

Mesh repair with user-friendly topology control

Franck Hétroy, Stéphanie Rey, Carlos Andujar, Pere Brunet, Alvar Vinacua

▶ **To cite this version:**

Franck Hétroy, Stéphanie Rey, Carlos Andujar, Pere Brunet, Alvar Vinacua. Mesh repair with user-friendly topology control. Computer-Aided Design, 2011, 43 (1), pp.101-113. 10.1016/j.cad.2010.09.012 . inria-00523005

HAL Id: inria-00523005

https://inria.hal.science/inria-00523005

Submitted on 4 Oct 2010

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

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

Mesh repair with user-friendly topology control

Franck Hétroy [a,*,1,2]

[a] *INRIA, 655 avenue de l'Europe, F-38334 Saint Ismier, France*
Phone: +33 476 615 504 – Fax: + 33 476 615 466

Stéphanie Rey [1] Carlos Andújar [3] Pere Brunet [3] Àlvar Vinacua [3]

Abstract

Limitations of current 3D acquisition technology often lead to polygonal meshes exhibiting a number of geometrical and topological defects which prevent them from widespread use. In this paper we present a new method for model repair which takes as input an arbitrary polygonal mesh and outputs a valid two-manifold triangle mesh. Unlike previous work, our method allows users to quickly identify areas with potential topological errors and to choose how to fix them in a user-friendly manner. Key steps of our algorithm include the conversion of the input model into a set of voxels, the use of morphological operators to allow the user to modify the topology of the discrete model, and the conversion of the corrected voxel set back into a two-manifold triangle mesh. Our experiments demonstrate that the proposed algorithm is suitable for repairing meshes of a large class of shapes.

Key words: topology, morphology, opening, closing, 2-manifold

1 Introduction

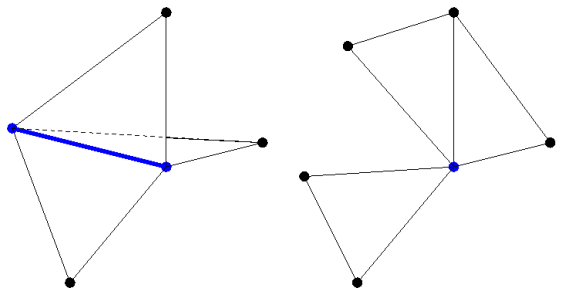
Mesh repair refers to the transformation of a mesh with singularities into an "acceptable" one. "Acceptable" often means a *two-manifold* – a surface S

* Corresponding author
Email address: Franck.Hetroy@imag.fr (Franck Hétroy).
URL: http://evasion.imag.fr/Membres/Franck.Hetroy/ (Franck Hétroy).
[1] Université de Grenoble & CNRS, Laboratoire Jean Kuntzmann
[2] INRIA Grenoble Rhône-Alpes
[3] Universitat Politècnica de Catalunya, Barcelona

such that each point on S has a neighbourhood on S homeomorphic to $\mathbb{R}^2$ (in particular, a two-manifold is a closed surface). Other conditions are sometimes required. A singularity can refer to very different things. We distinguish here three different types of surface singularities.

- *Combinatorial singularities* prevent the mesh, seen as a combinatorial object, from being a two-manifold. We refer to Guéziec et al. [22] for standard definitions related to manifold meshes. Among combinatorial singularities, we find:
  · singular edges: edges with at least three incident faces;
  · singular vertices: vertices whose link is neither a chain nor a cycle.
  See Figure 1 for an example.

  There also are conditions which are singularities only with respect to manifolds without boundaries:
  · boundary edges: edges with only one incident face;
  · boundary vertices: regular (i.e. not singular) endpoints of boundary edges.
  Isolated elements may also be considered singularities:
  · isolated edges: edges with no incident face;
  · isolated vertices: vertices which are not endpoints of any edge.



(a) (b)

Fig. 1. (a) Singular edge (its endpoints are singular vertices), (b) singular vertex.

- *Geometrical singularities* prevent the mesh, seen as the *embedding* of a surface in $\mathbb{R}^3$, from being the boundary of a three-dimensional object. For example, two triangles whose interiors intersect each other create a geometrical singularity. See [40] for a list of possible embedding inconsistencies.
- *Topological singularities* prevent the surface from having the desired genus or the desired number of connected components. As an example, complex meshes often contain small undesired handles, creating multiple small tunnels in the object they represent.

Another common singularity created by modern acquisition processes is a complex hole with (possibly) tiny islands within (see e.g. [15], or Figures 9, 10 and 11). This singularity can be seen either as a set of combinatorial singularities (boundary edges and boundary vertices), or as a geometrical singularity, since it prevents the surface from being the boundary of a volumetric object.

Mesh repair is important mainly for two reasons. First, meshes are widely used to represent (the surface of) 3D objects in computer graphics, because of their flexibility in visualization, manipulation and computation tasks. Second, the acquisition process from a real object (for example, using a scanner) often creates inconsistent meshes. These are due to the inherent limitations of 3D acquisition devices, but also sometimes to the inner geometry of the object. For instance, hidden parts of an object cannot be reconstructed correctly by a scanner. This might not be important, but unfortunately, many applications require as input mesh a valid two-manifold. Consequently, there is much work on tackling this problem of removing singularities from meshes.

In this paper we present a mesh repair algorithm able to remove all geometrical, combinatorial and topological singularities from an arbitrary polygonal model. The main idea is to discretize the input model into a voxel-set representation and to iteratively apply morphological operators to detect areas which are likely to accept topologically-different reconstructions. We allow the user to quickly identify these parts and to choose the desired topology in a user-friendly manner. Once the topology has been fixed, the algorithm extracts a valid two-manifold surface from the voxel set. The use of an intermediate discrete representation allows our algorithm to guarantee a valid two-manifold output for any input model, including polygon soups. To the best of our knowledge, this is the first mesh repair method where the user is assisted by an explicit indication of topologically-ambiguous areas in a discrete representation.

2 Related work

A number of surface reconstruction methods have been proposed to create manifold meshes from various types of input data, including point clouds (see [31] for a recent paper with a comprehensive discussion of related work). These algorithms can be applied also to mesh repair (taking as input e.g. the set of mesh vertices) but they offer no topological control, which is particularly important in presence of noisy or unproperly sampled data. In this section we focus on mesh repair methods for removing combinatorial and geometrical singularities (Section 2.1) and topology modification techniques (Section 2.2).

2.1 Combinatorial and geometrical singularity removal

Mesh repair methods can be split into two categories: surface-based methods and volume-based methods. The former operate directly on the input mesh,

while the latter use an intermediate voxel representation. Note that comprehensive overviews of existing works can be found in [11] and [26].

2.1.1 Surface-based methods

Guéziec et al. propose a method to remove combinatorial singularities [22]. Their method converts a set of polygons into a 2-manifold by applying local operators. It has several advantages: since it does not handle the geometry, coordinates are not important and there is no approximation error; moreover, attributes such as colours, normals and textures can be preserved, and the algorithm works in linear time. Unfortunately, it removes only combinatorial singularities, and much user intervention is often required. Borodin et al. remove several combinatorial and geometrical artefacts such as unwanted gaps and cracks using a vertex-pair contraction operator and an iterative decimation algorithm [10]. Unfortunately, this method does not handle the very complex holes (with possible tiny "islands" inside) produced by modern acquisition hardware. Creating a closed mesh that fills these holes is sometimes known as the *surface completion* problem. Davis et al.'s method [15] was one of the first to tackle this problem. Unfortunately, in some cases it can produce excessively curved regions. Other surface completion methods include [32,42,41,4]. Surface-based methods are often automatic, but fail to repair geometrical singularities such as self-intersecting polygons. For instance, in order to fill holes, these methods only consider the neigbourhood of theses holes, and do not prevent the patches they create from intersecting the surface away from them. An interactive surface-based method is described in [6]. In this work, both the input model and the currently (partly) corrected one are displayed in a visualization interface. Several types of geometrical or combinatorial singularities are highlighted, and the user can select the ones he wants to repair. Attene's automatic method [5] supposes that the sampling of the model is regular, thus practically avoids the previous problem since it modifies the mesh locally and as less as possible. Recently, Pauly and colleagues have proposed to repair a model by replicating discovered regular features [37]. Contrary to previous methods, this technique does not act locally but globally.

2.1.2 Volume-based methods

Volume-based methods generate consistent surfaces, since their output will be the boundary of a volume. Moreover, they provide accurate error bounds between original and final models. One of the first volume-based method is, to our knowledge, Murali and Funkhouser's [35]. This method uses a BSP tree to represent the original surface, but is quite expensive. Recently, Ju proposed a new volume-based algorithm to convert a "polygon soup" into a 2-manifold [25], using an octree to guarantee the creation of a closed surface. This method

is robust in the sense that it preserves detailed geometry and sharp features of the original model. However, it does not handle correctly thin structures and also does not remove topological singularities. Podolak and Rusinkiewicz recently presented a volume-based method for mesh completion [38]. The volume is represented by a graph, which is subsequently separated into two sub-graphs representing the interior and the exterior of the model. This method allows different ways of filling some holes, depending on the object's desired topology. Bischoff et al. [8,9] presented a method to remove combinatorial, geometrical and topological singularities from a CAD model, using an octree. As far as we know, this is the first work which solves all three types of singularities; unfortunately, it is designed mostly for CAD models, since it generates an approximation of the original model (the model is resampled), where sharp features are preserved. Moreover, holes in the mesh are closed only if greater than a user-defined threshold, whereas the value of a relevant threshold may differ from one part of the model to another, depending on the geometry.

