STAG: Smart Tools & Apps for Graphics (2015)
Silvia Biasotti and Marco Tarini and Andrea Giachetti (Editors)

RameshCleaner: conservative fixing of triangular meshes

Marco Centin and Alberto Signoroni

Information Engineering Dept. DII, University of Brescia, Italy

Abstract

In this work, after a careful examination of the most common errors and flaws which typically occur in meshes produced by 3D scanning processes, we propose a set of fixing tools which solve effectively several important mesh defects while preserving the original data. The proposed tools are then organized and activated in the RameshCleaner pipeline allowing the user to take advantage of a semi-automated fixing solution, optimized for speed and efficiency, as well as of the possibility to selectively activate individual tools. The comparison, over a set of representative scanned models, with free and commercial semi-automated fixing solutions gives a significant evidence of the defect abatement and computational speed characteristics of the proposed system.

1. Introduction

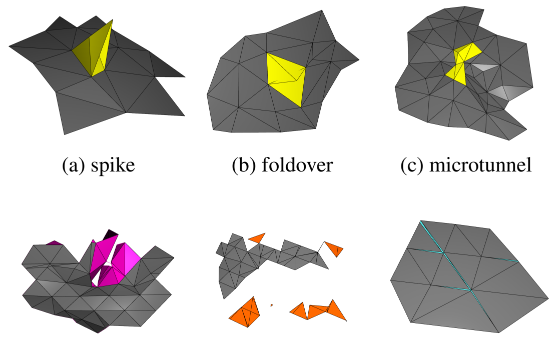
3D mesh models coming from devices and processes dedicated to the geometric digitization of real objects are typically originated from triangulation techniques that work on sets of overlapping range images (or derived point clouds). Because of several factors related to both the physics and to the geometry of the acquisition, meshes derived from point-based scan data are likely to contain a large number of flaws that, whether they are visually impacting or hardly visible, make it difficult or unstable the job of subsequent processing and editing stages which are needed, accordingly to the specific application context. A certain number of works have dealt with these problems. We refer the reader to two recent surveys which have been given by Ju [Ju09] and Attene, Campen and Kobbelt [ACK13]. The methods of mesh fixing may be divided into two categories: the volumetric methods that try to reconstruct volumetrically the object (and often imply a complete remeshing), and surface methods, that try to solve the degeneracies from the mesh topology and connectivity. In [Att10], Attene presents a surface-based fixing strategy and a software tool that try to preserve the original data while adopting effective local refinements. However, the presented heuristics assume that the object is closed and accessible from a single connected component. When the input object comes broken into multiple components, the method tends to fail, possibly erasing relevant connected components and closing large holes in an unnatural way. What we propose here is a structured set of effective fixing strategies that maximally preserve the original data while effectively solving several important mesh weaknesses. Our method is also able to reliably handle the fixing of the edges. This improves, with respect to other approaches, the possibility and the quality of closing holes below a certain size, while blunting and cleaning from large degeneracies the mesh borders.

The paper is organized as follows: we initially identify and analyze the most common problems which can arise on triangular meshes (Sec. 2), then we develop a number of effective solutions able to detect and solve most of degeneracies with low expenditure of time, high fixing success rate, flexibility and ease of use, minimal invasiveness and a high degree of preservation of the original meshing (Sec. 3). A (semi-)automated pipeline, named RameshCleaner, allows the user to take advantage of an automatic fixing solution, optimized for speed and efficiency, as well as of the possibility to selectively activate individual steps (Sec.4). Comparisons are proposed with respect to free and commercial reference solutions to exemplify and demonstrate the effectiveness of the proposed method (Sec. 5).

2. Typical problems in triangular meshes

We assume that we have in input a manifold triangular mesh with assigned connectivity. More precisely we require that there are no *complex vertices* (vertices whose neighborhood is not a topological disk) or *complex edges* (edges having more than two incident faces). Most scanning systems generate mesh with these requirements. We internally represent the mesh connectivity and the additional properties using the the halfedge-based data structure OpenMesh [BSBK02], that doesn't support the representation of complex edges by de-

DOI: 10.2312/stag.20151300



(a) spike (b) foldover (c) microtunnel

(d) boundary (e) components (f) near-degenerate

Figure 1: A showcase of some common issues fixed by the proposed method: spikes (a), fold overs (b), micro tunnels (c), complex boundaries (d), small connected components (e) and near-degenerate faces (f).

fault (see. [BSBK02]). In particular, the proposed method is not suitable for converting a non-manifold mesh into a manifold one or for rebuilding a valid mesh connectivity from a triangle soup (like it has been done, for example, in [GTLH98]).

In the following we list some of the most relevant problems that can be found in meshes acquired by modern 3D scanning systems, also depicted in Figure 1. These problems can also occur in a combined way in proximity of important features of the mesh, so it is important to adopt solutions that preserve as much as possible the mesh features.

Isolated and degenerate vertices. We consider as *degenerate* every vertex of the mesh that has no adjacent faces (isolated vertex) or such that the corresponding 3D coordinates are degenerate (having infinite or non-numeric values). Their presence may impact the performances of other algorithms by altering the size of the bounding box or other global statistics and causing ill-posed computing.

Spikes. We call *spike* any small protrusion of few triangles on a relatively-flat portion of the surface having the topology of a disk. An example of spike is shown in Fig.1a. Spikes are often the result of a triangulation of sparse outliers in proximity of the surface and can be identified by looking at edges having very high dihedral angles.

Foldovers. A *foldover* is the result of a wrong triangulation of points that are in proximity of the surface. As shown in Fig.1b, these problems consist in few vertices that are bent below (or above) the surface, having few triangles attached to them. These mesh parts are often manifold and non self-intersecting and causes wrong geometrical computations on the surrounding surface. Unfolding these clusters have a beneficial effect on the stability of many algorithms that depend on the local geometry. Foldovers are often generated by surface reconstruction techniques, but can be also produced by naive editing techniques that modify the mesh connectivity without implementing valid geometrical checks.

