As-exact-as-possible repair of unprintable STL files

Marco Attene*
CNR-IMATI-GE

Abstract

The class of models that can be represented by STL files is larger than the class of models that can be printed using additive manufacturing technologies. In this paper such a gap is formalized while providing an unambiguous description of all the mathematical entities involved in the modeling-printing pipeline. Possible defects of an STL file are formally defined and classified, and a fully automatic procedure is described to turn any such file into a printable model. The procedure is as exact as possible, meaning that no visible distortion is introduced unless it is strictly imposed by limitations of the printing device. Thanks to such an unprecedented flexibility and accuracy, this algorithm is expected to significantly simplify the modeling-printing process, in particular within the continuously emerging non-professional “maker” communities.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

Keywords: 3D printing, mesh fix, outer hull

1 Introduction

Today fabricating an appropriate 3D model using a low-cost 3D printer is nearly as easy as printing a textual document, but creating a 3D model which is actually “appropriate” for printing is definitely complicated. Many STL files indeed have a number of defects and flaws that make them unsuitable for printing (e.g. self-intersections, zero-thickness walls, incomplete geometry, ...). In particular, the computation of a valid toolpath becomes complicated and ill-posed when the mesh does not enclose a polyhedron in an unambiguous manner.

This scenario is further complicated by the fact that a model can appear perfectly fine if visualized within a 3D viewer, but the actual presence of hidden defects prevents the possibility to print it. Experts can exploit various mesh repairing methods: a vast selection of algorithms, indeed, allows fixing virtually any sort of defect, provided that the user is able to accommodate the requirements on the input that most repairing algorithms have [Attene et al. 2013]. Unfortunately the problem is much more complex if one considers the so-called “maker movement”. This quickly-growing community is mostly made by amateurs and persons whose background is not technical enough to recognize all the defects, to select the most appropriate repairing algorithms, and to stack them into workflows where each step guarantees the right working conditions for the next step. This class of users typically needs a single repairing method which is completely automatic and does not have any specific requirement on the input. The importance and timeliness of such a scenario are confirmed by the growing interest of key actors such as Autodesk and Microsoft, which recently released popular methods to automatically repair STL files (Section 2). Their solutions, however, are not guaranteed to succeed on all the input configurations and, when they succeed, the accuracy of the fixed model is mostly suboptimal.

This paper provides a rigorous formalization of all the mathematical entities involved in the modeling-printing pipeline and, based on it, describes an automatic conversion algorithm to turn any STL file into a printable model. No assumption is made on the input STL file, the user is not forced to interact with the algorithm, and no visible distortion is introduced if it is not strictly required by the specific printing technology. To the best of the author’s knowledge no previous mesh repairing algorithm encapsulates all of these characteristics. A prototype implementation has been developed, and the results reported in section 5 show that it could accurately fix STL files that could not be properly repaired by state-of-the-art algorithms.

2 State of the art

2.1 Mesh repairing

Mesh repairing has received increased attention in recent years, not only for 3D printing, but in general for all the scenarios where a “well-behaving” mesh is required (e.g. Finite Element Analysis, advanced shape editing, quad-based remeshing, ...). Some repairing methods transform the input into an intermediate volumetric representation and construct a new mesh out of it [Ju 2004] [Bischoff et al. 2005] [Chen and Wang 2013]. In a new trend of methods specifically tailored for 3D printing, a 3D mesh is converted into an implicit representation, and all the subsequent operations (including the slicing) are performed on this representation [Huang et al. 2013] [Huang et al. 2014]. These methods are very robust but necessarily introduce a distortion. Robustness and precision are indeed major issues in this area, in particular when self-intersections must be removed [Attene 2014]. In this case some approaches rely on exact arithmetics [Hachenberger et al. 2007], while some others can losslessly convert the input into a finite precision plane-based representation, and then reconstruct a provably good fixed mesh out of it [Campen and Kobbelt 2010] [Wang and Manocha 2013]. When used for 3D printing applications, however, the aforementioned approaches are useful only if the input actually encloses a solid, while they are not really suitable to fix open meshes such as the example in Figure 1. For a more comprehensive overview of mesh repairing methods, we point the reader to [Attene et al. 2013] and [Ju 2009].

2.2 Geometry processing for 3D printing

Even if we assume that the mesh has been fixed and unambiguously encloses a solid, a further analysis may be necessary to cope with a real 3D printing scenario. For example, some parts of the model might be so thin that their physical replication would break upon removal of support structures. Such parts can be detected using volumetric analysis [Telea and Jalba 2011], and then thickened as

\*e-mail:marco.attene@ge.imati.cnr.it
We currently miss a method that can cope with any STL file while preserving all the visible surfaces with no or minimal distortions. In many other areas where a solid model is required, such as skinning and animation, simulation, surface and shape analysis.