2.2 Topology simplification

Removing topological singularities from a mesh is often seen as a different problem. Besides methods simplifying both topology and geometry [18,1,3], a few methods try to simplify topology while preserving the geometry of a model. Guskov and Wood use a local wave front traversal to cut small handles [23], but they cannot detect long thin handles. Moreover, they need a 2-manifold as input. Similarly, the recent method proposed by Wood et al. [43] operates only on 2-manifolds. It finds handles using a Reeb graph, and then measures their size in order to select those to be removed. The final task is quite slow, leading to a relatively high computation time. In the context of medical imaging (the aim is to correctly segment a genus-0 cortex), Kriegeskorte and Goebel propose to use a heuristic estimate of the misclassification damage caused by inverting a voxel in the segmentation, in order to choose between cutting a handle and filling a hole [28]. Nooruddin and Turk proposed a method based on a volumetric representation and on morphological operators to repair and simplify the topology of a mesh before simplifying its geometry [36]. They first voxelize the model, using several scanning directions, apply open and close operators to simplify the topology, extract an isosurface, and then simplify it. Whether they can completely control the topology of the final object or not remains unclear. Recently Zhou et al. proposed a fast and robust method to break the smallest handles of a model [45]. This is done using a volumetric representation of the model, which is thinned to a topological skeleton. Smallest handles are removed by breaking skeleton cycles and then growing the modified skeleton accordingly. The same year, the authors proposed in another paper an original approach in which the user can control the location of handle removal or hole filling by sketching lines [27]. Another user-

assisted program has been proposed in [4]. This method can not only repair some geometrical singularities, but also automatically detect tiny handles. Finally, Campen and Kobbelt propose an approach to modify the topology of a polygonal model, which combines an adaptive octree and nested binary space partitions [12]. Their approach can be used to remove geometrical singularities in a mesh.

3 Method overview

We propose a new method to convert a triangular mesh with geometrical, combinatorial and topological singularities into a 2-manifold whose topology is supervised by the user. It combines volume-based 2-manifold creation and adapted topology modification. The algorithm proceeds through the following steps:

(1) the input surface is converted into a set of voxels, called a discrete membrane;
(2) morphological operators (openings and closings) are applied to the discrete membrane to detect areas which can change topology (hole creation or filling, shell connection or disconnection);
(3) the user selects the voxels to be added or deleted from the discrete membrane
(4) the modified voxel set is converted into a 2-manifold with guaranteed topology.

The pipeline of our algorithm is depicted on Figure 2. Note that stages (2) and (3) can be iterated several times.

We have chosen to use a volumetric intermediate model to be sure to remove all combinatorial and geometrical singularities. The output model is guaranteed to be a 2-manifold. Our method can be related to Nooruddin and Turk's [36] since we also use morphological operators to control the topology. However our classification between interior and exterior voxels is more robust due to the discrete membrane, and we allow the user to monitor the topology modification step. As in [27], topology modification is interactive: as the user often knows the topology of the object (including the location of handles and holes), this provides a better repair than a fully automatic method. But, unlike [27], we have chosen to assist the user during the process, by indicating topologically ambiguous areas. Another interactive program to repair geometrical singularities and remove tiny handles is described in [4]. This program is perfectly suited for meshes with a relatively low number of defects; however, since it uses a surface-based approach, it cannot handle completely degenerate meshes such as polygon soups (see Fig. 15 (a)).

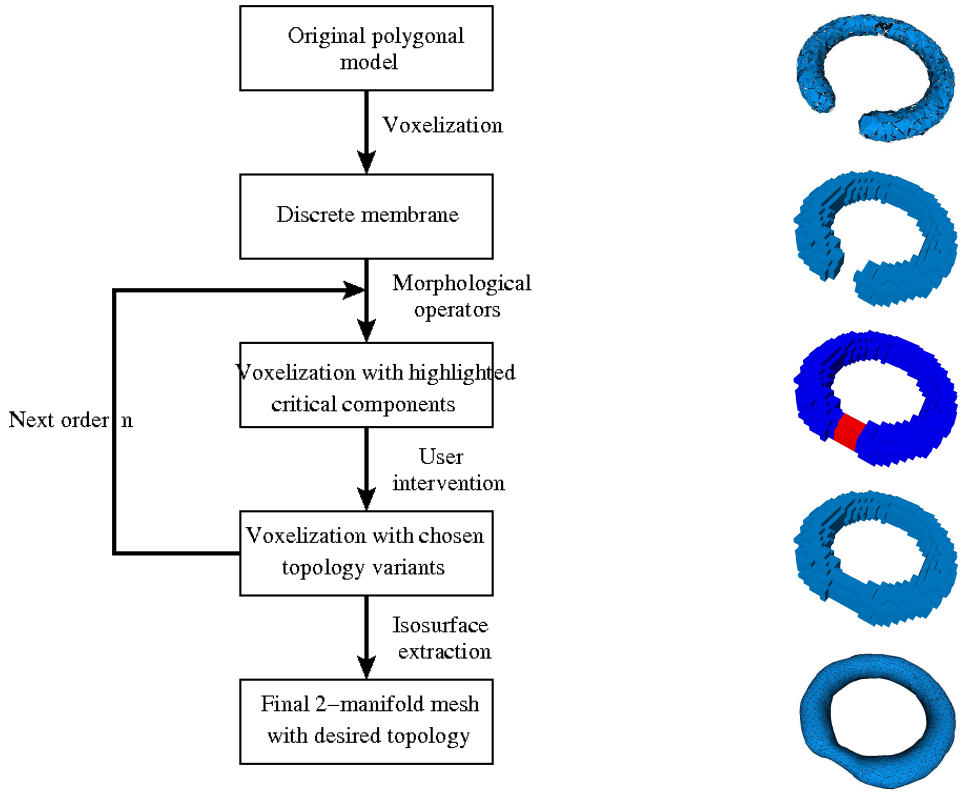


Fig. 2. Algorithm overview.

In our approach the input mesh can be as degenerate as a "soup" of triangles. It is supposed to be a potentially extremely noisy approximation of the boundary of a real, smooth, closed 3D object. The properties of the output mesh are the following:

- all types of singularities listed in Section 1 are solved, i.e. the final mesh is the 2-manifold boundary of a 3D object;
- its topology is controlled by the user;
- its geometry is an approximation of that of the input mesh.

We have chosen not to rely on the input mesh geometry because it can be extremely noisy. However, in case the user wants to repair only a small part of the input mesh, our algorithm can be applied locally. This is discussed in sections 4.4 and 6.3.

Our main contribution is the interactive correction of the mesh topology. We make the following assumptions:

- the mesh represents the boundary of one or several solid objects $O_1, \ldots, O_k$, and the set $\mathbb{R}^3 \backslash \{O_1, \ldots, O_k\}$ is connected; in other words there is no cavity inside any object at the beginning of the process;
- the user knows the correct topology of these objects;

- this topology is relatively simple with respect to the geometry – the number of connected components and holes in the objects is much lower than the number of triangles in the mesh;
- this topology is not necessarily trivial (0 genus), and the (geometrical) location of holes or handles needs user assistance.

Because there is no threshold on the feature size in our method (the user can choose to fill some holes while not filling smaller ones), the algorithm can automatically repair very tiny topological imperfections, in case greater ones exist. See Figure 12 for an example.

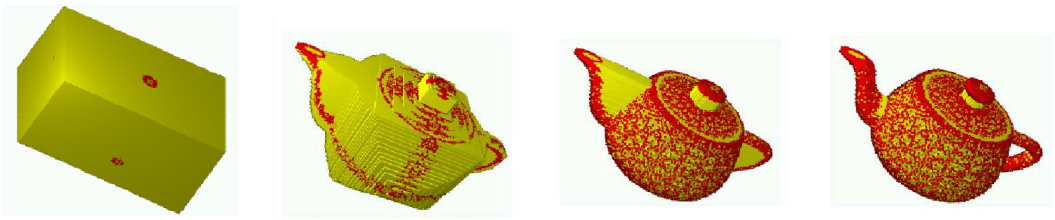
4 Voxelization

4.1 The Discrete Membrane algorithm

To construct a voxel set representing the input model, we use an adapted version of the algorithm described in [19]. This algorithm takes as input a cloud of points, voxelizes the space containing the point set and computes a *discrete membrane* of voxels containing these points (see Figure 3). The discrete membrane is a set of 6- (face-)connected voxels which divides the remaining voxels into *interior* and *exterior*, which means there is no path made of 26- (vertex-)connected voxels disjoint with the membrane that goes from a voxel labeled interior to a voxel labeled exterior. However, to be consistent with the subsequent stages of our method (see sections 5.1 and 5.2), we consider instead the "dual" case where the membrane is required to be 26-connected, while the path-connectedness uses 6-connected paths.

The discrete membrane is initialized as the boundary of the voxelization. It is then contracted using plates, which are sets of $n \times n$ voxels that form a square parallel to a coordinate plane, for decreasing values of n. Each plate is given an orientation, perpendicular to the plate. This orientation allows to distinguish between its front and back sides [19]. Front voxels are the voxels located in front of the plate according to its orientation. Lateral voxels are the voxels located around the plate. Lateral front voxels are the voxels located around the front side of the plate. A plate contraction converts discrete membrane voxels belonging to the plate to outside voxels, while the front, lateral and lateral front voxels of the plate are converted to discrete membrane voxels. The contraction operation is applied recursively at the front, up, down, left and right directions in relation to the plate orientation. If an incursion inside the model is detected, the contraction is undone and corresponding voxels are *frozen*. See [19] for details. The voxels containing the input points locally terminate the shrinking; the process is stopped when the membrane cannot be

contracted anywhere. The number of frozen voxels is at most the number of voxels of the discrete membrane minus the number of voxels containing input points. Note that the membrane is not necessarily simply connected; it can also have a non-0 genus, see for instance Figure 3 (d). Finally, the discrete membrane is relaxed to obtain a smoother surface afterwards.



(a) (b) (c) (d)

Fig. 3. From a cloud of points to a discrete membrane [19]: (a) voxelization of the 3D space (voxels containing input points are shown in red), (b,c) silhouette shrinking with reducing plate size, (d) final discrete membrane.