Micro tunnels. Similarly to foldovers, *microtunnels* are also the result of a wrong triangulation of noise in proximity of the surface. However, they differ from foldovers in having a non-trivial local topology, that is, there exist a geodesic neighborhood having a non-disk topology. We put these defects into a separate class because they require more involved fixing procedure. Some work explicitly considered ad-hoc solutions for detecting and fixing these problems (see [GW01]). The heuristic defined in Section 4 is capable of removing some of these problems if they can be identified as a cluster of few triangles (see Fig.1c).

Complex boundaries. A certain number of degeneracies can be present along the mesh boundaries, including small protrusions of few faces leading to boundary curves having a complex jagged shape and flipped faces (see Fig.1d). Fixing these issues before applying any hole-filling procedure has a drastic effect on the final result (as will be shown in Section 5). In Section 3.3 we describe how to remove these problems and smooth the boundary curves.

Small components. Small islands and connected components are often considered a problem. They can be the result of triangulation of nosy clusters of points or obtained by other techniques that involves deletion of some mesh parts. Our fixing method will remove these components since they generate small boundaries that cannot be easily merged with the rest of the mesh. Fig.1e show an example of small components along a boundary of the mesh.

Near-degenerate facets. We call *near-degenerate facet* every face of the mesh such that the corresponding triangle is almost degenerate, that is, its area or minimum angle are below a certain threshold. In Section 3.6 we extend this definition to every triangle having edges that are too short with respect to a locally-defined average edge length. The presence of near-degenerate triangles causes degenerate computations in estimating the face normals and leads to numerical instability in most mesh processing techniques. Removing these elements has a beneficial effect for every subsequent processing step and also increases the efficiency of the mesh representation. Near-degenerate facets can be generated by Marching Cubes contouring methods or by other processing techniques (e.g. decimation). An example of these elements is shown in Fig.1f (cyan color facets).

Self-intersections. The self intersecting parts of the mesh are obtained by the detection of the faces such that the corresponding triangles intersect with some other triangle of the mesh. Spatial sorting structures or parallelized computations are often used in oder to reduce the time-complexity of brute-force intersection tests (computed, for example, using the method of Möller [Möl97]). This test however remain quite expensive for large and dense meshes, while merely revealing the self intersecting triangles does not give additional information on how to fix them. Practically speaking, we distinguish between two types of self intersections (as it is schematically drawn in Figure 2). The *local self intersection* (Fig.2(a)) are clusters of self-intersecting triangles that are connectivity-wise close (that is, triangles intersect with faces that are in a certain small ring). This kind of self intersections are frequent in mesh obtained by scanning and reconstructing real world objects and they often correspond to a bad triangulation of outliers. Non-local self intersections (Fig.2(b)) are more commonly occurring in meshes that have been processed by CAD software or some algorithm that deforms the surface or merge different meshes. Therefore, we will adopt a simpler lightweight strategy by indirectly detecting clusters of the first type through their local geometry (spikes and highly creased edges), instead of identifying them as self-intersections using computationally expensive routines.

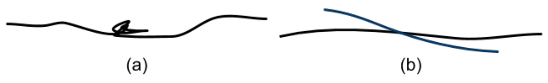


Figure 2: Local vs non-local self intersections.

3. Fast and effective fixing methods

In this section we develop a set of fast and effective tools for fixing local geometrical and topological errors. We rely on detecting mesh problems by looking at the regularity of some local geometrical statistics. We first describe how to compute and keep updated all the necessary properties (Sec.3.1) and then how to use them in order to fix the mesh problems. In Sec.3.2 we implement a fast detection of small connected components, in Sec.3.3 we develop an effective routine that can be used in order to remove degeneracies and smooth the mesh boundaries. An improved hole-filling method is proposed in Sec.3.4 and a method for automatically resolving failure cases is explained in Sec.3.5. In Sec.3.6 we explain how to identify degenerate triangles and fix them through sequences of safe edge collapses. Then, in Sec.3.7, we analyze the resolution of spiked vertices by implementing a local relaxation routine. In Sec.3.8 we eventually define an heuristic that can be used in order to fix spiked vertices that cannot be solved by local relaxation.

3.1. Computing local geometrical properties

The first step of our fixing routines is to precompute some geometrical statistics that allow us to quickly detect degenerate elements and implement fast geometrical checks. In our implementation we rely on the OpenMesh data structure [BSBK02] that allows the assignment of additional properties to every element of the mesh (vertices, halfedges, edges and faces). More precisely: a) we associate to every halfedge the angle of the opposite corner in its face (if existent); b) every inner angle of any triangle is then associated with some halfedge of the mesh; c) an edge is associated with its length and with the dihedral angle formed by the two adjacent faces (if existent); d) a face is associated with its area, its barycenter and the minimum inner angle; e) a vertex is associated with its valence, the average of the length of the adjacent edges and the maximum of the absolute values of the dihedral angles of those edges. In addition to the above properties an integer status is added in order to mark elements that are not yet computed or degenerate computations. Our fixing strategy requires that all the above stats are computed and kept updated in any operation for every involved element of the mesh. This goal is achieved by implementing routines both for the global and local update of the above statistics.

Global update. We efficiently compute all the statistics in three steps. A first cycle run over the (non-deleted) halfedges. The edge vector is computed. The edge length is collected from the edge stats (or computed if non-existent). If the edge is not boundary, the opposite vertex exists. We then compute the other edge vector, compute or retrieve its length, and compute the face area. The face area is stored in the face statistics. From the two normalized edge vectors, the inner angle is also computed and stored in the halfedge statistics. If the edge is not boundary, then we have two adjacent faces. The corresponding dihedral angle is then computed and stored in the edge properties. A second cycle run over the mesh facets. It assumes that the edge and halfedge properties are already computed, together with the face areas. By iterating over the face halfedges the minimum angle and the face centroid are computed and stored in the face properties. Finally, a third cycle over the mesh vertices collects local statistics by looking at the outgoing halfedges. By computing the properties in the above order it is possible to recycle expensive computations and have the geometrical statistics available in constant time.

Local updates. After each elementary editing operation on the mesh (flip, split, collapse) it is necessary to update the local statistics accordingly or computing them for the new generated elements. We therefore implemented optimized local update routines for the 1-ring of a vertex or a face.

