indexed face sets*; specifically, these file formats contain a first block specifying the position of the vertices, and a second block in which each polygon is represented through a sequence of indices of vertices in the first block. Clearly, files of this type are not guaranteed to represent a well-defined polyhedron, while they may easily encode non-manifold and/or non-orientable sets of polygons.

For the sake of simplicity, our algorithm first converts each polygon to a set of triangles through triangulation. Then, the first step in the topology reconstruction amounts to building an explicit connectivity between adjacent triangles. After having triangulated the input polygons, triangle connectivity can be reconstructed as follows: first, an empty list L of edges is created, then for each triangle $\{i, j, k\}$ its three edges $\{i, j\}$, $\{j, k\}$ and $\{k, i\}$ are inserted into L and, finally, L is sorted according to a suitable order relation; specifically, for a generic pair of edges $e_1 = \{i, j\}$ and $e_2 = \{k, n\}$ in L, without loss of generality we assume that $i \geq j$ and $k \geq n$, and define a lexicographical order relation as follows:

$$e_1 \leq e_2 \quad \text{iff } i < k \quad \text{OR} \quad (i = k \quad \text{AND} \quad j \leq n) \tag{1}$$

Two triangles are adjacent iff they induce consecutive edges in the sorted list L. Thus, by keeping a link between each edge e_i in L and the triangle that originated e_i, it is possible to retrieve all the triangles adjacent to a given one in optimal time. Specifically, let t be a triangle bounded by the edges e_1, e_2 and e_3. The triangles adjacent to t can be obtained by accessing e_i in L and by looking at its successive and previous edges in the list; if such previous (or successive) edge has the same vertices as e_i, then the triangle that originated it is adjacent to t. By explicitly encoding the edges in the data structure, this approach to compute triangle adjacencies requires a number of operations which is linearly proportional to the number of adjacent triangles, thus it is optimal. Therefore, this makes it possible to actually *walk* across adjacent triangles in optimal time, which is necessary to implement region-growing algorithms to, for example, calculate a consistent orientation of the triangles in a mesh.

At this point, a fundamental step of the algorithm aims to remove topological singularities. To achieve this objective, we chose to rely on the approach described in [19] because both the cutting strategy and the pinching phase are relatively cheap in terms of computational resources, which

is important when dealing with big meshes. If the resulting manifold mesh is made of more than one connected component, only the *biggest* component is kept, that is, the component made of the largest number of triangles. In this phase, isolated vertices are considered as connected components on their own, and hence are removed.

Next, the algorithm assigns an orientation to one *seed* triangle, and propagates the orientation to neighboring triangles; specifically, let t be an already processed triangle and let s be a triangle adjacent to t and not already processed. If s and t are not consistently oriented, then s is *inverted*, that is, two of its three vertices are swapped in the ordered set. Once all the triangles have been visited and possibly inverted, the mesh is traversed and possibly cut along edges having non-consistently oriented incident triangles. Clearly, if such a cut takes place, the combinatorial manifold provided by the algorithm does not completely conform to our requisites as it has a boundary; in other words, in this case the algorithm fails to produce a single combinatorial manifold without boundary.

If no cuts were necessary, the mesh may still have holes that were already present in the original input. For each hole, a patching procedure such as the one described in [25] in launched. This procedure first triangulates the hole as described in [5], and then inserts new vertices in the patching triangulation so as to resemble the sampling density of the surrounding region: these new vertices are moved to positions that minimize the normal-field variation. Note that though the hole-filling procedure is guaranteed to converge, the resulting patches are not guaranteed to be intersection-free.

To summarize, the topology reconstruction phase proceeds as described in Algorithm 1.

Algorithm 1 Main steps of the topology reconstruction phase

Require: An indexed face set $\mathcal{F}$.
Ensure: A single combinatorial manifold $\mathcal{M}$.

1: Triangulate all the faces in $\mathcal{F}$
2: Initialize $\mathcal{M}$ with the resulting triangles
3: Compute the triangle–triangle adjacency relations
4: Remove singularities as described in [19]
5: Remove all the connected components but the largest one
6: Orient the mesh
7: **if** cuts were necessary **then**
8: warn the user and terminate
9: **else**
10: patch mesh holes with new triangles [25]
11: **end if**

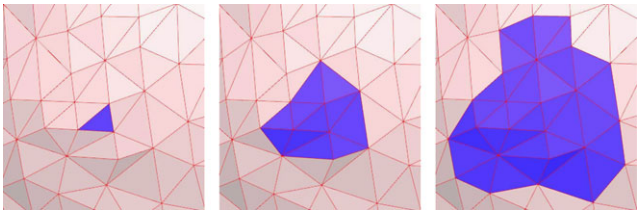


Fig. 2 The zero-order simplicial neighborhood $N_0(t, M)$ of a triangle t (*left*), its first-order neighborhood $N_1(t, M)$ (*middle*), and its second-order neighborhood $N_2(t, M)$ (*right*)

4.2 Geometry correction

Once the connectivity of the mesh is "fixed" as described in Sect. 4.1, the geometric aspects are dealt with as follows. Typical *geometric flaws* in a triangle mesh include degenerate elements (i.e. triangles with null area) and self-intersections.

4.2.1 Higher-order simplicial neighborhoods

In order to repair geometric flaws, the algorithm presented here makes use of the notion of *simplicial neighborhood*. Let L be a submesh of a combinatorial manifold M, possibly with boundary, and let $N(L, M)$ be the submesh defined by a set of triangles that share at least a vertex with L. $N(L, M)$ is said to be the simplicial neighborhood of L [31]. In the particular case in which L is made of a single triangle t, we define the notion of *higher-order* simplicial neighborhood. Specifically, we call kth-order simplicial neighborhood of a triangle t in M the submesh of M defined as $N(N(\ldots(N(t, M)\ldots), M), M)$, with $k-1$ nested levels. Thus, the first-order simplicial neighborhood corresponds to the simplicial neighborhood, the second-order is the simplicial neighborhood of the simplicial neighborhood, and so on. In the remainder a compact notation is used, and the kth-order simplicial neighborhood of a triangle t in M is denoted as $N_k(t, M)$. Finally, by convention, we say that the zero-order simplicial neighborhood of a triangle is the triangle itself, that is, $N_0(t, M) = t$. Examples of higher-order simplicial neighborhoods are shown in Fig. 2.