The main advantage of this algorithm is its robustness – it can handle point clouds with non-uniform density. Although the construction algorithm is not hierarchical and might be relatively slow for huge voxelizations, we have adopted it because because it guarantees us to obtain, as a starting point for subsequent topological and geometrical processing, a coarse approximation of the input soup of triangles which already is almost a 2-manifold: the set of external faces of the discrete membrane. Moreover, the topology of the computed discrete membrane is explicitly related to the distance between the input points (see [19], prop. 7 and 8). In other words, the correct genus and number of connected components are recovered if the point cloud is sufficiently dense (w.r.t. the size of the tunnels or the distance between connected components).

4.2 Voxelization size

The resolution of the voxelization is crucial, since it has a high influence on the results of morphological operators that would be applied to the voxel set. It is a user-defined parameter, but can also be automatically estimated (see [19], section 4.3). On one hand, to be able to represent a topological feature (handle, hole), the size of the voxels at the chosen resolution should be smaller than the feature. This gives a lower bound on the required resolution see Figure 12. However, the higher the resolution, the slower the computation: Table 2 presents timings for three different example voxelizations of the same object (a Sierpinski complex). This algorithm supposes that no voxel of the external "layer" of the initial voxelization belongs to the final discrete membrane. In other words, the voxelization size in each direction is at least equal to the

discrete membrane size in the same direction + 2.

4.3 Extension to meshes

Since our input is not a cloud of points but a triangulated mesh, we have modified the algorithm of [19]. We not only compute the voxels containing the input points, but also the voxels intersecting the faces. This is done using a small additional function, described below (Algorithm 1). Computation time depends on the voxelization size but is bounded by the number of voxels intersecting the triangle bounding boxes, thus the computation is usually done very fast: about 2 minutes for a Buddha model (see Figure 15) with about 300K faces and a voxelization size of more than 2M voxels on a low-end computer (see Section 7.5 for details on timings).

Algorithm 1 Computation of the intersection between the input mesh and the voxelization

function $ComputeIntersection(Voxelization\ W, Mesh\ M)$
  for each triangle T of the mesh M **do**
    Compute the bounding box BB of T;
    for each voxel V of W that intersects BB **do**
      if V intersects T **then**
        Label V as red;
      end if
    end for
  end for
  Return all voxels labeled as red;
end function

4.4 Local voxelization

To speed up the process, the user can choose to select only a part of the mesh and to apply this voxelization stage locally. This does not change the input mesh in distant areas from the selected polygons. For the subsequent topology correction to be efficient, the selected zone must be significantly greater than the size of the topological features to be repaired. Also, since the original discrete membrane algorithm aims at recovering a closed object, additional voxels must be added at the boundary of the computed voxelization to fill unwanted tunnels and get a closed set of voxels (see Figure 14 for an example). To do this, we add planes of voxels, lying on the boundary voxels. Areas of the final surface corresponding to these voxels will be removed later, once the topology has been modified.

4.5 Comparison with other voxelization techniques

As stated previously, we use an enhanced version of the algorithm presented in [19] because it can handle difficult cases, including non-uniform polygon soups. Some other techniques produce fast results but cannot be applied in all cases, such as [17] which requires as input a watertight mesh. We compared our method with the `binvox` program [34], which is based on the algorithm proposed by Nooruddin and Turk [36] (discussed in Section 2.2), and with the algorithm proposed by Haumont and Warzée [24]. On the noisy Buddha triangle soup (see Fig. 15 (a)), our method took 129s to create a $100 \times 234 \times 100$ voxelization, while `binvox` took 112s to create a $132 \times 132 \times 132$ voxelization. Results are similar voxelizations with a similar number of voxels. In the meantime, the method by Haumont and Warzée only took 56s on the same low-end computer to voxelize the input polygon soup with an octree of depth 7. However, some voxels outside the model are wrongly included in the voxel set.

5 Interactive topology modification using morphological operators

Before converting the discrete membrane into a two-manifold mesh, it is important to let the user decide the final topology, including the number of surface shells and their genus (number of holes or, similarly, handles). To control the topology of the output of our algorithm, we apply morphological operators on a volume. The volume is not the discrete membrane itself, but the discrete membrane plus the interior of the object it bounds, which is automatically known from the discrete membrane construction. Since each voxel of the discrete membrane is 6-connected to at least one interior voxel, this voxel set is a 3-manifold with boundary – the neighbourhood of each point is homeomorphic to either a sphere or a hemisphere.

5.1 Topology of discrete volumes: notations and definitions

Let $\mathcal{V}$ be the voxel set. Its numbers of vertices, edges, faces and voxels are respectively noted $k_0(\mathcal{V})$, $k_1(\mathcal{V})$, $k_2(\mathcal{V})$ and $k_3(\mathcal{V})$. The Euler characteristic χ of $\mathcal{V}$ is defined as $\chi = k_0(\mathcal{V}) - k_1(\mathcal{V}) + k_2(\mathcal{V}) - k_3(\mathcal{V})$.

The topology of a 3-manifold can be characterized by three numbers, named *Betti numbers*. j^{th} Betti number β_j is defined as the rank of the j^{th} homology group H_j (an introduction to homology groups, with accurate definitions of Betti numbers, can be found in [16]). What is more interesting for our study is

that Betti numbers correspond to numbers of connected components ($\beta_0(\mathcal{V})$), tunnels ($\beta_1(\mathcal{V})$) and voids (or cavities) ($\beta_2(\mathcal{V})$) of the volume $\mathcal{V}$. Betti numbers are also related to χ, because $\mathcal{V}$ is a cell complex: $\chi = \beta_0(\mathcal{V}) - \beta_1(\mathcal{V}) + \beta_2(\mathcal{V})$.

The topology of the final surface is linked to the topology of our voxel set, since this surface corresponds to its boundary. The number of connected components of the surface equals $\beta_0(\mathcal{V})$, and its genus (sum of the numbers of holes of all connected components) equals $\beta_1(\mathcal{V})$, provided that we use consistent neighbourhood definitions. In the following, we use the 26-neighbourhood relationship for the volume $\mathcal{V}$, and the 6-neighbourhood relationship for $\mathcal{V}^C$ the complementary set of $\mathcal{V}$: two voxels sharing a vertex, but not an edge, are said to be neighbours if they both belong to $\mathcal{V}$, but not if they belong to $\mathcal{V}^C$. This prevents topological inconsistencies. We use the 26- instead of the 6-neighbourhood relationship for $\mathcal{V}$ to be consistent with the computation of χ as $k_0(\mathcal{V}) - k_1(\mathcal{V}) + k_2(\mathcal{V}) - k_3(\mathcal{V})$.

Computing Betti numbers is not a trivial task [21]. The number of connected components $\beta_0(\mathcal{V})$ can be computed in various ways, for instance using disjoint-set data structures [14]. The number of cavities equals the number of connected components of $\mathcal{V}^C$ minus one (representing the "exterior" of $\mathcal{V}$), so it can also be computed. However $\beta_1(\mathcal{V})$ is not easily found. Lee et al. compute $\beta_1(\mathcal{V})$ counting the number of non-separating cuts [30] ($\beta_1(\mathcal{V})$ is the maximal number of non-separating cuts not increasing $\beta_0(\mathcal{V})$). Another algorithm to detect non-separating cuts has been proposed by Guskov and Wood [23]. In our approach, we prefer to compute $\beta_1(\mathcal{V})$ locally, following the approach of Bischoff and Kobbelt [7]. This enables us to quickly detect topological changes caused by the application of morphological operators (see Section 5.3), even if the two other Betti numbers still need to be computed globally. $\beta_1(\mathcal{V})$ is computed thanks to previous relations $\chi = \beta_0(\mathcal{V}) - \beta_1(\mathcal{V}) + \beta_2(\mathcal{V})$ and $\chi = k_0(\mathcal{V}) - k_1(\mathcal{V}) + k_2(\mathcal{V}) - k_3(\mathcal{V})$. In order to efficiently compute χ (without keeping track of faces, edges, etc.), we exploit the fact that χ is additive, as did for instance [7] – given two sets A and B, $\chi(A \cup B) = \chi(A) + \chi(B) - \chi(A \cap B)$. Since each vertex of $\mathcal{V}$ belongs to 8 voxels (remember that no voxel of $\mathcal{V}$ belongs to the external "layer" of the initial voxelization), χ is the sum of the local Euler characteristics around each vertex of the voxelization divided by 8. The local Euler characteristic around a vertex v is defined as $k_0(\mathcal{V}, v) - k_1(\mathcal{V}, v) + k_2(\mathcal{V}, v) - k_3(\mathcal{V}, v)$, with $k_0(\mathcal{V}, v), k_1(\mathcal{V}, v), k_2(\mathcal{V}, v)$ and $k_3(\mathcal{V}, v)$ denoting respectively the number of vertices, edges, faces and voxels that both belong to $\mathcal{V}$ and are incident to v. It can be computed using a lookup table, since only 256 2×2 voxel configurations can occur (in fact, up to isomorphism, we only have 22 different configurations). Moreover, since χ is additive, we do not need to compute the Euler characteristic around each vertex at each step of our algorithm. Each time we add or remove voxels, we only need to update local Euler characteristics around corresponding ver-

tices. Once χ is computed, we immediately have the number of tunnels in our volume: $\beta_1(\mathcal{V}) = \beta_0(\mathcal{V}) + \beta_2(\mathcal{V}) - \chi$.