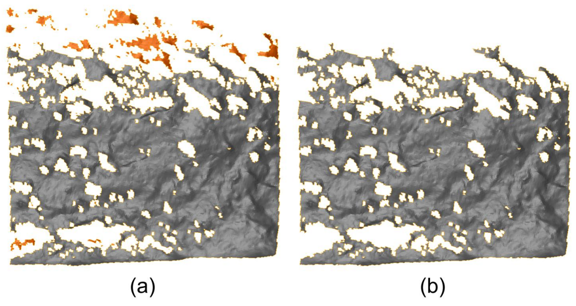


Figure 3: Example of boundary with many small connected components (a) that are removed by our procedure (b).

3.2. Removing small connected components

The sparsity of the data samples (caused by insufficient acquisition) often leads to small isolated surface patches (Figure 3 shows an example taken from the the Awakening model of the Stanford repository). In our fixing routine we remove these small components if their dimension is less than a user-defined threshold, since they increase the complexity of hole filling and may lead to self intersections or wrong hole patches. These components are computed using a queue-based region-growing procedure that iteratively collects adjacent faces up to a certain size.

3.3. Boundary cleaning and smoothing

One main component of our mesh fixing tool is a boundary cleaning method, designed for the following purposes:

- Removing flipped boundary faces that introduces illness in every subsequent hole-filling procedure;
- Cleaning small isolated components from the boundary whose closure is often ambiguous;
- Regularizing and smoothing the boundary curves both for improving the aspect of the open boundaries and simplifying closure of small holes.

The first goal is achieved by adopting a procedure that iteratively removes vertices from the boundary. A *bad boundary vertex* is a (non-deleted) boundary vertex such that at least one of the following conditions is satisfied: a) the valence of the vertex is equal to 2; b) there exist an adjacent edge having dihedral angle above a certain user-defined threshold (in all the examples of this paper we used 120 degrees); c) the number of adjacent boundary edges is greater than 2.

The cleaning phase proceeds as follows. Given a boundary of the mesh, i.e. a sequence of boundary vertices pairwise connected by boundary edges, we initially determine a *confidence region* by computing the n-ring of the boundary vertices (in all the shown examples we set $n = 4$). Then we iterate over these vertices and progressively remove bad boundary vertices from the mesh. The algorithm terminates when one of the following conditions is satisfied:

- The number of non-deleted bad boundary vertices contained in the confidence region is 0;
- A maximum number of iterations is reached (in all our examples the maximum number of iterations was 30);

Figure 4(a) shows an example of mesh with many bad boundary vertices and 4(b) the result of the iterative procedure defined above. The deletion of vertices from the mesh potentially generate new isolated components of the mesh. We therefore compute the face-connected components of the confidence region and remove the small ones (in all the examples we removed the components whose size is inferior to 40 faces. Finally we apply a modified Laplacian relaxation to the 1-ring of the (updated) boundary vertices. In order to avoid boundary shrinkage, we restrict the contributes of the Laplacian along the boundary curves as follows. If v is a non-boundary vertex, then we denote the $\mathcal{N}(v)$ the 1-ring

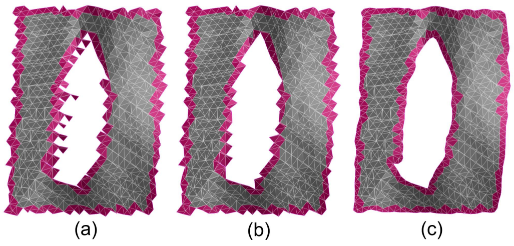


Figure 4: Example of boundary cleaning. The jagged boundaries of the input mesh (a) are first cleaned by our iterative procedure (b). The vertex 1-ring of the boundaries is then smoothed in order to obtain a pleasant shape.

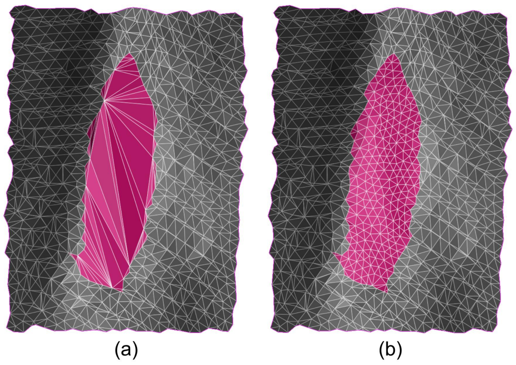


Figure 5: Example of hole filling. A triangulation of the initial boundary loop is computed (a). The hole patch is then remeshed in order to obtain a semi-regular hole patch (b).

neighborhood of v, otherwise $\mathcal{N}(v)$ is defined as the 1-ring of boundary vertices only. If $p(v)$ denotes the 3D point associated with the vertex v, the uniform *restricted Laplacian* operator is defined by the following formula:

$$L(v) = \frac{1}{|\mathcal{N}(v)|} \sum_{w \in \mathcal{N}(v)} p(w) - p(v) \ . \qquad (1)$$

After collecting the 1-ring of the boundary vertices we apply few smoothing iterations using the above Laplacian operator (by default, 10 iterations with timestep of 0.02). Figure 4(c) shows the result of this step applied to the cleaned boundaries of Figure 4(b).

3.4. Filling small holes

We adopt a terminological distinction between *hole filling*, where the model have many small holes that can be closed by simply triangulating the corresponding boundary (e.g. [BS95] and [Lie03]), and *mesh completion* that aims to complete bigger missing parts of the model by adopting volumetric techniques (e.g. [PR05] and [CPS15]) or by considering contextual information (e.g. [SACO04], [HTG14]). In our mesh fixing pipeline we include some routines that are suitable for filling small simple holes enclosed by a single

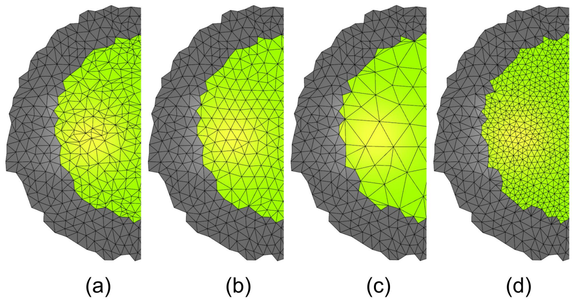


Figure 6: With respect to [Lie03] (a), our solution produces better patches (b) and allows to configure the triangle density in order to obtain coarser (c) or denser (d) patches.

boundary loop. The proposed technique is a variant of the well-known method of Liepa [Lie03], where the remeshing part is replaced by the fast and effective method of Botsch et al. [BK04]. The procedure is composed of two steps, represented in Figure 5 and described below.