3 Terminology and Definitions

Various authors in geometry processing literature use similar terms to indicate slightly different concepts. For example, while some papers deal with self-intersecting triangle meshes, some others refer to singularities (see Sect. 3.4).
fine a triangle mesh as a (geometric) simplicial complex, even if a simplicial complex cannot have self-intersections by definition. Since in the scope of this article there is no room for similar ambiguities, the remainder of this section formalizes a series of concepts that constitute the building blocks for the main definition of printability. This definition is then used to define a conversion algorithm that produces printable meshes out of raw triangle collections. Specifically, the following definitions conceptualize configurations that may lead to failures along the printing pipeline, and are used to progressively restrict the class of valid models. We start with all the models that can be represented by an STL file (Sec. 3.1) and put them in relation with the notion of triangle mesh (Sec. 3.2). Then, Sec. 3.3 characterizes the subset of meshes that do not self-intersect, and Sec. 3.4 further specializes the subset of non-intersecting meshes that enclose printable solids.

Triangle meshes can be encoded within several file formats, but since STL is a de facto standard and is general enough to represent any polyhedral object, this article specifically focuses on this format. Herewith we assume that the only reliable information brought by an STL file is the vertex position. Hence, our formalization intentionally disregards any additional information such as, e.g., triangle normals or vertex orientation. No specific characteristic is required on the input STL file.

In the remainder we assume that the reader is familiar with the fundamental concepts of combinatorial topology. In particular, we make use of terms such as, e.g., abstract simplicial complex, face of a simplex, simplex orientation. Basic definitions can be found in [Glaser 1970], [Stillwell 1993], and [Ferrario and Piccinini 2011].

3.1 STL Models

An STL file is essentially an unstructured collection of triangles. Some undesirable characteristics are purely syntactical (e.g. triangles with coincident vertices, duplicated triangles) and can be readily detected through a comparison of coordinates: no intermediate value must be computed and no numerical robustness issue can occur in this phase.

Definition 3.1. STL Triangle - A triplet \( < v_1, v_2, v_3 > \) with \( v_i \in \mathbb{R}^3 \) is called an STL triangle, and the \( v_i \)'s are its Vertices.

Definition 3.2. Equivalent STL Triangles - Two STL triangles are equivalent if they have the same vertices up to permutations. Example: \( < v_1, v_2, v_3 > \) and \( < v_1, v_3, v_2 > \) are equivalent.

Definition 3.3. Regular STL Triangle - An STL triangle is regular if it has three different vertices.

Definition 3.4. STL Model - A finite collection of STL triangles \( T = \{ t_i = < v_{i1}, v_{i2}, v_{i3} >, v_{ij} \in \mathbb{R}^3 \} \) is an STL model (or simply an STL), and the subset of \( \mathbb{R}^3 \) made by the union of all the \( v_{ij} \)'s is its vertex set and is denoted by \( V(T) \). Example: \( T = \{ < 1, 0, 0 >, < 0, 1, 0 >, < 0, 0, 0 >, < 1, 1, 0 > \} \) and \( V(T) = \{ 1, 0, 0 >, < 0, 1, 0 >, < 0, 0, 0 >, < 1, 1, 0 > \} \).

Definition 3.5. Regular STL Model - An STL model is regular if all its triangles are regular.

Definition 3.6. Layer - Let \( T \) be an STL model. Any maximal subset of non-equivalent triangles in \( T \) is called a Layer of \( T \) and is denoted by \( L(T) \). Note that \( V(L(T)) = V(T) \).

3.2 Meshes

A triangle mesh is a set of triangles with an explicit structure. Such a structure, or connectivity, takes the form of an abstract simplicial complex and is independent of the vertex position. When the complex is realized in the Euclidean space, some degenerate cases may occur, and particular care is necessary to detect them when using floating point arithmetic [Shewchuk 1997].

Definition 3.7. Triangle Mesh - A triangle mesh is a pair \( M = (V, S) \), where \( V \) is a set of points in \( \mathbb{R}^3 \), \( S \) is a pure two-dimensional abstract simplicial complex [Glaser 1970], and there is a bijective map between the set of vertices of \( S \) and \( V \). Such a map is called vertex embedding and is denoted as \( \varphi : S_0 \rightarrow V \), where \( S_0 \) denotes the set of vertices of \( S \). Thus, any 0-simplex \( s \) in \( S \) is mapped to exactly one of the points in \( V \) through the vertex embedding. Notice that \( S \) is not necessarily a combinatorial manifold. Since this article does not deal with non-triangular meshes, in the remainder we shall omit the qualifying term “triangle”.