5.2 Morphological operators

In order to track down areas of the object where topology is wrong (that is to say, irrelevant handles creating tunnels or connecting different components, or on the contrary missing tunnels or bridges between several parts of a connected component), we use morphological operators. Basic operators are *erosion* and *dilation*. The erosion operator $\mathcal{E}$ transforms the set of voxels $\mathcal{V}$ into the set $\mathcal{E}(\mathcal{V}) = \{V \in \mathcal{V}$, all 26-neighbours of V are also in $\mathcal{V}\}$. The dilation operator $\mathcal{D}$ transforms $\mathcal{V}$ into the set $\mathcal{D}(\mathcal{V}) = \{$ voxels $V \in \mathcal{V}$ or V is a 26-neighbour of some voxel of $\mathcal{V}\}$ [7]. Two combinations of these two operators are called *opening* and *closing*: $\mathcal{O} = \mathcal{D} \circ \mathcal{E}$ and $\mathcal{C} = \mathcal{E} \circ \mathcal{D}$. Erosion and opening can expand holes and disconnect parts, while dilation and closing can close holes and connect previously disconnected parts of the volume. Figure 4 shows these four operators applied on an example. Note that they can be defined using either the 6- or the 26-neighbourhood relationship; we choose to use the 26-neighbourhood relationship because it generates larger modifications to the set $\mathcal{V}$. This choice has no influence on the neighbourhood relationship defined later for the connected components of the set.

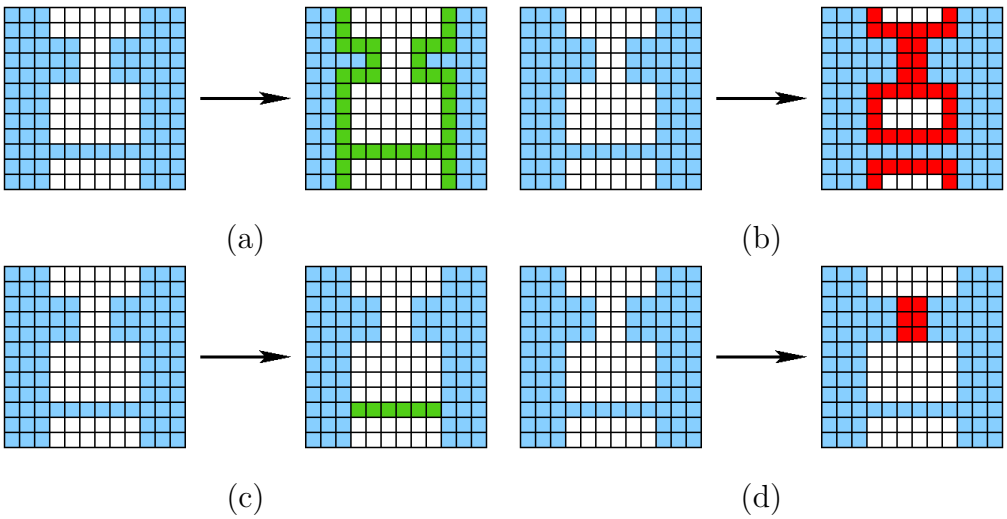


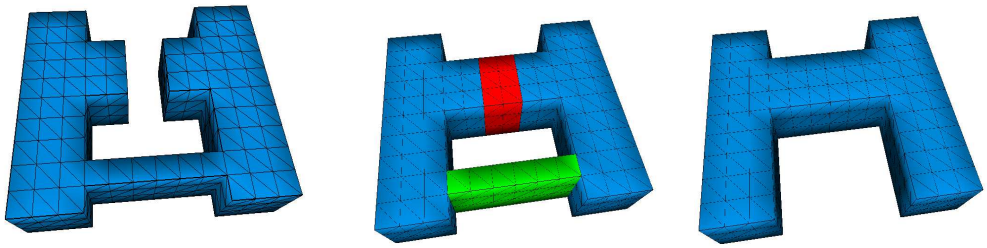
Fig. 4. Morphological operators applied on a 2D set of blue pixels: (a) erosion, (b) dilation, (c) opening, (d) closing. Removed pixels are in green while added pixels are in red.

To go further and detect bigger topologically critical areas, we can iterate this process. We call *opening of order n* (noted $\mathcal{O}^n$) a sequence of n erosions followed by n dilations, and *closing of order n* (noted $\mathcal{C}^n$) a sequence of n dilations followed by n erosions, $n \geq 1$.

Note that we choose to use opening and closing instead of erosion and dilation because they avoid shrinkage or expansion of the model. Erosion and dilation are faster to compute, but they usually shrink or expand the model.

5.3 Algorithm

We start from the voxel set $\mathcal{V}$ (the discrete membrane plus its interior volume). The discrete membrane is computed as described in section 4 (see Figure 5 (a)). In order to detect topologically critical areas, we apply openings and closings to $\mathcal{V}$ (the order n is selected by the user). We then compute the set of voxels which belong to $\mathcal{V}$ and not to $\mathcal{O}^n(\mathcal{V})$ and the set of voxels which belong to $\mathcal{C}^n(\mathcal{V})$ and not to $\mathcal{V}$. We cluster voxels of these two sets in 26-connected components (because we use the 26-neighbouring relationship for our set of voxels). For each component K, we then compute the Betti numbers of the new set of voxels $\mathcal{V}\backslash K$ or $\mathcal{V} \cup K$ (remember that the new Euler characteristic can be computed simply by adding or removing local Euler characteristics of vertices of K to or from the Euler characteristic of $\mathcal{V}$), and compare it to the Betti numbers of $\mathcal{V}$. If one of them changes, we have detected a topologically critical area, which we call a *critical component* of the voxel set. In this case, K is labelled with a special tag: "candidate for removal" or "candidate for addition". The discrete membrane together with the critical components are displayed in a visualization interface, in which the user can select to remove and/or add some critical components to the voxel set (Figure 5 (b) and (c)). Each time a critical component is removed or added, the new topology is displayed.



(a) (b) (c)

Fig. 5. (a) Discrete membrane. (b) Discrete membrane with critical components: candidate voxels for removal are shown in green and candidate voxels for addition are shown in red. (c) Voxel set after the user chose to add the red component and to remove the green one. In this example, we applied an opening and a closing of order 1.

In case the user does not have a guess about the opening and closing order n he should apply, or in cases he wants to remove topological artifacts with various sizes, he can apply this algorithm several times, with different values

for n. For instance, small topological errors can be corrected first, with a small value for n; then large ones can be detected with a larger value for n. Figure 16 shows an example of this process.

6 Isosurface computation

Once we get a discrete volume with the desired topology, we convert it to a surface by using a Marching Cubes-like algorithm. We use the dual of the voxelization as the grid; each vertex of this grid is labeled as *interior* to the surface if it corresponds to a voxel of $\mathcal{V}$, otherwise it is considered as *outside* the surface. The volume of the resulting surface roughly corresponds to the volume of $\mathcal{V}$. This volume may thus be a bit greater than the volume inside the input mesh, because $\mathcal{V}$ includes all voxels intersecting the mesh. However, the surface is then shrunk during the smoothing step (see section 6.2), thus reducing this volume.

6.1 Topology preservation

The standard Marching Cubes algorithm [33] is known to generate topological inconsistencies, due to ambiguous configurations, which are called *X-faces* and *X-cubes* in [2] (see Figure 6). X-cube corresponds to pattern number 4 of the original Marching Cubes look-up table [33], while X-faces appear in patterns number 3, 6, 7, 10, 12 and 13. In our case, each X-face is related to two voxels (either of $\mathcal{V}$ or of $\mathcal{V}^C$) which are edge-connected. Each X-cube is related to two voxels which are only vertex-connected.

Since we defined connections between voxels of $\mathcal{V}$ using the 26-neighbourhood relationship (thus using the 6-neighbourhood relationship for $\mathcal{V}^C$, see Section 5), we need to connect interior vertices, within each X-face and each X-cube. This corresponds to situations described on Figure 7, which lead to cases 3.1, 3.2, 4.1.1, 4.1.2, 6.1.1, 6.2, 7.1, 7.4.1, 10.1.1 and its opposite configuration, 12.1.1 and its opposite configuration, and 13.1 and its opposite configuration, in Chernyaev's advanced look-up table ([13]; see Figure 8 of this paper for the corresponding local triangulations).

6.2 Smoothing

The previous stage generates a 2-manifold whose vertex coordinates have been estimated in a very simple way: each vertex lies in the midpoint of an edge

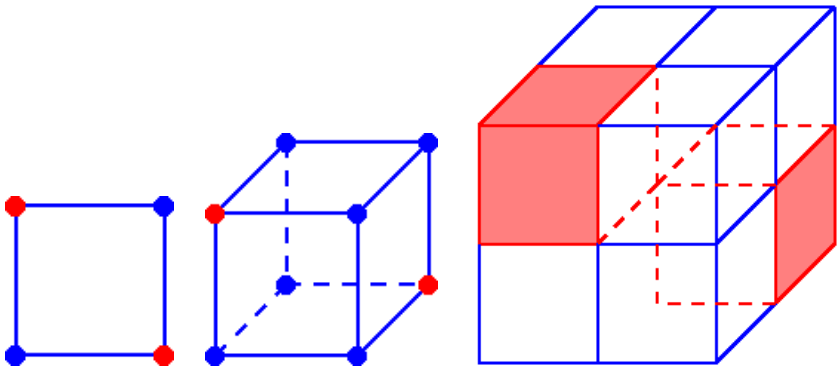


Fig. 6. Marching Cubes ambiguous cases (from [2]). (a) X-face: two opposite vertices correspond to voxels of $\mathcal{V}$ (*interior vertices*), while the two others correspond to voxels of $\mathcal{V}^C$ (*outside vertices*). (b) X-cube is the only Marching Cubes ambiguous configuration without any X-face. (c) Corresponding voxelization of (b): voxels of $\mathcal{V}$ (in red) are 26-connected, but not 6-connected.