Loop triangulation. Given a boundary loop (i.e. a closed boundary curve), we want to compute an initial (possibly non self-intersecting) minimum-weight triangulation of the corresponding 3D polygon. A popular approach to solve this problem is to adopt a dynamic programming technique that was developed (in the planar case) independently by Klincsek [Kli80] and Gilbert [Gil79]. This technique was adapted for the 3D case by computing area-minimizing triangulations by Barequet and Sharir [BS95] and then extended by Liepa in order to reduce self-intersections in presence of crenellations [Lie03]. Our mesh fixing tool implements by default the method of Liepa for computing the initial triangulation of holes whose boundary loop is below a certain user-defined size. The result of the initial triangulation of the cleaned boundary of Fig.4(c) is shown in Fig.5(a).

Hole patch remeshing. After an initial weight-minimizing triangulation of the boundary loop is computed, a refinement is necessary in order to add internal vertices and improve the quality of the hole patch. Liepa [Lie03] proposes a refinement strategy that propagates a local sampling density across the hole (see Fig.6(a)). This propagation combined with the irregularity of the initial triangulation often leads to artifacts or patterns of denser triangles across the hole. We therefore replace this refinement strategy with the uniform remeshing method proposed by Botsch et al. [BK04] which produces well shaped triangulations (see Figg.5(b) and 6(b)) and allows to configure the triangle density (Figg.6(c)-(d)).

3.5. Handling incorrect hole fills

In some special cases, the definition of the boundary loops that identify the small holes is ill-posed. Fig.7(a) is an example of a broken triangulation that can be classified as a single hole but is formed by many conflicting parts (that are not even self-intersecting or spiked). An initial triangulation generated with the method explained in Section 3.4 would generate complex edges, that are disallowed by OpenMesh [BSBK02]. We detect these cases by checking the validity of the corresponding handles and we treat them as *aborted fills*. We then attach to these events a special cleaning that enlarges these *bad holes* by removing their vertex 1-ring and then fills the cleaned hole, as shown in Fig.7.

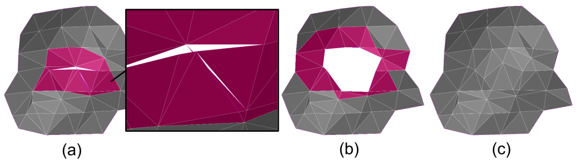


Figure 7: Example of hole whose closure generates complex edges (a). The aborted fill is handled by cleaning the 1-ring of the boundary (b), and filling the new hole (c).

3.6. Fixing near-degenerate faces

Near degenerate triangles are repaired through an iterative procedure that evaluates reliable edge collapses. We initially run over the mesh edges and check the local statistics. Let l_g be the *global average edge length* and l_e be the *local average length* (computed by averaging the edge lengths stored in the vertices of the edge). The edge e is said to be *collapsible* if at least one of the following conditions is satisfied:

1. **Zero edge**: the edge length is below a (very small) global threshold, defined as $\varepsilon \cdot l_g$, where ε is a parameter (0.1 by default in all our examples);
2. **Skinny edge**: the edge length is below a local threshold, defined as $\delta \cdot l_e$, where δ is a parameter (0.25 by default) and at least one of the adjacent faces have opposite angle inferior to a certain threshold (10 degrees by default).

Note that the definition of *collapsible* edge is not including any topological or geometrical check of the correctness of

Algorithm 1 Fixing near-degenerate faces

```
1: function FIXNEARDEGENERATEFACES($M$)
2:     reset edge timestamps to 0;
3:     $\mathcal{Q} \leftarrow$ priority queue of quadruplets $(\alpha, he, p, t)$;
4:     initialize $\mathcal{Q}$ by inserting collapsible edges;
5:     while $\mathcal{Q} \neq \emptyset$ do
6:         pop front element $(\alpha, he, p, t)$ from $\mathcal{Q}$;
7:         if( $t \neq$ timeStamp$(e)$ ){ skip }
8:         collapse $he$ onto $p$, update neighbor timestamps;
9:         update all statistics in the neighborhood of $he$;
10:        detect new neighboring collapsible edges;
11:        compute optimal halfedge $he$ and destination $p$
12:        push $(\alpha, he, p, t)$ in queue
13:    end while
14: end function
```

the actual collapse. Once these edges are identified, a suitable collapse direction and destination point is computed by testing the correctness of different solutions (respectively, collapse on most featured vertex, collapse to midpoint and collapse to less featured). The *correctness* of a collapse of the halfedge *he* onto the destination point p is verified by simultaneously checking two conditions:

1. **Connectivity check**: is done by testing if, on each side of the collapsed edge, only one pair of edges are joined;
2. **Geometrical check**: the normals of the faces in the 1-ring neighborhood of the extremal vertices that persist after the collapse operations are tested before and after a simulated collapse in order to verify if the angle deviation is below a certain threshold (45 degrees by default);

After a proper collapse direction and destination is determined, the edge is inserted into a priority queue with a priority given by the inverse of its length ($\alpha = 1/(l+\varepsilon)$) so that short edges have higher collapse priority. Algorithm 1 summarizes the fixing procedure. After an edge is collapsed, the local statistics of the neighborhood are updated and new collapsible edges are inserted in the priority queue. By updating an edge time-stamp property in the edge neighborhood it is possible to disable adjacent collapses that were invalidated in the process and re-classify the neighborhood. In Section 5 we will show examples of fixing near-degenerate faces occurring in popular datasets.