4.2.2 Degeneracy removal

While the removal of exactly degenerate elements can be easily performed using [10], removing *nearly* degenerate triangles appears to be much harder. Unfortunately, however, skinny triangles may be the source of several problems even if they are not exactly degenerate. The attempt to measure the level of degeneracy of a triangle has led to several definitions, in particular in the study of Finite Element methods [33] where well-shaped simplices are fundamental for a

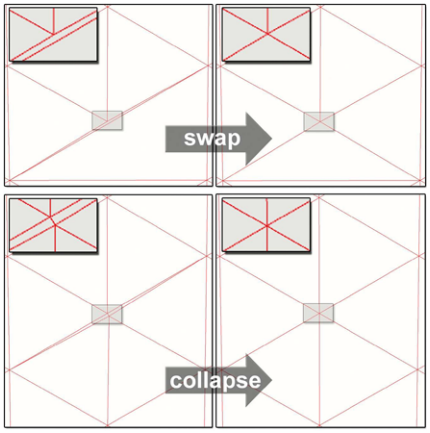


Fig. 3 Nearly degenerate triangles are removed through edge swap (*top*) or edge collapse (*bottom*) as described in Sect. 4.2.2. In this image the value of ϵ is exaggerated on purpose to better convey the concept

robust computation. In order to assess the degeneracy of triangles, we chose to implement a strategy based on the Epsilon Geometry introduced in [20]. In our framework, the value of ϵ is an angle. If a triangle has an angle smaller than ϵ or bigger than $\pi - \epsilon$, such a triangle is declared to be degenerate. In this case, the algorithm strives to resolve the degeneracy through swapping and contraction of edges, inspired from ideas of [10]. Specifically, triangles having a nearly flat angle are treated by swapping the edge opposite to such angle, while triangles having a nearly null angle are removed by collapsing the edge opposite to such angle to its midpoint (checks are performed in this order). The value of ϵ may be tuned by the user. To run the experiments shown in this paper, a default value of $\arcsin(10^{-5})$ was used; in [3], this particular value was proven to be a good compromise between precision and robustness in most practical cases. Schematic examples are shown in Fig. 3.

Note that due to topological constraints (i.e. the mesh must remain a combinatorial manifold), not all the edge swaps and the edge collapse operations can be performed [15]. Therefore, it may happen that a degeneracy would need to be treated but actually cannot be due to such constraints. In this case, we remove all the triangles belonging to the simplicial neighborhood $N(t_i, M)$ of the degeneracy t_i and patch the resulting gap using [5]. Then, we run the swap/collapse routine within the patch and, if once again it does not succeed, for each remaining degeneracy t_{j_k} we compute the second-order simplicial neighborhood $N_2(t_{j_k}, M)$, remove it from the mesh and triangulate the gap again, and so on. In other words, at each iteration we enlarge the size of the neighborhood of the degeneracies that could not be solved through the swap/collapse routine. The process stops with failure after a prescribed number of attempts. In our prototype implementation the number of such iterations is limited to 3; this value provides an extremely high percent of success (all the degeneracies in our 400 test models could be resolved) while keeping the modification enclosed within a tight neighborhood (i.e. a third-order simplicial neighborhood) of the original flaw. After the removal of the patch, but before the gap re-triangulation, the algorithm needs to check and possibly remove small disconnected components that detached from the main object. When k is sufficiently large, in fact, the kth-order simplicial neighborhood of a triangle may be non-simply connected, and in this case its removal would leave little disconnected pieces that need to be removed. Also, in the hole left by the removal of a simplicial neighborhood, some of the bounding vertices may be non-manifold: in this case, each such singularity is automatically duplicated [19] and, by construction, the patching triangulation exhibits (exactly) degenerate triangles that can be easily removed through a single edge-collapse.

Algorithm 2 provides a sketch of the proposed iterative approach.

Algorithm 2 Algorithm for degeneracy removal

Require: A combinatorial manifold $\mathcal{M}$ and an integer threshold *max_iterations*

Ensure: A combinatorial manifold $\mathcal{M}'$ and a status notice (success/failure)

$\mathcal{M}' := \mathcal{M}$
Let S be the set of all the triangles of $\mathcal{M}'$
for $k = 1$ to *max_iterations* **do**
    Run the swap/collapse algorithm within S
    Let T be the set of degeneracies in S untreatable due to topological constraints
    if $T = \emptyset$ **then**
        terminate with **success** /* $\mathcal{M}'$ *is degeneracy-free* */
    end if
    Let R be the union of the kth-order simplicial neighborhoods of the t_is $\in T$
    Remove R from $\mathcal{M}'$
    Remove possible disconnected components from $\mathcal{M}'$
    Patch the remaining gaps with a new set P of triangles
    $S := P$
end for
if S contains degenerate triangles **then**
    terminate with **failure** /* $\mathcal{M}'$ *has degenerate triangles* */
else
    terminate with **success** /* $\mathcal{M}'$ *is degeneracy-free* */
end if

4.2.3 Removal of self-intersections

In several application contexts the input mesh is assumed to enclose a polyhedron, and thus it is required not to have self-intersections. While it is relatively easy to check a mesh for self-intersections, it is not that trivial to remove them while modifying the mesh only locally. Similarly to the treatment of degenerate triangles, the algorithm proposed here is based on an iterative removal/gap-filling approach.

Clearly, the detection of intersecting triangles cannot be performed by simply checking each pair of triangles, as this would lead to a quadratic complexity which is not affordable for even relatively small meshes. Hence, the automatic detection must rely on some kind of optimization, which is often based on spatial subdivision. Typical approaches are based on kd-trees, octrees [32] or on hierarchies of object-oriented bounding boxes [18]. In our context, however, we mostly target the processing of digitized models, which have the characteristic that triangles do not have much variation in size. For this reason, we rely on a uniform space subdivision that can be efficiently computed as described in [29]. Besides being efficient from the point of view of the computation time, using this solution is also cheap in terms of memory consumption, as only the voxels that actually contain portions of triangles are encoded. Within each voxel, a *brute-force* algorithm is run which checks all the possible pairs of triangles for intersection. During experimentation, a fixed grid size made of 100^3 voxels proved to behave satisfactorily for scanned models ranging from few thousand to nearly 4 million faces. Probably a more elaborate study of the grid size, which might adapt to the mesh at hand, would lead to slightly better performances; however, this is not the main focus of this article, and the fixed cubic grid used was good enough for all our test cases.

After having identified all the pairs of intersecting triangles, the algorithm just removes them (both the intersecting and the intersected triangles are deleted for each pair). Possible disconnected components resulting after the removal are deleted too, and the remaining gaps are filled using [5]. At this point, the voxel grid is updated with the new triangles that have been created to fill the gaps, and the self-intersection check is run again within the updated voxels. For each triangle that still intersects other parts of the mesh, its first-order simplicial neighborhood is removed, the gaps filled and the voxels updated. If there are still intersecting triangles, their second-order neighborhood is removed, and so on. The process stops with failure when the simplicial neighborhood reaches a prescribed maximum order. As in the degeneracy removal step, in our prototype implementation the number of such iterations is limited to 3. In Fig. 4 an example is shown in which all the self-intersections could be removed in two iterations.