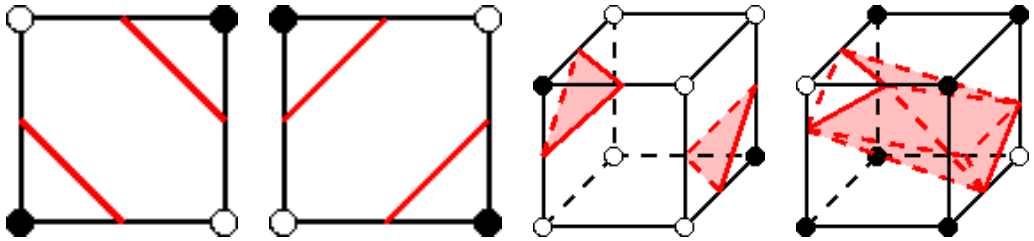


Fig. 7. Solving Marching Cubes ambiguous cases. White vertices correspond to voxels of $\mathcal{V}$, while black vertices correspond to voxels of $\mathcal{V}^C$.

of the grid. Since we assume the input mesh is noisy, it seems reasonable not to rely on the original vertex positions. In case a smooth mesh is expected as output, some postprocessing is required. The smoothing method must fulfill the following requirements:

- it should not require additional information (such as expected normals),as those produced by the discrete Marching Cubes are not suitable;
- it should preserve features as much as possible while correctly smoothing the sharp edges introduced by the previous method;
- most importantly, it must neither change the topology of the mesh nor create new singularities, such as auto-intersections.

In our implementation, we have chosen to apply the bilateral mesh denoising method of [20], which is fast and satisfies the previous conditions. In practice, no singularity is created as long as the smoothing does not destroy geometrical features; the strength of the smoothing can be controlled with very simple parameters. Only one parameter has been kept in our implementation – the

number of iterations. The normal to the surface at a vertex is computed using the 2-ring neighborhood of the vertex, because by the discretereconstruction technique used, normals are also discretized. The neighborhood used for the computation of the other parameters is set to the 1-ring neighborhood; this is a valid approximation because the aspect ratio of the faces is, by construction, bounded over the mesh.

6.3 Local computation

In case only a part of the input mesh was voxelized, we need to compute a surface that will be merged with the input one. Let $\mathcal{M}$ be the input mesh, and let $\mathcal{S}$ be the selected part of $\mathcal{M}$ that is voxelized into the set $\mathcal{V}$. We recall that additional voxels are added to $\mathcal{V}$ to fill tunnels. Once the Marching Cubes algorithm has been applied, triangles corresponding to at least one of these added voxels are removed first. The boundary of the computed surface $\mathcal{S}'$ is made of one or several (edge) loops, since we added connected, genus-0, sets of voxels. More precisely, the boundary of $\mathcal{S}'$ should be made of as many loops as the boundary of $\mathcal{S}$. Finally, we merge $\mathcal{S}'$ to $\mathcal{M}\backslash\mathcal{S}$ by stitching both boundaries, using pairs of loops. Several algorithms, such as [39,44,29], can be applied to stitch boundaries. In our case we use the following simple user-assisted method, which yields satisfactory results (see Figure 14).

We assume that boundaries we want to stitch are sets of closed lines which need to be matched by pairs. Each closed line is defined as a set of k vertices $v_0, v_1, \ldots, v_{k-1}$, together with the set of edges $(v_0, v_1), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_0)$. Let us denote $L = (v_0, v_1, \ldots, v_{k-1})$ and $L' = (w_0, w_1, \ldots, w_{l-1})$ two closed lines to be matched. We suppose these closed lines do not intersect; if this is not the case we remove a strip of boundary triangles of $\mathcal{S}'$ so that the new connected component of the boundary L does not intersect L'. We then create new triangles between L and L'. First, each of the two closed lines is cut into developable pieces. This is done manually, usually in a few seconds. We may cut each line into more pieces than required, in order to be sure new triangles subsequently computed from various pieces will not intersect. An efficient heuristic is to cut lines around sharp angles.

Let us now suppose that the subset $(v_i, \ldots, v_j)$ of L needs to be matched with the subset $(w_{i'}, \ldots, w_{j'})$ of L'. We project these two pieces onto a common plane $\mathcal{P}$, which is defined by its normal $v_iv_j \times v_iw_{i'}$ and the point $\frac{v_i+w_i}{2}$ through which it passes. Orthogonal projections of the vertices to this plane are denoted as $v_i^p, \ldots, v_j^p, w_{i'}^p, \ldots, w_{j'}^p$. We then compute a constrained Delaunay triangulation of this set of points, enforcing $(v_i^p, v_{i+1}^p), \ldots, (v_{j-1}^p, v_j^p), (w_{i'}^p, w_{i'+1}^p), \ldots, (w_{j'-1}^p, w_{j'}^p), (v_i^p, w_{i'}^p)$ and $(v_j^p, w_{j'}^p)$ to be edges of this triangulation. Neighbourhood relationships between vertices induced by this triangulation enables us to finally create new

triangles that stitch $(v_i, \ldots, v_j)$ to $(w_{i'}, \ldots, w_{j'})$.

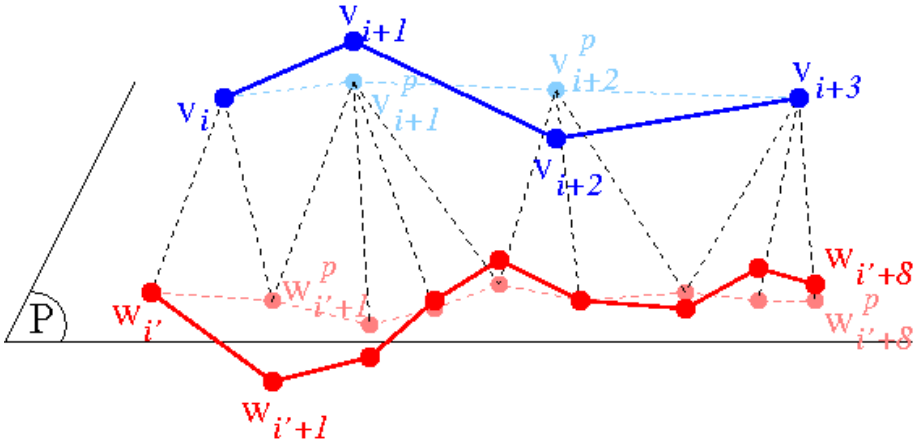


Fig. 8. Stitching algorithm: lines to be stitched $v_i, \ldots, v_j$ and $w_{i'} \ldots w_{j'}$ are projected onto a common plane, then a constrained Delaunay triangulation is computed.

Although this method is quite simple, it yields suitable results in our experiments. Nevertheless more sophisticated, automatic techniques are available (see e.g. [39,44,29]).

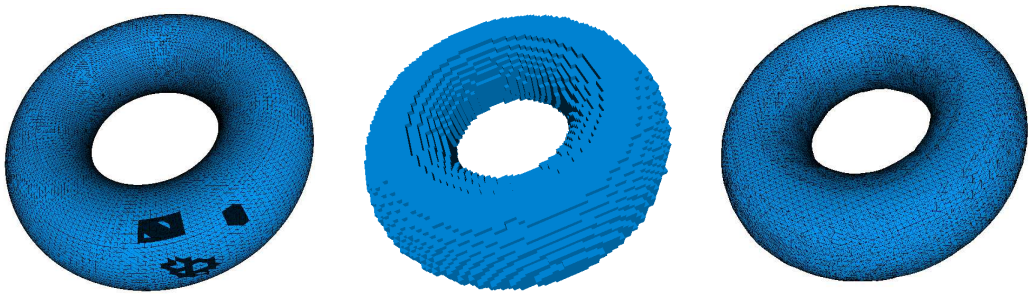
7 Results and discussion

Figure 2 shows the entire process on a model with all types of singularities. This model is a soup of triangles, some of them overlapping, with a hole-like region in the background. The computation of the discrete membrane leads to a voxel set with one connected component with no tunnels. A closing of order 2 enables us to create a tunnel, which leads to a torus-shaped final 2-manifold surface: one connected component, genus 1.

7.1 Repair of combinatorial and geometric singularities

Models with complex holes can be repaired with our method, as can be seen on Figure 9. Very noisy models with holes can also be transformed into smooth 2-manifolds, such as the Buddha model in Figure 15 (a).

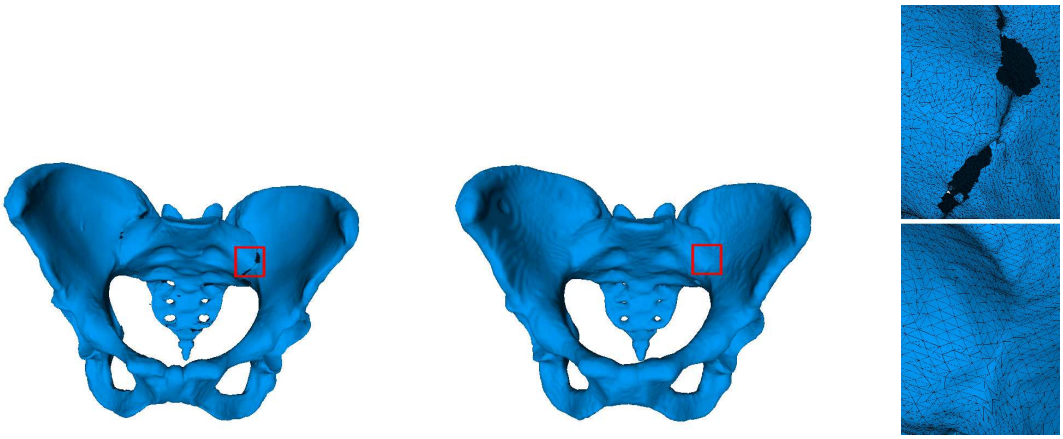
Figures 10 and 11 show how our algorithm can repair noisy meshes acquired with a laser scan. Both input meshes contain many small holes, some of them with islands, and even outliers. In both cases our algorithm produces a two-manifold mesh, with the correct topology (one connected component, genus 9 for the pelvis model, one connected component and genus 0 for the statue). Outliers have been removed from the pelvis model by applying an order 1



(a) (b) (c)

Fig. 9. (a) Input model, containing complex holes with islands within. (b) Computed discrete membrane. (c) 2-manifold result.

opening, and holes have been filled by applying order 1 and order 2 closings. On the statue model, an order 1 closing was sufficient to recover the correct topology.