3.7. Relaxing spiked vertices

With reference to the notations of Section 2, we now consider procedures that can fix mesh foldovers and spikes. In our procedure, a *spiked vertex* is any vertex having at least one edge whose dihedral angle is greater (in absolute value) than a user-specified *dihedral angle threshold*. Since we already computed all the geometrical properties, we can rapidly detect these vertices by looking at the stored values. If the local connectivity have the topology of a disk, we found that a simple Laplacian relaxation of the surrounding vertices rapidly unfolds few vertices that are bent off the surface. We define a modified Laplacian smoothing whose speed is controlled by the maximum absolute dihedral angle of the vertices. The fundamental idea is to quickly relax spiked elements while reducing the motion of neighbor areas that are encoding a correct geometry. The process is applied only on the vertices that have absolute dihedral angle above a certain threshold (and their 1-ring). The *relaxing Laplacian* vector for a vertex v is computed with the formula

$$R(v) = \begin{cases} 0 & \theta(v) \leq a \\ \big(\theta(v) - a/(b-a)\big)L(v) & \theta(v) > a \end{cases}, \quad (2)$$

where $\theta(v)$ is the maximum absolute dihedral angle of the edges adjacent to vertex v, $L(v)$ is the Laplacian defined by (1) and a, b are used-defined parameters denoting respectively the minimum dihedral angle that will be fixed in the smoothing and the threshold level at which the vertices will be uniformly smoothed. This smoothing routine, repeated

Algorithm 2 Relax spiked vertices

```
1:  function RELAXSPIKEDVERTICES($M$)
2:      $S \leftarrow$ spiked vertices in $M$
3:      $W \leftarrow$ working region ($n$-ring) of $S$
4:      $i \leftarrow 0$
5:      while $S \neq \emptyset$ and $i <$ maxIterations do
6:          compute 1-ring $R$ of $S$
7:          relax $R$ by applying (2)
8:          update geometrical properties over $R$
9:          $S \leftarrow$ spiked vertices in $W$
10:         $i \leftarrow i+1$
11:     end while
12: end function
```

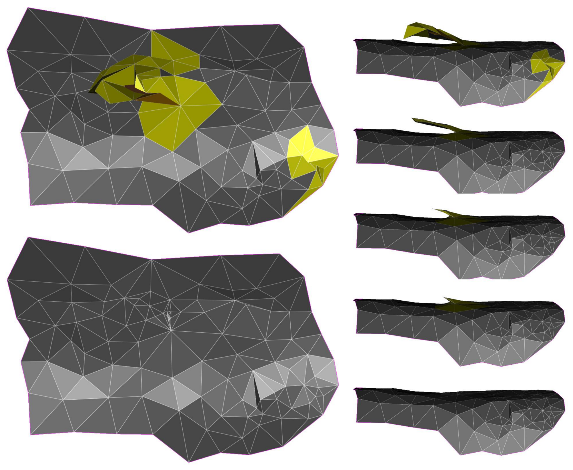


Figure 8: Removing spikes and highly creased edges by relaxing the vertex positions using the operator defined in equation (2). The small protrusion (top part) is gradually absorbed and eliminated (bottom part).

few times, introduces a tension on the surface that tends to resolve many simple degenerate configurations while preserving the neighboring mesh geometry. Figure 8 shows an example of applying this operator on a protrusion of the mesh. The right part of the picture shows that the evolution of the surface tends to absorb these small protrusions while preserving the surrounding geometry. Algorithm 2 summarizes the described relaxation procedure.

3.8. Fixing unrelaxable spiked vertices

Not all the spiked vertices can be fixed by a local relaxation. In particular, if the local connectivity is not disk-shaped, then the operator defined in Sec.3.7 is unable to unfold the edges with high dihedral angles. An example is shown in Fig.9(a) where a micro-tunnel is considered. The relaxation of these degenerate configurations results in applying a tensity that tends to emphasize the problem. In our procedure we define *unrelaxable spiked vertices* the non-deleted vertices of the mesh that cannot be relaxed by few applications of the operator described in Sec.3.7 (including microtunnels or severe foldovers). These vertices are removed, as shown in Fig.9(b): the boundaries generated in the process are cleaned and the small holes are filled, as shown in Fig.9(c). Holes generated by the deletion of unrelaxable spiked vertices are handled by the fixing routine explained in Sec.3.4.

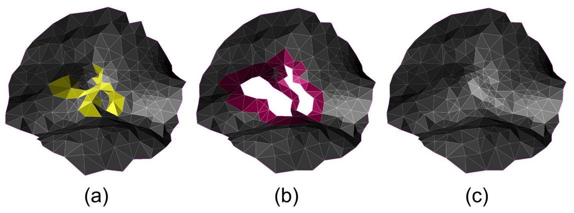


Figure 9: Microtunnels of severe foldovers that cannot be unfolded using a local relaxation (a) are removed and cleaned (b). Small holes eventually generated are then filled (c).

4. Automatic mesh repair pipeline

We designed and implemented a semi-automatic fixing pipeline that makes use of the solutions developed in Sec.3.

Preprocessing. The mesh is scanned for degenerate vertices and these are removed. We do this operation in a preprocessing step since the procedures described in Section 3 do not generate new degenerate vertices. The geometrical properties are added and computed as explained in Sec.3.1.

Fixing pipeline. A single iteration of the automatic fixing tool consists in the following steps. The execution order of the operations is designed to avoid the repetition of multiple cleaning operations. For example, operations involving deletion of elements of the mesh (that potentially generates new holes or components) are done before removing small components and filling holes.