Algorithm 3 provides a sketch of the proposed approach.

Algorithm 3 Algorithm for removal of self-intersections

Require: A combinatorial manifold $\mathcal{M}$ and an integer threshold *max_iterations*

Ensure: A combinatorial manifold $\mathcal{M}'$ and a status notice (success/failure)

1: $\mathcal{M}' := \mathcal{M}$
2: Let S be the set of all the triangles of $\mathcal{M}'$
3: Let G be a uniform 100^3 voxel grid tightly enclosing $\mathcal{M}'$
4: **for** $k = 0$ to *max_iterations* **do**
5: Let H be the set of voxels intersecting at least a triangle of S
6: Check for triangle–triangle intersections within each voxel of H
7: Let T be the set of intersecting triangles detected above
8: **if** $T = \emptyset$ **then**
9: terminate with **success** /* $\mathcal{M}'$ *is not self-intersecting* */
10: **end if**
11: Let R be the union of the kth-order simplicial neighborhoods of all $t \in T$
12: Remove R from $\mathcal{M}'$
13: Remove possible disconnected components from $\mathcal{M}'$
14: Patch the remaining gaps with a new set P of triangles
15: $S := P$
16: **end for**
17: Let H be the set of voxels intersecting at least a triangle of S
18: **if** S contains intersecting triangles **then**
19: terminate with **failure** /* $\mathcal{M}'$ *has self-intersections* */
20: **else**
21: terminate with **success** /* $\mathcal{M}'$ *is not self-intersecting* */
22: **end if**

4.2.4 Integrated flaw removal

Since the objective of the repairing algorithm is to remove both degeneracies and self-intersections, the above-described iterations must be integrated into a single loop. Notice that the two routines cannot be simply launched in sequence, because there is no guarantee that the degeneracy removal does not introduce self-intersections and vice versa. Indeed, after having deleted all the faulty triangles and the resulting disconnected components, the remaining holes are patched as described in [5]: this hole-filling algorithm sim-

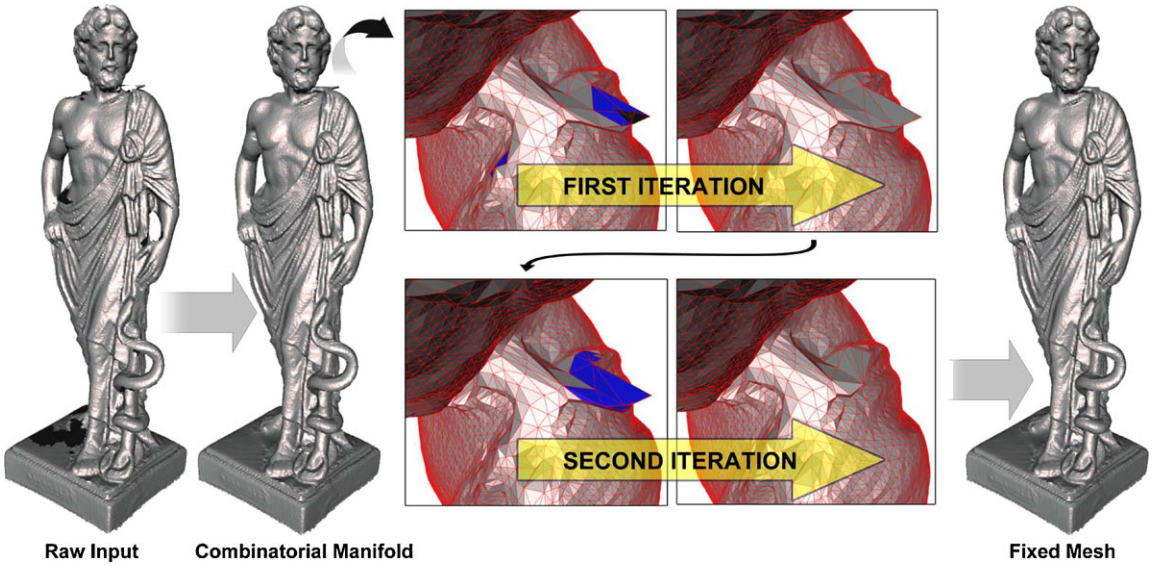


Fig. 4 After having converted the model to a combinatorial manifold without degeneracies, self-intersecting triangles (depicted in *blue*) are located and their higher-order simplicial neighborhoods are re-triangulated as described in Sect. 4.2.3. After two iterations the mesh no longer contains any self-intersection. Note that vertices which are entirely in the blue regions are simply eliminated from the mesh

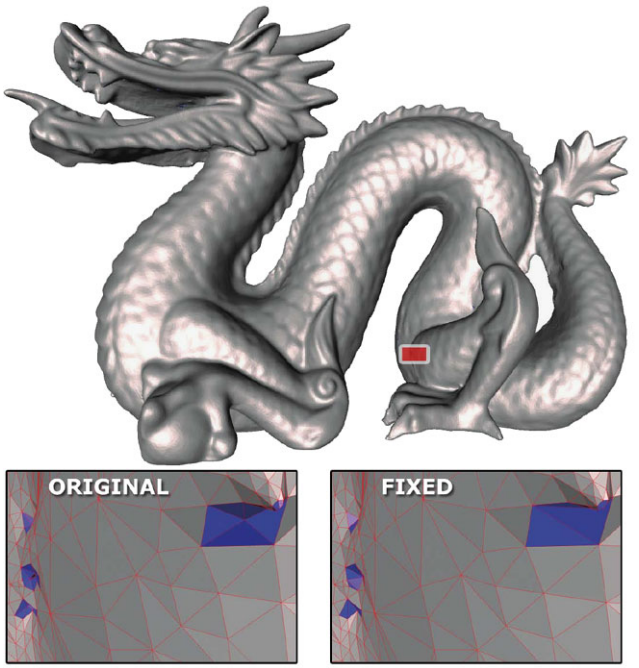


Fig. 5 Self-intersection removal in the Stanford dragon. All the intersecting triangles were selected (*bottom-left*, line 7 in Algorithm 3) and removed. The resulting holes were patched (*bottom-right*, line 14 in Algorithm 3). The rear-leg region shown in the magnification was fixed in a single iteration

ply triangulates the boundary polygon without adding any new vertex. In [5], the authors prove that there exist some 3D polygons for which all the possible triangulations self-intersect, and verifying that a given polygon can be triangulated without self-intersections is an NP-complete problem. Therefore, some heuristics have been introduced to compute the triangulation, and though the algorithm is guaranteed to converge to a solution, there is no guarantee that such a solution is a geometrically manifold mesh without degeneracies.

Thus, in its integrated version, the algorithm alternates between degeneracy and intersection treatment within a unique loop, until all the flaws are removed. As for the individual sub-algorithms described in Sects. 4.2.2 and 4.2.3, the algorithm stops with failure after a prescribed number of attempts (10 in our experimental implementation).