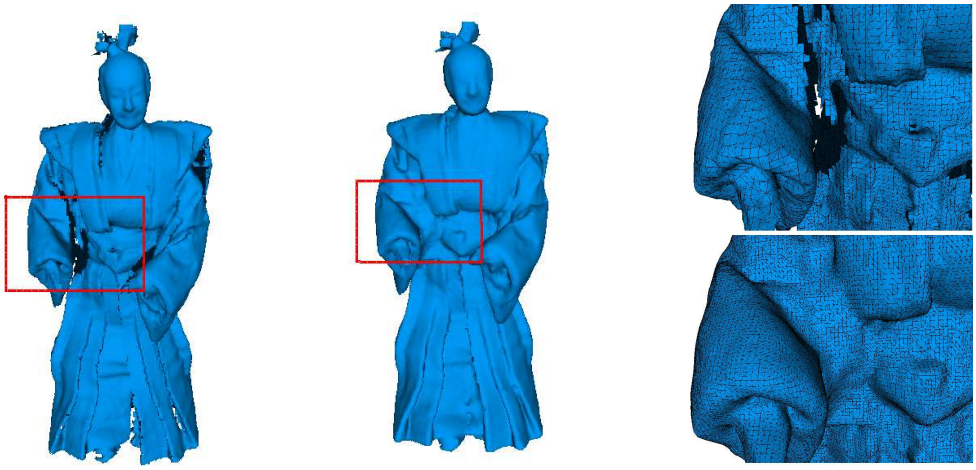
(a) (b) (c)

Fig. 10. (a) Input model. (b) Output mesh. (c) Close up.

7.2 Repair of topological singularities

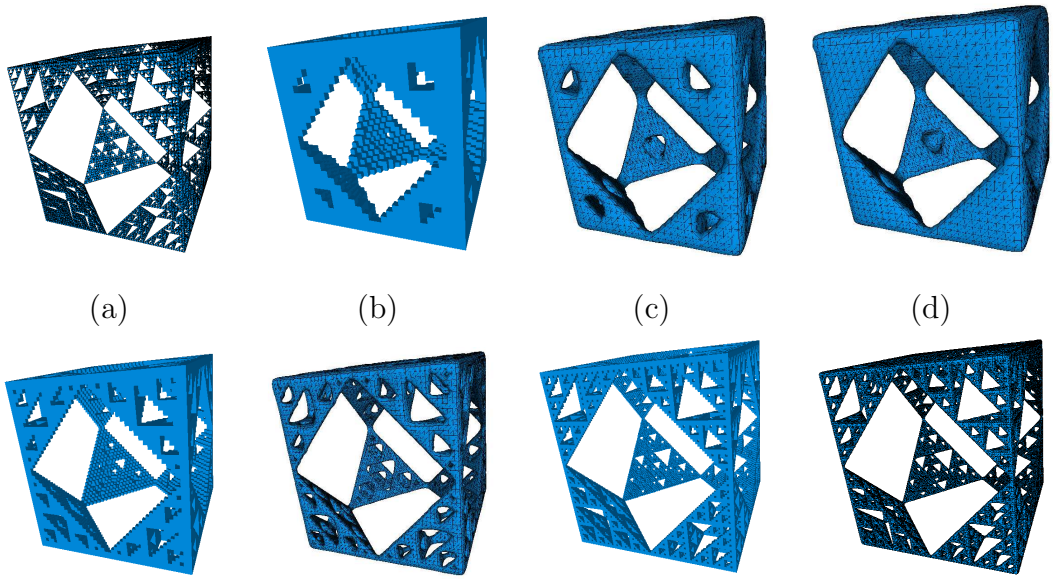
As can be noticed on Figure 12, the resolution chosen for the voxelization has a major influence on the resulting topology: topological features smaller than the voxel size are not recovered. Thus, choosing an appropriate voxelization size can save time for the user, by automatically filling the smallest holes during the discrete membrane computation.

The same model can be repaired in different ways, depending on the expected topology, as shown on Figures 13 and 15. On the statue model, an opening of order 3 is necessary to modify the topology. A closing of order 2 was applied on the noisy Buddha model to fill the two holes on its left side. No morphological



(a) (b) (c)

Fig. 11. (a) Input model. (b) Output mesh. (c) Close up.



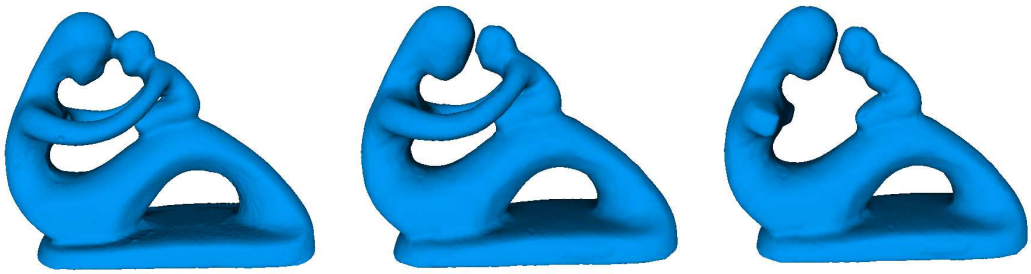
(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 12. (a) Input model: approximation of a Sierpinski fractal. (b) $30 \times 30 \times 30$ voxelization. (c) Output mesh (no morphological operator applied). (d) Output mesh (one closing applied). (e) $50 \times 50 \times 50$ voxelization. (f) Output mesh (no morphological operator applied). (g) $100 \times 100 \times 100$ voxelization. (h) Output mesh (no morphological operator applied).

operator is necessary to recover the topology (genus = 6) of the original non noisy model.

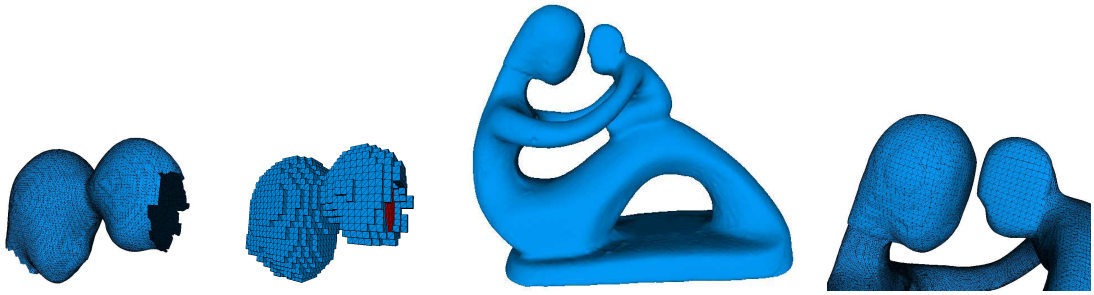
Figure 14 shows an example of a local topological repair on the model depicted in Figure 13. Selecting only a small part of the input model saves time at all stages (see Table 2). It thus allows to select a finer voxelization size, in order



(a) (b) (c)

Fig. 13. (a) Input model, with genus 4. (b) Genus 3 output mesh. (c) Genus 1 output mesh.

to get a more accurate final surface with respect to the initial mesh geometry.



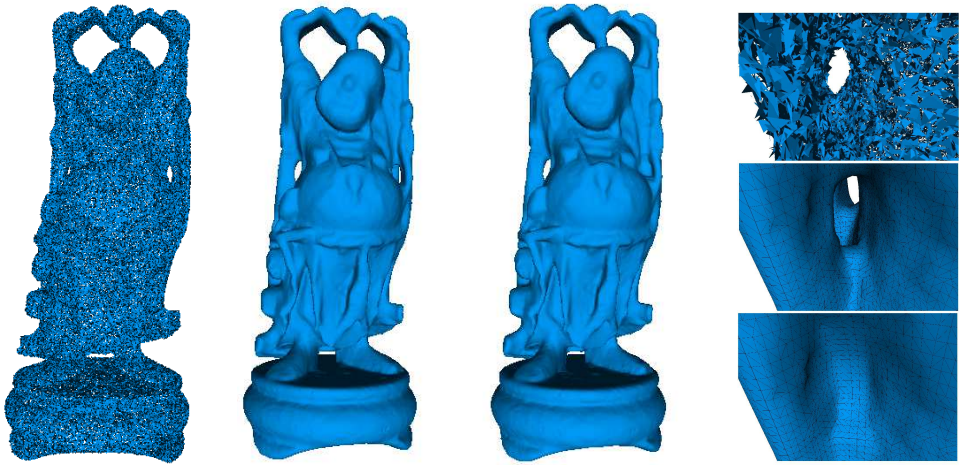
(a) (b) (c) (d)

Fig. 14. (a) Selected zone on the input model. (b) Voxelization, with added voxels in red. (c,d) Output mesh. Compare with Figure 13 (b).

Figure 16 shows how our method can be applied for interactive topology correction. In this example, we merge all the bones of the foot into one connected component. After voxelizing the model, we start by computing the order 1 opening and closing (b). We select some 6-connected components of voxels to be added to the set (in yellow), while we discard others (in red). Green voxels refer to connected components proposed for removal. Obviously, we select none of them. Once the desired voxels have been added to the set, we compute the order 2 opening and closing (c). Once again we choose to add some of the proposed connected components to the voxel set. This leads us to a final set made of only one 6-connected component; thus we can now convert it back to a surface (d).