1. **Relax spiked vertices**: the method of 3.7 is applied in order to eliminate spiked vertices that are relaxable. The properties in the 1-ring of these vertices are updated;
2. **Remove unrelaxable spikes**: the unrelaxable spiked vertices are collected and deleted. The properties in the 1-ring of these vertices are updated;
3. **Clean the boundaries**: the cleaning procedure described in Section 3.3 is applied. The stats in the vertex 1-ring of the new mesh boundaries are updated;
4. **Remove small components**: small face-connected components of the mesh are computed as described in Section 3.2 and deleted from the mesh. No update for the local statistics is necessary after this step;
5. **Fill small holes** The boundary loops having a number of vertices below a user-specified threshold are filled using the method described in Section 3.4. Aborted fills are handled with the enlargement procedure as described in Section 3.5. All the local geometrical properties corresponding to the new faces are computed;
6. **Fix near-degenerate faces**: based on the updated local statistics, the list of *collapsible* edges is collected and the iterative fixing routine of Section 3.6 is applied. All the stats are left updated by the method;
7. **Relax spiked vertices**: the relaxation step of Section 3.7 is applied again in order to eventually fix new highly creased edges generated by the above steps. The local stats are updated after this process.

Postprocessing. All the additional properties are removed and a compact representation of the mesh is obtained by removing the elements marked as deleted (garbage collection).

User parameters. The described processing pipeline allows the user to select few intuitive parameters for the fixing routine. Every step of the fixing pipeline can be executed separately or disabled. The spiked vertices are determined through a threshold on the dihedral angles (120 degrees by default). The hole filling procedure requires a maximum hole size, given as the number of vertices of the boundary loop.

5. Examples, experiments and results

In this section, results and comparisons on meshes acquired with 3D scanners are illustrated. We selected some representative data from the Stanford 3D Scanning Repository[1] (Happy Buddha, Awakening), from the Stanford Digital Michelangelo Project[2] (David) and a dataset acquired using the 3D scanner Scan-in-a-Box[3] (Hulk). We provide experimental comparisons with respect to the MeshFix[4] tool by Marco Attene [Att10] and the Mesh Doctor tool (here referred as MeshDr) included in the commercial software Geomagic Studio® 2014[5] for Windows x64. The open source software MeshLab[6] also includes a number of cleaning and repairing routines that can be applied on a mesh (e.g. removing small components, isolated vertices, merging closed vertices, etc.). It could be however a non-trivial task for the user to identify which problems might be present in the input and how to properly combine these tools for finalizing the input. Since in this work we are not considering user behaviour, we limited our comparisons to tools that can detect and repair mesh problems in a guided and semi-automated way. The proposed method was implemented in C++11 using the OpenMesh [BSBK02] library. The timings provided refer to a version of the software compiled for Windows x64 using Visual Studio 2012 and executed on a PC equipped with an Intel® Core™ i7-4790 CPU 4.00 GHz and 16 GB of RAM. The timings provided in the tables of this paper refer to the total user time, which includes loading the input mesh, initializing the tool, applying one or multiple fixing

[1] http://graphics.stanford.edu/data/3Dscanrep/
[2] http://graphics.stanford.edu/data/mich/
[3] http://www.scaninabox.com/
[4] http://sourceforge.net/projects/meshfix/
[5] http://www.geomagic.com/
[6] http://meshlab.sourceforge.net/

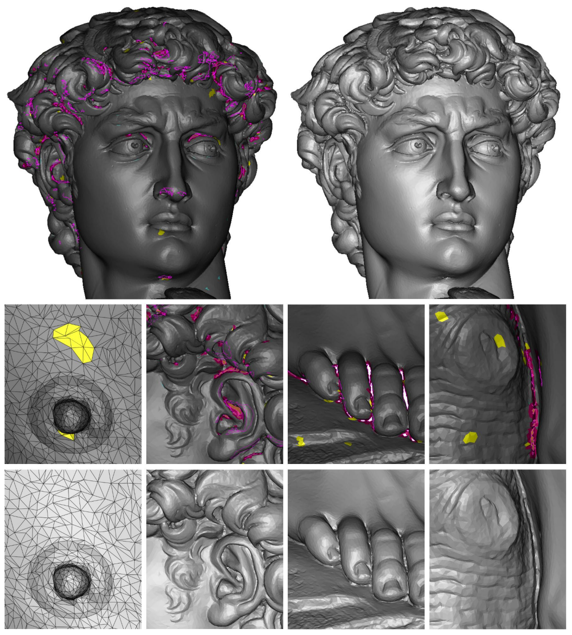


Figure 10: Fixing the David model.

	Input	MeshFix	MeshDr	Ramesh
isolated vertices	5	0	6440	**0**
components	259	0	0	**0**
small holes	1543	0	0	**0**
near-degenerate	571	1318	856	**0**
spiked vertices	2874	4711	235	**0**
intersections	541	0	0	**0**
microtunnels	1	3	0	**0**
time	-	239s	43s	**5s**

Table 1: Results for the **David** model (500k vertices, 1M faces). Components below 400 faces and holes below 100 vertices are considered small. The dihedral angle threshold for the spiked vertices was set to 110 degrees.

iterations and saving the output mesh. The speed gap of the proposed method with respect to MeshFix and MeshDr is explained by the fact that our method is focused in resolving geometrical errors in a local way and is not relying on a global self-intersection test. Also, the performance of MeshFix are also affected by a pre-processing step that attempt to resolve eventual non-manifold parts of the mesh, while our tool assumes a manifold input. We evaluate the quality of the output both visually and quantitatively by comparing the number of problems detected before and after fixing.

The tables of this paper indicate a list of problems that were be detected with our tool (before and after the application of the various tools). Only the number of self-intersections and the small tunnels were detected using MeshDr, since our tool do not include an explicit detection of those problems. In the following we show how the simple lightweight strategies described in the previous sections can effectively resolve most of the mesh problems, including flaws that were not explicitly addressed (such as self intersections and microtunnels), with a significant boost in the performances with respect to other popular solutions.