In the example in Fig. 5, the original model has 407 boundary loops, 26 082 degenerate triangles and 6711 intersecting triangles. All the loops have been filled by Algorithm 1, whereas after the first iteration of Algorithm 4 the model still has 31 degenerate triangles and 8 self-intersections. After the second iteration of Algorithm 4 the model is completely fixed.

Summarizing, the overall geometry correction can be implemented through Algorithm 4. The whole procedure, which includes both the topology reconstruction and geometry correction phases, is shown in Fig. 6.

4.2.5 Adaptation to other domains

The algorithm presented here has been designed to process raw digitized models, hence it should not be expected to produce accurate results on meshes created differently; unsuitable models include sparsely tessellated CAD surfaces or even digitized models which have already been processed somehow, for example through mesh simplification. In these

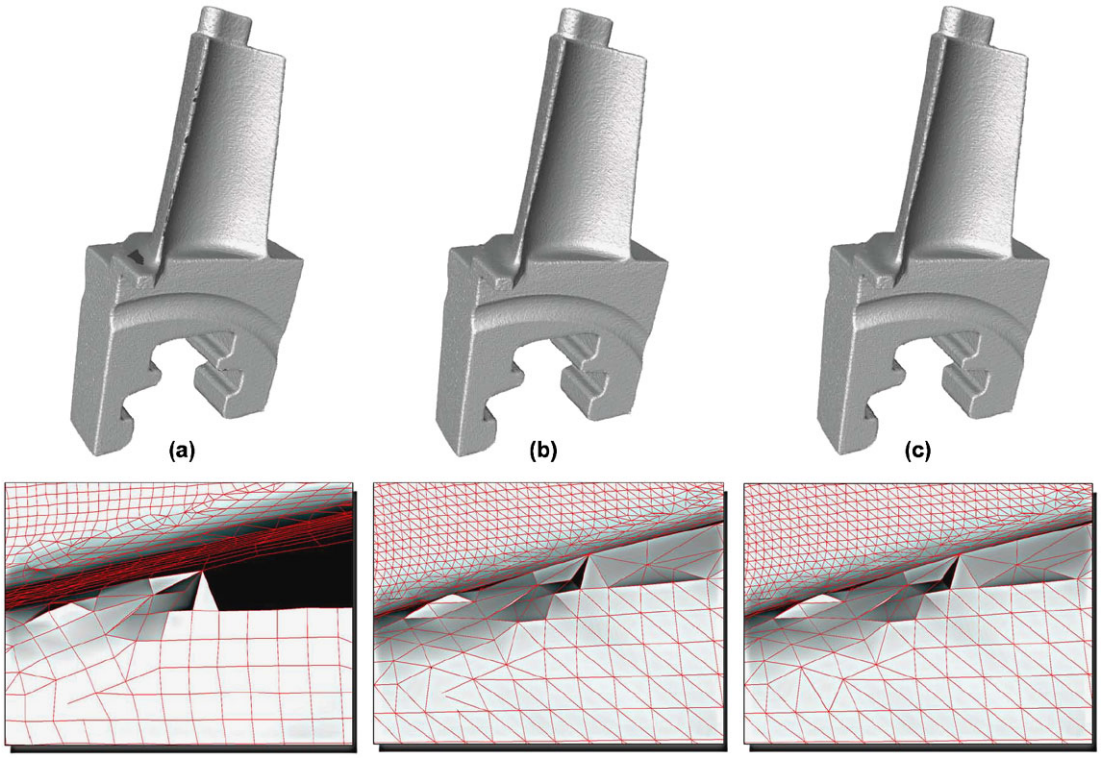


Fig. 6 **a** Original raw mesh coming from the fusion of 11 aligned range images (courtesy of Aim@Shape). Note the hole on the right and the self-intersecting facets on the left of the magnification. **b** The combinatorial manifold resulting from the topology reconstruction phase. The hole has been filled, whereas some triangles are still self-intersecting. **c** The final fixed mesh produced by processing the combinatorial manifold through the geometry correction phase

Algorithm 4 Main steps of the geometry correction phase

Require: A combinatorial manifold $\mathcal{M}$
Ensure: A combinatorial manifold $\mathcal{M}'$ and a status notice (success/failure)

1: $\mathcal{M}' := \mathcal{M}$
2: *max_iterations* $:= 10$
3: **for** $k = 0$ to *max_iterations* **do**
4: Run Algorithm 2 with parameters $\mathcal{M}'$ and 3. Let $\tilde{\mathcal{M}}$ be the output.
5: $\mathcal{M}' := \tilde{\mathcal{M}}$
6: Run Algorithm 3 with parameters $\mathcal{M}'$ and 3. Let $\tilde{\mathcal{M}}$ be the output.
7: $\mathcal{M}' := \tilde{\mathcal{M}}$
8: **if** both the algorithms succeeded and $\mathcal{M}'$ has no degenerate faces **then**
9: terminate with **success** /* $\mathcal{M}'$ has no geometric flaw */
10: **end if**
11: **end for**
12: terminate with **failure**

cases the algorithm can be run as well, but the modifications introduced may easily become too coarse (see Fig. 7(b)). This behavior can be explained by observing that, differently from raw digitized shapes, other kinds of models may be characterized by a sparse vertex sampling, either locally or globally. Clearly, re-triangulating a simplicial neighborhood in a sparsely sampled region may easily become a *macroscopic* operation. Thus, an adaptation of the degeneracy removal phase is necessary to process the aforementioned *unsuitable* models while maintaining the modifications within tolerable bounds.

To achieve this result, in our prototype implementation we have added an optional local refinement driven by a user-specified tolerance value. Specifically, before proceeding with Algorithm 4, the mesh is analyzed to detect all the faulty triangles (i.e. nearly degenerate or intersecting other triangles) having at least an edge longer than the prescribed threshold. Each such triangle is subdivided in four subtriangles by splitting its three edges at their midpoints. Then, the mesh is re-analyzed and the resulting defects are subdivided again, and so on, until all the edges of faulty triangles become shorter than the prescribed threshold. An example of the effect of this improvement is shown in Fig. 7(c).