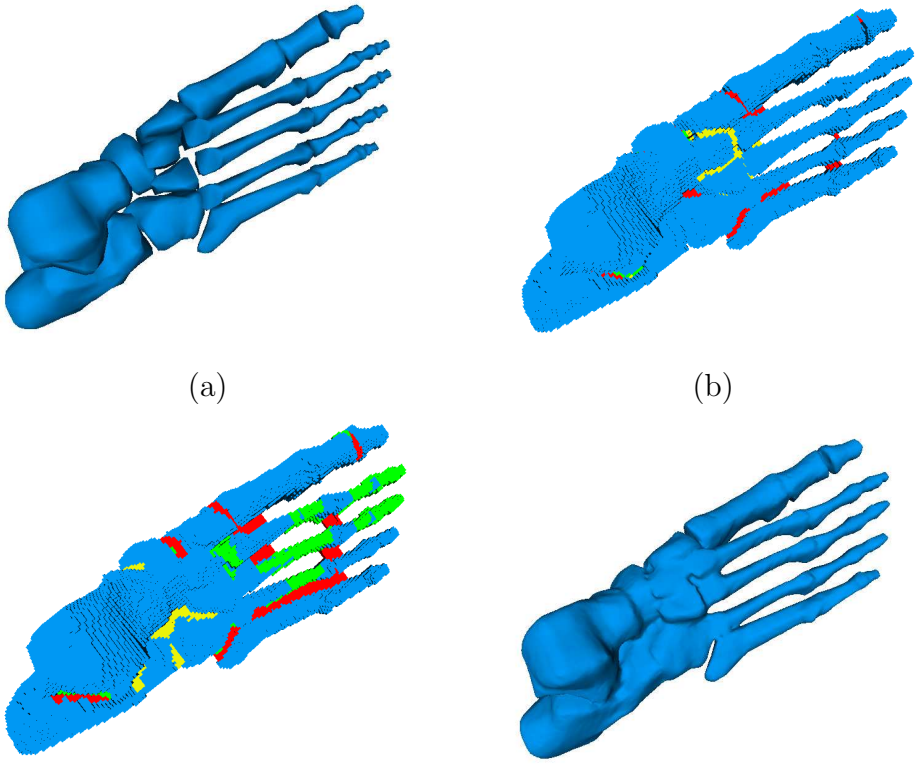
7.3 Number of critical connected components

Table 1 gives the number of computed critical components for some models shown in Figures 10 to 18. This number can be large, therefore inspecting all critical components one by one, as described before, can be tedious and



(a) (b) (c) (d)

Fig. 15. (a) Input model: noisy Buddha triangle soup. (b) Genus 6 output mesh. (c) Genus 4 output mesh. (d) Close-up on the left side (back view).



(a) (b)

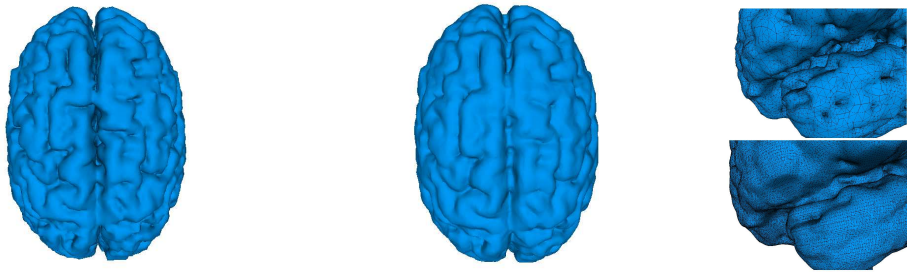
(c) (d)

Fig. 16. (a) Input model. (b) After the order 1 opening and closing. (c) After the order 2 opening and closing. (d) Output mesh.

Model	$\mathcal{O}^1$	$\mathcal{C}^1$	$\mathcal{O}^2$	$\mathcal{C}^2$	$\mathcal{O}^3$	$\mathcal{C}^3$	**Total**
Pelvis	4	32	–	10	–	–	46
Statue 1	–	26	–	–	–	–	26
Statue 2	–	–	–	–	3	–	3
Buddha	–	–	–	5	–	–	5
Foot	21	7	16	5	–	–	49
Brain	17	16	–	–	–	–	33
Paperclip	–	–	3	1	–	–	4

Table 1
Number of critical connected components for some models. $\mathcal{O}^1$ means after applying an order 1 opening, $\mathcal{C}^1$ after an order 1 closing, etc.

time consuming. In case the desired topology is simple (e.g. one connected component with genus 0) or, more generally, in case the location of critical components is not important, our algorithm can be applied without user intervention, since for each critical component we know exactly how it affects the topology. Figure 17 illustrates this point: the initial cortex model has genus 45, while we want a genus 0 mesh. We automatically set all critical components which reduce the genus to be selected. Doing so, we get a final mesh with genus 0 after applying only the order 1 opening and closing.



(a) (b) (c)

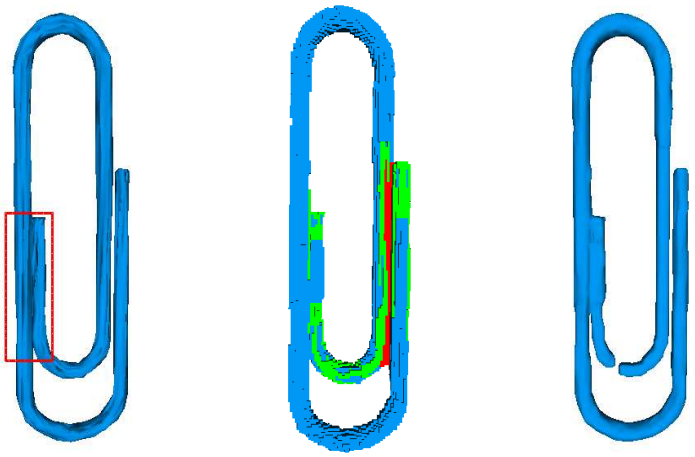
Fig. 17. (a) Input model, with genus 45. (b) Genus 0 output mesh. (c) Close up.

Nevertheless, please note that this simple idea cannot be used in case the desired genus is not 0.

7.4 Failure case

Our algorithm can fail in some cases, especially when trying to change the topology in thin, elongated areas. For instance, see the noisy paperclip model in Figure 18 (a). The original model is supposed to have genus 0, being just a winded stem. Hence, we would like to repair this mesh around the area

highlighted in red. Unfortunately, an order 1 opening is not able to detect any critical component. Meanwhile, the three critical components that an order 2 opening finds encompass a large set of voxels (see on Figure 18 (b); one changes the genus, while the two other break the model into two connected components). This is because of the thin, tubular shape of the model. When applying erosion, we hardly control the area where the voxel set is broken into several components. It may even happen that almost all voxels disappear at the same time.



(a) (b) (c)

Fig. 18. (a) Input mesh with the wrong genus (1 instead of 0) due to noise. (b) After the order 2 opening and closing. (c) Selecting a critical component for deletion yields the correct genus, but the result is not the one sought.

7.5 Timings

Table 2 gives computation times (in seconds) for the all models presented in this paper, on a desktop PC with a 2.13 GHz Pentium 4 processor. We also give user interaction times. 10 iterations were performed for the smoothing stage, except on the pelvis model (5 iterations only). No opening or closing was performed for the torus and the bunny models, since the discrete membrane alone yielded the desired topology. Timings for the pelvis and the foot models include both order 1 and order 2 morphological operators. Interaction times are given for comparison purpose only, since they greatly depend on the visualization interface and the framerate.

The most time-consuming stage is the application of morphological operators. This time can be prohibitive in some cases. Fortunately, it can often be shortened. For instance, many models do not have any internal cavity, and morphological operators hardly create one. In these cases, the computation of the number of cavities can be turned off, resulting in a significant speed up of

Model	Nb of faces	Vox. size	1	2	3	Total	User
Torus	25 300	66x23x66	< 1	0	3	< 4	0
Pelvis	500 000	120x100x136	93	669	12	774	20
Statue 1	141 000	285x123x100	64	3491	34	3589	170
Sierpinski	32 000	30x30x30	< 1	< 1	2	< 4	40
Sierpinski	32 000	50x50x50	1	0	5	6	0
Sierpinski	32 000	100x100x100	34	0	23	57	0
Statue 2 (genus 3)	483 000	113x83x45	3	30	7	40	5
Statue 2 (genus 1)	483 000	113x83x45	3	30	5	38	10
Statue 2 (local)	42 500	39x28x21	< 1	< 1	1	< 3	5
Buddha (genus 6)	293 000	100x234x100	129	0	27	156	0
Buddha (genus 4)	293 000	100x234x100	129	2206	29	2364	10
Foot	4 200	216x75x84	16	962	39	990	80
Brain	290 000	134x159x177	21	4014	34	4069	0
Paperclip	2 900	54x180x20	< 1	6	4	< 11	5

Table 2
Computation times (in seconds) for the models of Figures 9 to 18. Stage 1 is the discrete membrane computation, stage 2 is the application of morphological operators and stage 3 is the isosurface computation and smoothing. Statue 1 corresponds to the model of Figure 11, and Statue 2 to the model of Figures 13 and 14.

the process. In our experiments, from 30 to 15s for the second statue model (Figure 13), from 21min 56s to 3min 26s for the foot bone model, and even from 36min 46s to 6min 3s for the Buddha model.

7.6 A guide on user intervention

We now sum up the different ways the user can control the mesh repair process with the proposed method.

- **Voxelization size**: this parameter can be set automatically (see Section 4.2). The larger the size, the slower the algorithm but also the finer the geometrical and topological features that can be recovered. In case the user does not have any clue, we advice to test the voxelization stage with 3 to 5 different sizes, then to choose the smallest one which generates the discrete membrane with sufficient detail.
- **Local application**: in case the model has a huge number of faces, but only very localized singularities to be repaired, we suggest to use our method

locally, then to stitch the result to the non-modified part of the mesh (see Section 6.3).