David. In Fig.10 and Tab.1 we consider the (decimated) David model from the Stanford repository. The mesh has 500k vertices and 1M faces. We finalize this mesh by applying 3 iterations of the fixing routine explained in Sec.4 (total processing time 5s). By comparing the details in Fig.10 before and after fixing, it is possible to appreciate how foldovers are quickly resolved and the combination of boundary cleaning and improved hole filling effectively finalizes the model with full preservation of fine details and minimal invasiveness. In particular, note the complex holes in the hair of the model are recovered by the combination of our boundary cleaning routine with the improved hole filling method. The 541 self intersections of the input model can easily be identified as spiked vertices and locally resolved by our tool. In comparison, MeshFix does not resolve spiked vertices, foldovers and near-degenerate faces and automatically fills every hole (including a wide hole in the bottom part). Although most applications require working with watertight models, automatically filling large holes using a basic surface-based method drastically increases the computational cost (see Table 1) and often defines inappropriate patches. In contrast, the proposed method will only polish the mesh boundary, so that, if needed, more advanced completion techniques could be applied afterwards. Eventually note how in this case, as well as in the following ones, MeshDr produces many isolated vertices in output (probably corresponding to deleted portions of the mesh) that have to be fixed in a separated step.

Happy Buddha. In Fig.11 we considered the Happy Buddha model, which is a good example of mesh having a large number of near-degenerate faces (see Sec.3.6) that cause unstable computation in many algorithms. By removing these elements we obtain a mesh that is more suitable for further processing and it is more compactly represented. The model has 543k vertices, 1M faces and can be fixed by applying 5 iterations of the proposed method (about 11s of total processing time). As before, Table 2 provides a comparison with MeshFix and MeshDr. Note that our tool is the only one that automatically removes near-degenerate faces. The input mesh has 79 small tunnels detected using MeshDr. Although our method does not implement any explicit fixing of topological noise [GW01], the number of tunnels is reduced to 4 in the finalized model. As shown in Fig.11, the remaining 4 larger tunnels are well-sampled on the model and their simplification can be considered out of the scope of a general-purpose fixing routine. The number of tunnels

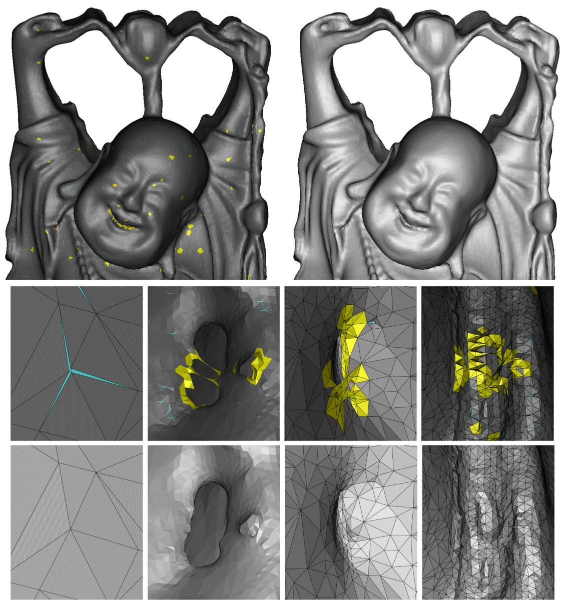


Figure 11: Fixing the Happy Buddha model.

	Input	MeshFix	MeshDr	Ramesh
isolated vertices	0	0	29538	**0**
components	51	0	0	**0**
small holes	56	0	0	**0**
near-degenerate	111815	108681	107589	**51**
spiked vertices	5879	5694	1206	**0**
intersections	499	4	0	**0**
microtunnels	79	69	**0**	4
time	-	59s	174s	**11s**

Table 2: Results for the **Happy Buddha** model (544k vertices, 1M faces). Components below 400 faces and holes below 100 vertices are considered small. The dihedral angle threshold for the spiked vertices was set to 60 degrees.

is not drastically reduced by MeshFix, since most of them cannot be detected as self-intersecting triangles.

Awakening. Fig.12 shows some details of fixing the Awakening model. The input mesh has 2M vertices, 4M faces and a large number of small connected components that correspond to incomplete data (see Fig.3 and Fig.12). As a consequence, the input has a large number of irregular boundaries whose closure is not well-defined. Quantitative results are shown in Table 3. The heuristic implemented in MeshFix, described in [Att10], fails in finalizing the model, since it relies in resolving the errors by iteratively filling holes and removing self intersections. Since both operations are quite expensive when applied to the inconsistent boundaries of this example, the computational cost and the memory required to fulfill the task is too high and the program crashes. MeshDr terminates after several minutes of processing time, but the final result is not satisfactory since the output mesh still present small components, spikes, foldovers and small holes and not all the self intersections are resolved. The lightness of the proposed method allows us to finalize the model in 22s of total processing time. The output have no self intersections, 1 small tunnel and 116 near-degenerate faces that are not collapsed since they do not pass the correctness test explained in Sec.3.6.

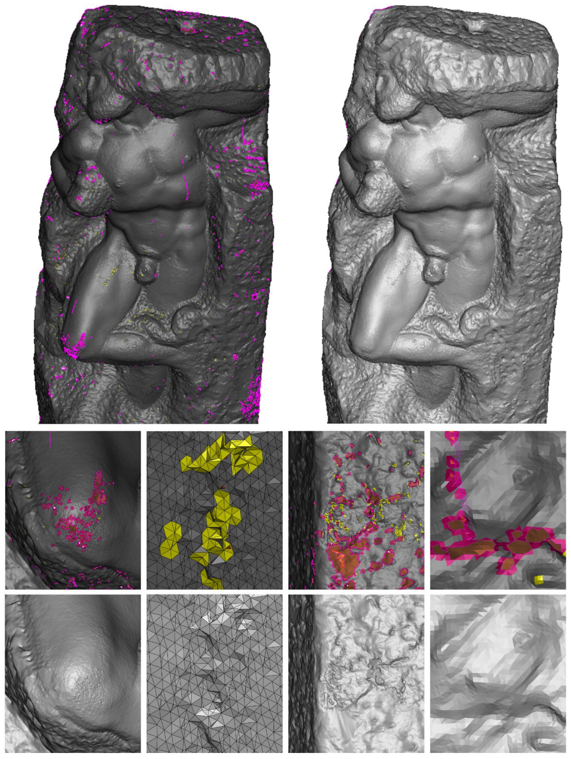


Figure 12: Fixing the Awakening model.