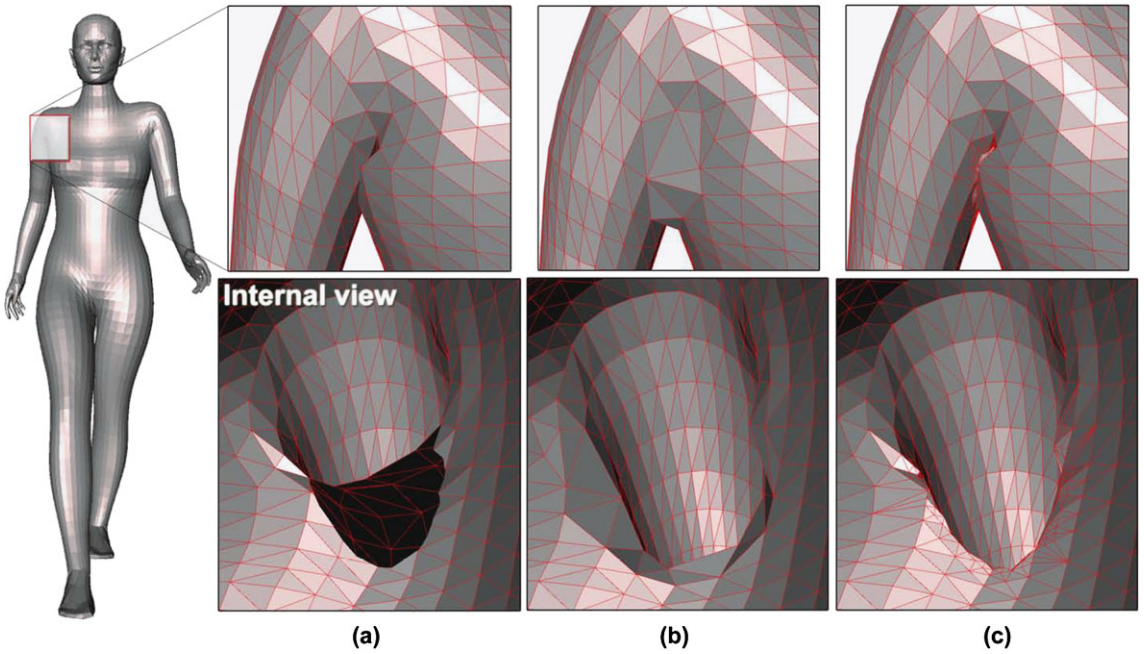


Fig. 7 The original woman model (**a**) is too sparsely sampled, and the repairing algorithm produces coarse artifacts near formerly self-intersecting areas (**b**); by performing few local iterations of midpoint subdivision within such areas before correction, the resulting patches become more accurate (**c**)

5 Experimental results

The repairing algorithm described in this paper has been extensively tested on several models, and it always succeeded. Experiments were run both on raw scanned models downloaded from the Internet [1] and on models created on purpose through digitization of real objects; for this purpose, a Minolta Vivid 910 laser scanner was used to acquire the shapes, whereas range images were aligned and merged through Minolta's software *Polygon Editing Tool*. On the resulting polygonal meshes the vertex density is rather high and uniform, and the modifications introduced in order to repair each geometric flaw are hardly visible unless a significant zoom is used. In numbers, however, the distance between the original mesh and the repaired model depends on the resolution used by the scanner, on the distance between the surface and the scanner's sensor, and on the specific lens employed. In our experiments, we scanned objects whose absolute size is at most 520 millimeters (length of the bounding-box diagonal), the resolution of each range image was 640×480 pixels, the focal distance was approximately 600 millimeters, and a "tele" lens provided by Minolta was used. In these conditions, the maximum distance between the original model and its repaired version was measured using the Metro tool [13], and we have verified that it never exceeded 0.0098 times the model's bounding-box diagonal (see Fig. 8 and Table 1). Note that on laser-scanned models this value is comparable with the distance of a typical outlier from the underlying surface. Moreover, such a maximum distortion is extremely localized, as demonstrated by the value of the mean distance which is typically smaller by several orders of magnitude.

Since they dealt with raw scanned models, all the aforementioned experiments were run without the adaptation discussed in Sect. 4.2.5. Nevertheless, such adaptation was used during the testing of the algorithm on another set of 40 sparsely sampled models (see Fig. 7 for an example). For these tests, the threshold length for edges of faulty triangles was set to 0.005 times the length of the bounding-box diagonal. Even in these cases, the algorithm has always been able to terminate with success.

It is worth mentioning that the algorithm presented here proved to be computationally efficient. As an example, the whole repairing of the *bimba* model (3.7 million triangles) has taken 87 seconds, which is comparable with the time spent by [7] in its best run to repair the *ventilator* model (269 thousand triangles, Fig. 11 in [7]), and is significantly better than the 126 seconds spent by [27] to produce the fixed *woman* model shown in Fig. 9d. More timing results are reported in Table 1.

5.1 Comparison with existing approaches

We compared the algorithm described in this paper with other three approaches, two employing a global surface resampling ([27] and [24]), and one attempting to fix defects by modifying the mesh only locally around them [7].

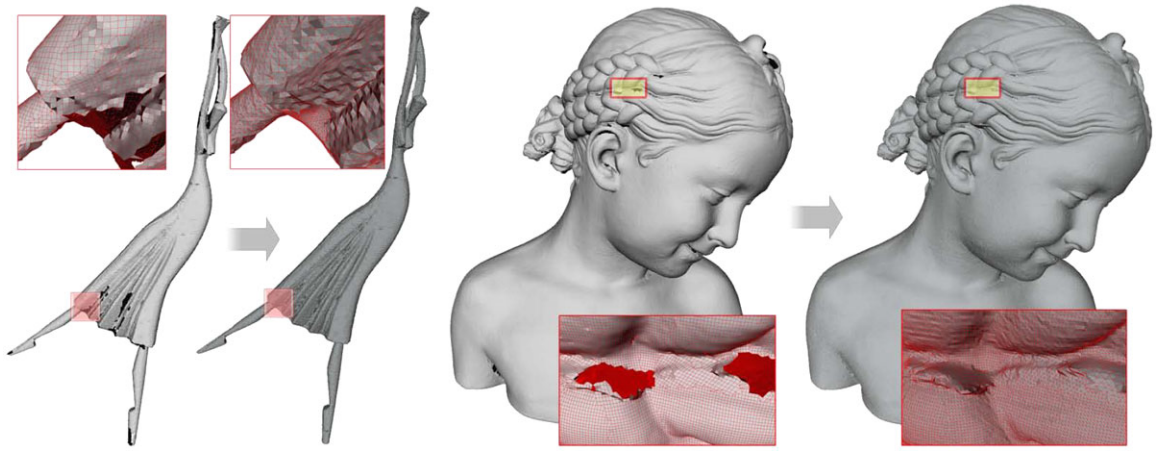


Fig. 8 Two examples of test models successfully repaired. Original raw data courtesy of Aim@Shape

Table 1 *Distortion*. For each model, this table reports the number of triangles in the original and in the fixed mesh, and distance information as reported by Metro [13]: namely, the maximum absolute distance of the original from the fixed mesh, the mean and the rooted-mean-square distances, and the maximum distance with respect to the length of the bounding-box diagonal. Distances are expressed in millimeters, with the exception of the *woman* and the *dragon* models for which no specific unit of measurement was available. Time is expressed in seconds, and does not include input/output from/to file