- **Order of the morphological operators**: this parameter controls the size of the topological singularities that can be detected. Usually, orders lower than 3 are sufficient to recover most singularities. They can be used one after the other.
- **A priori knowledge about the model's topology**: the application of the morphological operators to the voxel set is the most time-consuming stage of the algorithm, but it can be shortened if the user has some information about the desired topology. For instance, the computation of the number of cavities can be turned off if it is known that the model does not have any, or the repair can be done without user intervention if the desired genus is 0 (see Section 7.3).
- **Mesh smoothness**: The third and last user-controlled parameter is related to the bilateral mesh denoising algorithm we use for mesh smoothing. In our experiments, setting this parameter to 10 iterations has been sufficient in all cases. Yet, this is not critical since the smoothing stage is not the most time-consuming one.

We thus propose various ways to control the behaviour of our algorithm. However, it should be recalled that our method is not perfect and can fail in some cases, as discussed in Section 7.4.

8 Conclusion

In this paper we have presented a method to repair meshes with combinatorial, geometrical or topological singularities, or all of them. This method is fast and produces 2-manifolds whose topology is controlled by the user. It runs in four stages:

(1) creation of a discrete volumetric model, which is a 3-manifold with boundary;
(2) application of morphological operators (openings and closings) to detect topologically critical areas;
(3) selection by the user of the areas to remove or add to the model;
(4) reconstruction of a smooth 2-manifold;

Apart from the selection of the topologically critical areas, the user can control some parameters (one for each other stage):

(1) voxelization size;
(2) strength of the morphological operators to apply;
(3) strength (number of iterations) of the surface smoothing stage.

Even if this method is better suited for smooth models because of the final smoothing stage, it has no requirement of the input model, which can be as simple as a triangle soup. A local application of this algorithm is possible, in case most parts of the input mesh do not need repair. It speeds up computation, but needs the user to cleverly select the area to repair.

Possible enhancements of this work include:

- trying to use an octree for the first stage, to speed up the computation of the final voxel set, especially in case the whole input model is selected for repair;
- investigating ways to better control the size and shape of computed topologically critical components;
- developing methods to automatically display these components from relevant viewpoints in the visualization interface;
- modifying the surface reconstruction stage in order to possibly fit some geometrical features of the input model, if the user wants to (e.g. sharp edges). This could be done by using the exact intersection point between the input model and each edge of the grid, instead of the midpoint of the edge. This can only be done when this intersection point is available and reliable; the key issue is to find when it is the case.

Acknowledgements

Part of this work was done while the first author was visiting the Universitat Politècnica de Catalunya in Barcelona with a Lavoisier grant from French Ministry of Foreign Affairs. The Buddha model is courtesy of the Stanford 3D Scanning Repository, the hip model is courtesy of Cyberware, and the pelvis, statue and brain models are courtesy of the AIM@SHAPE Digital Shape WorkBench. The authors would like to thank the anonymous reviewers for their valuable comments, which helped greatly improve the paper, and Nassim Jibai and Kartic Subr for proof-reading.

References

[1] Andújar C, Brunet P, Ayala D. Topology-reducing surface simplification using a discrete solid representation. ACM Transactions on Graphics 2002; 21(2):88–105.

[2] Andújar C, Brunet P, Chica A, Navazo I, Rossignac J, Vinacua À. Optimizing the topological and combinatorial complexity of isosurfaces. Computer-Aided

Design 2005; 37(8):847–857.

[3] Andújar C, Brunet P, Fairen M, Cebollada V. Error-Bounded Simplification of Topologically-Complex Assemblies. Workshop on Multiresolution and Geometric Modelling 2003, p. 355–366.

[4] Attene M, Falcidieno B. ReMESH: an interactive environment to edit and repair triangle meshes. Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI) 2006, p. 271-276.

[5] Attene M. A lightweight approach to repairing digitized polygon meshes. The Visual Computer 2010; 26.

[6] Barequet G, Duncan C, Kumar S. RSVP: a geometric toolkit for controlled repair of solid models. IEEE Transactions on Visualization and Computer Graphics 1998; 4(2):162–177.

[7] Bischoff S, Kobbelt L. Isosurface reconstruction with topology control. Proceedings of Pacific Graphics 2002, p. 246–255.

[8] Bischoff S, Kobbelt L. Structure preserving CAD model repair. Computer Graphics Forum (Eurographics Proceedings) 2005; 24(3):527–536.

[9] Bischoff S, Pavic D, Kobbelt L. Automatic restoration of polygon models. ACM Transactions on Graphics 2005; 24(4):1332–1352.

[10] Borodin P, Novotni M, Klein R. Progressive gap closing for mesh repairing. Advances in Modelling, Animation and Rendering, Springer-Verlag; 2002, p. 201–213.

[11] Botsch M, Pauly M, Kobbelt L, Alliez P, Levy B, Bischoff S, Rössl C. Geometric modeling based on polygonal meshes. SIGGRAPH 2007 Course Notes.

[12] Campen M, Kobbelt L. Exact and robust (self-)intersections for polygonal meshes. Computer Graphics Forum (Eurographics Proceedings) 2010; 29(3).

[13] Chernyaev EV. Marching Cubes 33: construction of topologically correct isosurfaces. Technical Report CERN CN 95-17, CERN, 1995.

[14] Cormen T, Leiserson C, Rivest R, Stein C. Introduction to algorithms, 2nd edition. MIT Press, 2001.

[15] Davis J, Marschner SR, Garr M, Levoy M. Filling holes in complex surfaces using volumetric diffusion. Proceedings of the Symposium on 3D Data Processing, Visualization, and Transmission 2002; p. 428–438.

[16] Dey TK, Guha S. Computing homology groups of simplicial complexes in $\mathbb{R}^3$. Journal of the ACM 1998; 45:266–287.

[17] Eisemann E, Décoret X. Single-pass GPU solid voxelization for real-time applications. Proceedings of Graphics Interface 2008; p. 73–80.

[18] El Sana J, Varshney A. Topology simplification for polygonal virtual environments. IEEE Transactions on Visualization and Computer Graphics 1998;4(2):133–144.

[19] Esteve J, Brunet P, Vinacua À. Approximation of a cloud of points by shrinking a discrete membrane. Computer Graphics Forum 2005; 24(4):791–807.

[20] Fleishman S, Drori I, Cohen-Or D. Bilateral mesh denoising. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2003; 22(3):950–3.

[21] Friedman J. Computing Betti numbers via combinatorial Laplacians. Algorithmica 1998; 21(4):331–346.

[22] Guéziec A, Taubin G, Lazarus F, Horn W. Cutting and stitching: converting sets of polygons to manifold surfaces. IEEE Transactions on Visualization and Computer Graphics 2001; 7(2):136–151.

[23] Guskov I, Wood Z. Topological noise removal. Proceedings of Graphics Interface 2001; p. 19–26.

[24] Haumont D, Warzée N. Complete polygonal scene voxelization. Journal of Graphics Tools 2002; 7(3):27–41.

[25] Ju T. Robust repair of polygonal models. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2004; 23(3):888–895.

[26] Ju T. Fixing geometric errors on polygonal models: a survey. Journal of Computer Science and Technology, 2009; 24(1):19–29.

[27] Ju T, Zhou QY, Hu SM. Editing the topology of 3D models by sketching. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2007; 26(3):42.

[28] Kriegeskorte N, Goebel R. An efficient algorithm for topologically correct segmentation of the cortical sheet in anatomical MR volumes. NeuroImage 2001; 14:329–346.

[29] Lai HC, Lai JY. A partial mesh replacement technique for design modification in rapid prototyping. Computers and Industrial Engineering 2009.

[30] Lee CN, Poston T, Rosenfeld A. Holes and genus of 2d and 3d digital images. Graphical Models and Image Processing 1993; 55(1):20–47.

[31] Li X, Han CY, Wee W. On surface reconstruction: a priority driven approach. Computer-Aided Design 2009;41(9):626–640.

[32] Liepa P. Filling holes in meshes. Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2003; p. 200–5.

[33] Lorensen W, Cline H. Marching cubes: a high resolution 3D surface construction algorithm. Computer Graphics (SIGGRAPH Proceedings) 1987; 21(4):163–170.

[34] Min P. [binvox] 3D mesh voxelizer. `http://www.cs.princeton.edu/~min/binvox/`. Accessed Sept. 3rd, 2009.

[35] Murali T, Funkhouser T. Consistent solid and boundary representations from arbitrary polygonal data. Proceedings of the Symposium on Interactive 3D Graphics 1997; p. 155–162.

[36] Nooruddin FS, Turk G. Simplification and repair of polygonal models using volumetric techniques. IEEE Transactions on Visualization and Computer Graphics 2003; 9(2):191–205.

[37] Pauly M, Mitra NJ, Wallner J, Pottmann H, Guibas L. Discovering structural regularity in 3D geometry. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2008; 27(3).

[38] Podolak J, Rusinkiewicz S. Atomic volumes for mesh completion. Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2005; p. 33–42.

[39] Rocchini C, Cignoni P, Ganovelli F, Montani C, Pingi P, Scopigno R. The Marching Intersections algorithm for merging range images. The Visual Computer 2004; 20:149–164.

[40] Rossignac J, Cardoze D. Matchmaker: manifold B-Reps for non-manifold r-sets. Proceedings of the ACM Symposium on Solid Modeling and Applications 1999; p. 31–41.

[41] Sharf A, Alexa M, Cohen-Or D. Context-based surface completion. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2004; 23(3):878–887.

[42] Verdera J, Caselles V, Bertalmío M, Sapiro G. Inpainting surface holes. Proceedings of the IEEE International Conference on Image Processing 2003; p. 903–6.

[43] Wood Z, Hoppe H, Desbrun M, Schröder P. Removing excess topology from isosurfaces. ACM Transactions on Graphics 2004; 23(2):190–208.

[44] Zaharescu A, Boyer E, Horaud R. TransforMesh: A topology-adaptive mesh-based approach to surface evolution. Proceedings of the Asian Conference on Computer Vision 2007; LNCS 4844:166–175.

[45] Zhou QY, Ju T, Hu SM. Topology repair of solid models using skeletons. IEEE Transactions on Visualization and Computer Graphics 2007; 13(4):675–685.