	Input	MeshFix	MeshDr	Ramesh
isolated vertices	116	-	56886	**0**
components	1563	-	2	**0**
small holes	3596	-	16	**0**
near-degenerate	8193	-	7003	**116**
spiked vertices	3065	-	855	**0**
intersections	1361	-	60	**0**
microtunnels	6	-	**0**	1
time	-	-	391s	**22s**

Table 3: Results for the **Awakening** model (2M vertices, 4M faces). Components below 700 faces and holes below 300 vertices are considered small. The dihedral angle threshold for the spiked vertices was set to 70 degrees.

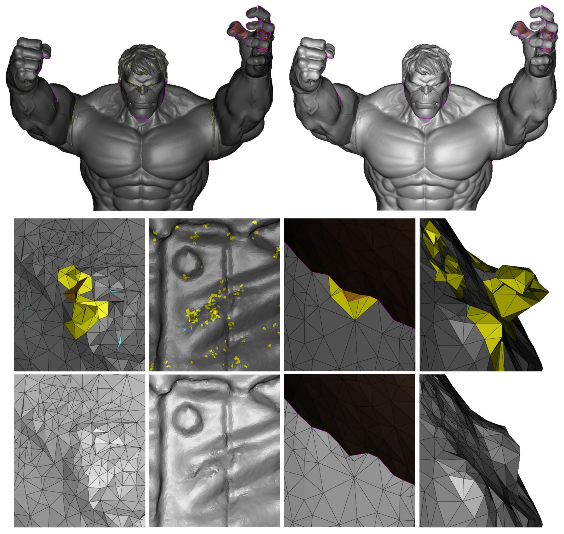


Figure 13: Hulk model, acquired with Scan-in-a-Box.

	Input	MeshFix	MeshDr	Ramesh
isolated vertices	0	0	1470	**0**
components	7	0	0	**0**
small holes	3	0	0	**0**
near-degenerate	7477	7995	6234	**9**
spiked vertices	13251	12605	1174	**0**
intersections	3178	0	0	**0**
microtunnels	7	2	0	**0**
time	-	382s	138s	**11s**

Table 4: Results for the **Hulk** model (1.3M vertices, 2.7M faces). Components below 400 faces and holes below 60 vertices are considered small. The dihedral angle threshold for the spiked vertices was set to 70 degrees.

Hulk. Figure 13 shows some details of fixing a model of a toy acquired using the 3D scanner Scan-in-a-Box. The model has 1.3M vertices and 2.7M faces. Although the overall fidelity is high, the mesh has many small foldovers caused by outliers. By applying few iterations of the proposed method these problems are effectively resolved. As it can be seen in Table 4, all the self intersections and microtunnels are indirectly eliminated with less computational effort. The spiked vertices are not resolved by MeshFix and only partially fixed by MeshDr. Although the model finalized using MeshFix is watertight, big or complex holes (e.g. the left hand of the model) are closed in an inappropriate manner. In comparison, the proposed method only cleans degeneracies along the boundary curves so that these parts can be easily completed using other dedicated tools.

6. Conclusion

In this paper we described the main flaws that can affect triangular meshes, especially when they derive from 3D scan data, and we implemented a lightweight workflow that can effectively finalize the digitized model for other application-oriented tasks. The experimental results show the effectiveness of the proposed method also in comparison to other semi-automated solutions. Further research could investigate convergence heuristics to further increase the automation level, alternate strategies to further reduce the unaddressed problems, methods for adapting the proposed tool for specific applications (e.g. non-manifold meshes, highly irregular meshes, non-triangular meshes, CAD meshes).

References

[ACK13] ATTENE M., CAMPEN M., KOBBELT L.: Polygon mesh repairing: An application perspective. *ACM Comput. Surv. 45*, 2 (Mar. 2013), 15:1–15:33. 1

[Att10] ATTENE M.: A lightweight approach to repairing digitized polygon meshes. *Vis. Comput. 26*, 11 (Nov. 2010), 1393–1406. 1, 7, 9

[BK04] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Symp. on Geom. Proc.* (2004), pp. 185–192. 5

[BS95] BAREQUET G., SHARIR M.: Filling gaps in the boundary of a polyhedron. *Comput. Aided Geom. Des. 12*, 2 (Mar. 1995), 207–229. 4, 5

[BSBK02] BOTSCH M., STEINBERG S., BISCHOFF S., KOBBELT L.: Openmesh-a generic and efficient polygon mesh data structure. 1, 2, 3, 5, 7

[CPS15] CENTIN M., PEZZOTTI N., SIGNORONI A.: Poisson-driven seamless completion of triangular meshes. *Comp. Aided Geom. Design 35* (2015), 42–55. 4

[Gil79] GILBERT P. D.: *New Results on Planar Triangulations*. Tech. rep., DTIC Document, 1979. 5

[GTLH98] GUÉZIEC A., TAUBIN G., LAZARUS F., HORN W.: Converting sets of polygons to manifold surfaces by cutting and stitching. In *Visualiz.'98.* (1998), pp. 383–390. 2

[GW01] GUSKOV I., WOOD Z. J.: Topological noise removal. *2001 Graphics Interface Proceedings* (2001), 19. 2, 8

[HTG14] HARARY G., TAL A., GRINSPUN E.: Context-based coherent surface completion. *ACM Trans. Graph. 33*, 1 (Feb. 2014), 5:1–5:12. 4

[Ju09] JU T.: Fixing geometric errors on polygonal models: A survey. *J. Comput. Sci. Technol. 24*, 1 (Jan. 2009), 19–29. 1

[Kli80] KLINCSEK G.: Minimal triangulations of polygonal domains. *Ann. Discrete Math 9* (1980), 121–123. 5

[Lie03] LIEPA P.: Filling holes in meshes. In *Symp. on Geom. Proc.* (2003), pp. 200–205. 4, 5

[Möl97] MÖLLER T.: A fast triangle-triangle intersection test. *Journal of graphics tools 2*, 2 (1997), 25–30. 2

[PR05] PODOLAK J., RUSINKIEWICZ S.: Atomic volumes for mesh completion. In *Proceedings of the Third Eurographics Symposium on Geometry Processing* (2005), SGP '05. 4

[SACO04] SHARF A., ALEXA M., COHEN-OR D.: Context-based surface completion. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 878–887. 4