Model Name	Fig.	Input Triangles	Output Triangles	Max Dist	Mean Dist ($\times 10^{-5}$)	RMS ($\times 10^{-3}$)	Max Dist wrt bb-diag	bb-diag	time
chair	1	98 523	101 784	1.116099	1.1	0.435	0.002460	453.70	3.2
dancer	8	140 804	149 206	0.994584	6.9	5.676	0.002831	351.32	4.1
sofocle	11	350 664	410 138	0.472599	2.3	0.445	0.003317	142.48	9.4
blade	6	389 103	391 314	0.035633	0.1	0.087	0.000211	168.88	7.8
bimba	8	3 745 150	3 755 938	1.169993	5.2	5.297	0.002242	521.85	87.1
woman	7	11 534	12 634	0.001109	0.5	0.051	0.000600	1.74	1.2
dragon	5	871 414	844 248	0.002774	0.1	0.033	0.009748	0.27	16.1

Although the method in [27] proved to be robust and always succeeded, it requires a significant amount of resources to produce satisfactory results. Better performances are achieved by [24] but, in both the cases, the results are isosurfaces of discrete scalar fields; hence, to provide sufficiently accurate results, the resolution of the volume grids must be significantly high, with consequent high triangle counts in the corresponding output isosurfaces. With respect to the method presented in [7], our approach appears to be slightly more general for some aspects: in [7] the algorithm is able to detect and fix only intersections between different tessellated *patches*, and the new triangles inserted through the dual-contouring approach are not guaranteed to be intersection-free. Nevertheless, we could compare our results with those coming from an adapted version of [7] in which all the self-intersections can be detected, without the limitation of belonging to different patches. In this case, we could verify that our approach is less *invasive* in the sense that the modifications introduced are less visible; furthermore, with a comparable accuracy, the results of our approach contain significantly less additional triangles, and in all our experiments the results are perfectly free of any intersection or degeneracy (see Fig. 9). As a final remark, it is worth to remind that the algorithm presented here is designed to fix meshes which are assumed to enclose solid objects. In such a context, [7] may be difficult to use because, in order to avoid introducing unnecessary distortion while closing all the gaps, the threshold may need to differ from one part of the model to another. On the other hand, however, the method presented here is not suitable to treat meshes with *desired* boundaries; in this regard, the approach proposed in [7] gives more flexibility.

5.2 Limitations and failure cases

The repairing procedure requires to access the various parts of the mesh, and unfortunately an obvious solution was not found to subdivide the input into smaller pieces to be

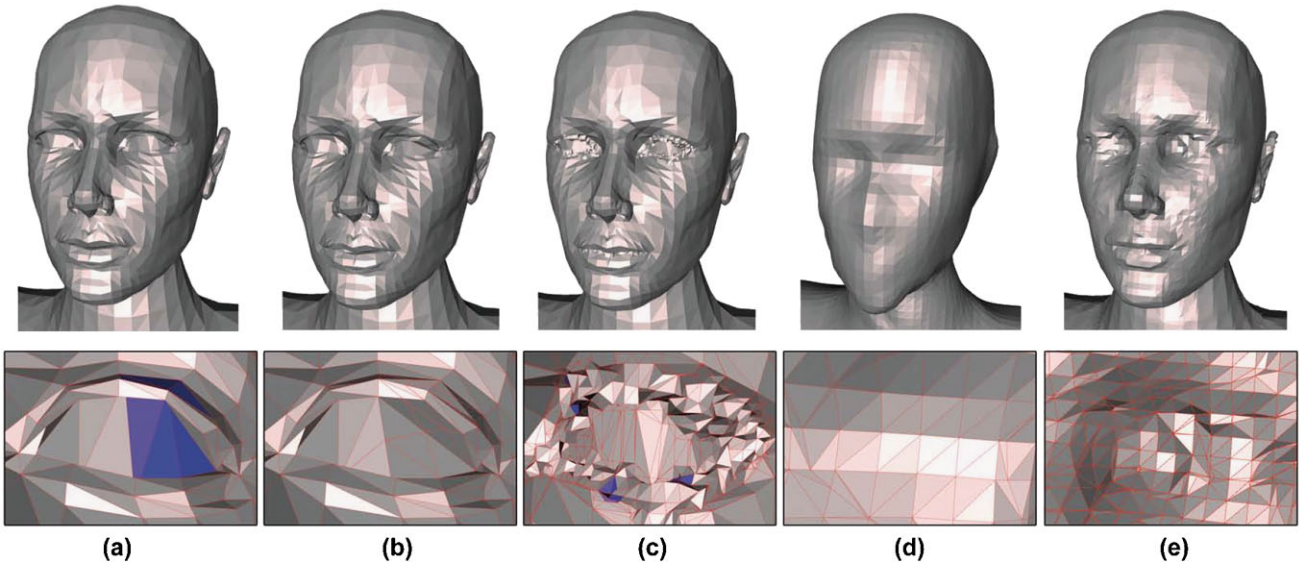


Fig. 9 An example showing a comparison of the results achieved by **b** our approach, **c** [7], **d** [27] and **e** [24], starting from a common original mesh **a**. Self-intersecting triangles are shown in *blue*. Besides the evident benefit in terms of visual quality, our algorithm provides more accurate results using less triangles. In this example, the original **a** is made of 11 534 triangles, while **b**, **c**, **d** and **e** contain 12 634, 17 174, 110 908 and 363 776 triangles respectively. In contrast, the maximum distance from **a** is 0.001109 for **b**, 0.001391 for **c**, 0.018388 for **d**, and 0.006189 for **e**

processed separately. To fix huge meshes, a possibility is to use dedicated PCs equipped with large memory made accessible through 64 bit operating systems. Our 32 bit implementation could treat models made of up to 4 million triangles without switching to virtual memory.

The algorithms discussed in this article are not guaranteed to terminate with success, thus it is worth mentioning which are the possible causes of failure. While Algorithm 1 fails only if the input is intrinsically not orientable (e.g. a Möbius strip), Algorithms 2 and 3 can fail if the hole-filling procedure creates unwanted elements (degeneracies or self-intersections) at all the iterations. Though this is extremely rare for digitized meshes, it is possible to synthesize a pathological case in which the configuration of the holes makes the filling algorithm produce self-intersections at all the iterations (see Fig. 10).

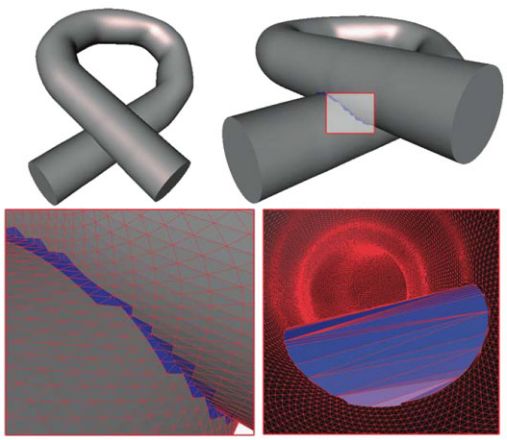


Fig. 10 A self-intersecting bent tube representing a synthesized failure case (*top-left*). After having identified the intersecting triangles (*top-right* and *bottom-left*), the algorithm removes them and the resulting disconnected pieces. By attempting to minimize the normal variation, the hole filler patches the two holes with cylindrical surfaces (*bottom-right*, inner view) that intersect each other at all the iterations

6 Applications

The fully automatic nature of the tool proposed in this article opens various possibilities. For example, it can be incorporated as an optional post-loading stage within novel 3D processing systems; if the input can be successfully fixed, the system may activate additional functionalities which are defined only for meshes bounding polyhedra. Also, the new algorithm can be plugged into a batch script to automatically fix big shape repositories without any user intervention. In this case, possible failures can be simply logged and later analyzed and treated manually using appropriate tools [3].

6.1 Benchmark generation

As a practical application, the algorithm presented here has been used to automatically produce the SHREC 2008 benchmark on watertight models [6]. After having collected the 400 raw meshes constituting the previous year's version of the benchmark, we have created a batch script to run the repairing algorithm on each model. As a result, we obtained 400 clean meshes that could undergo further au-

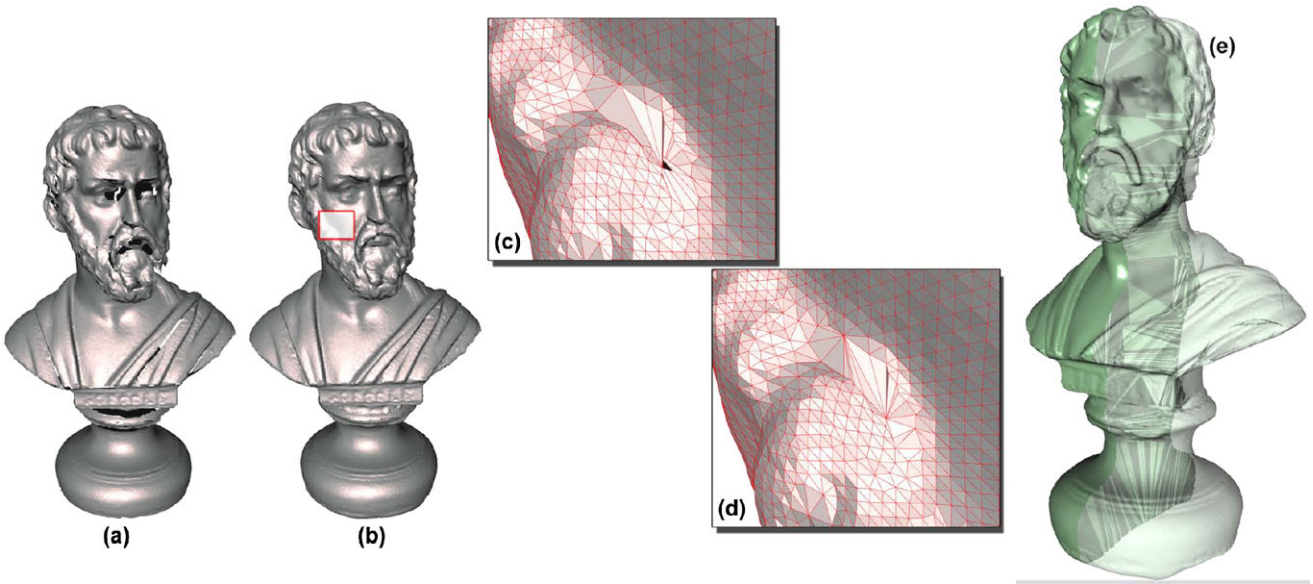


Fig. 11 The original raw mesh has holes, degeneracies and self-intersections (**a**). After a global topology reconstruction which includes hole-filling (**b**), geometric flaws (**c**) are fixed as described in Sect. 4.2 (**d**). The final clean mesh can be successfully tetrahedrized using *tetgen* (**e**)

tomatic processing which was necessary to create the final benchmark. Most input models were already clean, and thus were not modified by the algorithm, whereas all the other meshes were successfully fixed. Specifically, the algorithm presented in this paper was able to successfully fix all the 5584 degenerate triangles and all the 17 961 self-intersections present in 146 out of 400 meshes constituting the whole data set.

6.2 Conversion to tet mesh

The generation of tetrahedral meshes out of raw input surface meshes represents a further example in which the automatic nature of the algorithm can be exploited. In our experiments several fixed models, including the 400 fixed meshes discussed in Sect. 6.1, have been converted to tetrahedral meshes through the *tetgen* open-source software tool [34]. This tool was able to tetrahedrize all the repaired models, and this is a further demonstration of the good quality of the results of the algorithm presented here. The resulting tetrahedral meshes could be successfully used to test a novel shape segmentation method [4]. Figure 11 shows an example of the whole processing pipeline, from the input raw surface mesh to the output tetrahedral mesh.

7 Conclusions

The algorithm described in this paper, the experimental results obtained so far and the comparison with existing approaches, demonstrate that raw digitized 3D models can be improved through *ad hoc* procedures with several advantages. Among these, we have shown that it is possible to significantly reduce the distortion introduced in the fixed model and, at the same time, it is possible to design completely automatic conversion mechanisms. This latter aspect is not negligible, as it makes it possible to automatically correct and convert even big databases without user intervention.

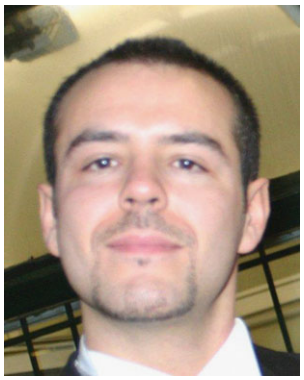
Future research will investigate strategies to computationally detect the suitability of an input model to be repaired; we argue that the number of facets and their sampling distribution can provide useful indications in this sense, and may lead to automatic approaches to set plausible thresholds for the adaptation scheme proposed in Sect. 4.2.5.

Acknowledgements This work has been partially supported by the FP7 FOCUS K3D Coordination Action. Special thanks are due to Prof. Leif Kobbelt for the time spent to adapt his algorithm, for having provided results for comparison, and for the valuable information provided about his work. Also, I would like to thank Stephen Bischoff and his co-authors for having provided an implementation of the repairing method by Nooruddin and Turk that could be used to further compare the results presented here against the state of the art. Last but not least, I am grateful to Bianca Falcidieno, Michela Spagnuolo and all the members of the *Shape Modeling Group* at IMATI-CNR for helpful discussions.

References

1. AIM@SHAPE shape repository (2004). http://shapes.aimatshape.net/
2. Albertoni, R., Papaleo, L., Robbiano, F., Spagnuolo, M.: Towards a conceptualization for shape acquisition and processing. In: Proc. of 1st International Workshop on Shapes and Semantics (2006)

3. Attene, M., Falcidieno, B.: Remesh: An interactive environment to edit and repair triangle meshes. In: Shape Modeling and Applications, pp. 271–276 (2006)
4. Attene, M., Mortara, M., Spagnuolo, M., Falcidieno, B.: Hierarchical convex approximation for fast region selection. Comput. Graph. Forum **27**(5), 1323–1333 (2008)
5. Barequet, G., Sharir, M.: Filling gaps in the boundary of a polyhedron. Comput. Aided Geom. Des. **12**(2), 207–229 (1995)
6. Biasotti, S., Attene, M.: Shrec08 entry: Report of the stability track on watertight models. In: Shape Modeling and Applications (2008)
7. Bischoff, S., Kobbelt, L.: Structure preserving cad model repair. Comput. Graph. Forum **24**(3), 527–536 (2005)
8. Bischoff, S., Pavic, D., Kobbelt, L.: Automatic restoration of polygon models. ACM Trans. Graph. **24**(4), 1332–1352 (2005)
9. Borodin, P., Novotni, M., Klein, R.: Progressive gap closing for mesh repairing. In: Advances in Modelling, Animation and Rendering, pp. 201–213 (2002)
10. Botsch, M., Kobbelt, L.: A robust procedure to eliminate degenerate faces from triangle meshes. In: Vision, Modeling and Visualization, pp. 283–290 (2001)
11. Botsch, M., Pauly, M., Kobbelt, L., Alliez, P., Levy, B., Bischoff, S., Roessl, C.: Geometric modeling based on polygonal meshes. In: SIGGRAPH Course Notes (2007)
12. Branch, J., Prieto, F., Boulanger, P.: Automatic hole-filling of triangular meshes using local radial basis function. In: Proceedings of 3DPVT'06, pp. 727–734 (2006)
13. Cignoni, P., Rocchini, C., Scopigno, R.: Metro: measuring error on simplified surfaces. Comput. Graph. Forum **17**(2), 167–174 (1998)
14. Davis, J., Marschner, S., Garr, M., Levoy, M.: Filling holes in complex surfaces using volumetric diffusion. In: Int. Symposium on 3D Data Processing, Visualization, Transmission, pp. 428–438 (2002)
15. Dey, T.K., Edelsbrunner, H., Guha, S., Nekhayev, D.: Topology preserving edge contraction. Publ. Inst. Math. (Beograd) **20**(80), 23–45 (1999)
16. Floriani, L.D., Morando, F., Puppo, E.: Representation of non-manifold objects through decomposition into nearly manifold parts. In: ACM Solid Modeling, pp. 304–309 (2003)
17. Geomagic, Inc. Geomagic Studio. http://www.geomagic.com/
18. Gottschalk, S., Lin, M.C., Manocha, D.: Obbtree: A hierarchical structure for rapid interference detection. In: ACM Siggraph Proceedings, pp. 171–180 (1996)
19. Guéziec, A., Taubin, G., Lazarus, F., Horn, B.: Cutting and stitching: Converting sets of polygons to manifold surfaces. IEEE Trans. Vis. Comput. Graph. **7**(2), 136–151 (2001)
20. Guibas, L., Salesin, D., Stolfi, J.: Epsilon geometry: building robust algorithms from imprecise computations. In: ACM Symposium on Computational Geometry, pp. 208–217 (1989)
21. Guskov, I., Wood, Z.: Topological noise removal. In: Proceedings of Graphics Interface, pp. 19–26 (2001)
22. InnovMetric Software, Inc. Polyworks. http://www.innovmetric.com/
23. INUS Technology, Inc. Rapidform. http://www.rapidform.com/
24. Ju, T.: Robust repair of polygonal models. ACM Trans. Graph. **23**(3), 888–895 (2004)
25. Liepa, P.: Filling holes in meshes. In: Eurographics Symposium on Geometry Processing, pp. 200–205 (2003)
26. Murali, T., Funkhouser, T.: Consistent solid and boundary representations from arbitrary polygonal data. In: Proceedings of Symposium on Interactive 3D Graphics, pp. 155–162 (1997)
27. Nooruddin, F., Turk, G.: Simplification and repair of polygonal models using volumetric techniques. ACM Trans. Vis. Comput. Graph. **9**(2), 191–205 (2003)
28. Podolak, J., Rusinkiewicz, S.: Atomic volumes for mesh completion. In: Eurographics Symposium on Geometry Processing, pp. 33–42 (2005)
29. Rocchini, C., Cignoni, P., Ganovelli, F., Montani, C., Pingi, P., Scopigno, R.: The marching intersections algorithm for merging range images. Vis. Comput. **20**(2–3), 149–164 (2004)
30. Rossignac, J., Cardoze, D.: Matchmaker: manifold breps for non-manifold r-sets. In: Proc. of 5th ACM Symposium on Solid Modeling and Applications, pp. 31–41 (1999)
31. Rourke, C.P., Sanderson, B.J.: Introduction to Piecewise Linear Topology. Springer, Berlin (1972)
32. Samet, H.: Spatial Data Structures: Quadtrees, Octrees and Other Hierarchical Methods. Addison Wesley, Reading (1989)
33. Shewchuk, J.R.: Delaunay refinement algorithms for triangular mesh generation. Comput. Geom. Theory Appl. **22**(1–3), 21–74 (2002)
34. Si, H., Gaertner, K.: Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In: Proceedings of the 14th International Meshing Roundtable, pp. 147–163 (2005)



Marco Attene is a researcher at IMATI-GE/CNR, where he investigates new directions, paradigms and algorithms for 3D geometric modelling, processing and analysis. Marco holds a PhD in Electronic and Computer Engineering and a Research Management diploma.
He has been an invited PhD student at the University of Otago (NZ) and an invited researcher at INRIA (F). Marco contributed to the conception and implementation of successful international projects, and within the EU FP6 AIM@SHAPE NoE he coordinated a team of 20 experts for the definition of metadata to describe digital shapes, currently at the basis of the popular AIM@SHAPE Repository. He promoted key collaborations and joint research programs with world-leading groups in the area. He regularly serves as program committee member for international conferences, and has been member of the organizing board of SMI'01 (IEEE Shape Modelling Int. Conference) and SAMT'07 (Int. Conf. on Semantic and Digital Media Technology 2007). Marco has been organizer of the "Stability on watertight models" track of the SHREC 2008 international contest on 3D shape retrieval.