Definition 3.8. Closed Mesh - A mesh \( M = (V, S) \) is closed if \( S \) has no boundary, that is, if any 1-simplex in \( S \) is a face of an even number of 2-simplices in \( S \) [Stillwell 1993].

Definition 3.9. STL-induced Mesh - Let \( T \) be a regular STL and let \( L(T) \) be one of its layers. Also, let \( M = (V(T), S(T)) \) be a mesh where \( V(T) \) is the vertex set of \( T \) and \( S(T) \) is obtained by considering each triangle \( t_i \) in \( L(T) \) as an abstract 2-simplex whose vertices are mapped to \( t_i \)'s vertices. \( M \) is called a \( T \)-induced mesh.

Definition 3.10. Geometric Realization - Let \( M = (V, S) \) be a mesh. The geometric realization of a k-simplex \( s = \{ v_0, ..., v_k \} \in S \) is the convex hull of the points \( \varphi(v_0), ..., \varphi(v_k) \) and is denoted by \( |s| \). Thus, any point in \( |s| \) can be expressed as \( \lambda_0 \varphi(v_0) + ... + \lambda_k \varphi(v_k) \), \( 1 \geq \lambda_i \geq 0 \), and \( \lambda_1 + ... + \lambda_k = 1 \). The union of the geometric realization of all the simplices is the geometric realization of the mesh, denoted by \( |M| \).

Definition 3.11. Degenerate Realization - Let \( M = (V, S) \) be a mesh. The geometric realization of a 2-simplex in \( S \) is a properly 2-dimensional subset of \( \mathbb{R}^3 \) only if its three vertices are mapped to points in \( V \) which are in general position (i.e. the three points are neither coincident nor collinear). In all the other cases the geometric realization is said to be degenerate.

Definition 3.12. Self-intersection - Let \( M = (V, S) \) be a mesh and let \( s_1 \) and \( s_2 \) be simplexes in \( S \). Also, let \( X = \) the intersection of \( |s_1| \) and \( |s_2| \). If \( X \) is not empty and \( X \) is not the geometric realization of a simplex in \( S \) which is a face of both \( s_1 \) and \( s_2 \), then \( X \) is called a self-intersection of \( M \).

Proposition 3.13. Any 2-simplex with a degenerate realization leads to self-intersections.

Proof: Let \( s = \{ v_1, v_2, v_3 \} \in S \) be such a simplex. \( |v_1|, |v_2| \) and \( |v_3| \) are not affinely independent, hence one of them can be expressed as linear combination of the other two. Without loss of generality, let us say that \( |v_1| = \lambda_1 |v_2| + \lambda_2 |v_3| \) where \( 1 > \lambda_1 \geq 0 \) and \( \lambda_2 = 1 \). Let us say that \( s_1 = \{ v_1 \} \) and \( s_2 = \{ v_2, v_3 \} \), and let \( X = |s_1| \cap |s_2| \). We observe that \( X = |v_1| \), but \( v_1 \) is not a face of \( s_2 \).

3.3 Polyhedra

Definition 3.14. Polyhedron - A polyhedron is the geometric realization of a mesh \( M \) without self-intersections. In this case the geometric realizations of the simplexes in \( M \) are Euclidean simplexes forming an Euclidean simplicial complex [Ferrario and Piccinini 2011]. We observe that a polyhedron is not necessarily a two-manifold with boundary. Also, due to Proposition 3.13, a mesh with degenerate elements does not admit a polyhedron.

Definition 3.15. Closed Polyhedron - A polyhedron \( |M| \) is closed if the non-self-intersecting mesh \( M \) is also closed.

The concept of outer hull is central in this work, therefore it is worth providing a more accessible intuitive introduction of such an object before giving its formal definition. Roughly speaking, the outer hull of a polyhedron is the subset of its points which can be reached from infinity through continuous paths. In other words, we may loosely say that the outer hull is made by the points of the polyhedron which are visible from the outside. Furthermore, if a point of the outer hull can be reached from one side only, then it is part of the so-called...
solid outer hull.

**Definition 3.16. Outer Hull** - Let $|M|$ be a polyhedron and let $p \in \mathbb{R}^3$ be a point out of it (e.g. if $v$ is the vertex in $V$ having the maximum $x$ coordinate, take $p = v+ <1,0,0>$). Let $|M'| \subseteq |M|$ be the set of points $q$ for which there exists a path $<q,p>$ which does not contain any other point of $|M|$ (excluding $q$ itself). Then $|M'|$ is the outer hull of $|M|$. The outer hull is a polyhedron.

**Definition 3.17. Solid Outer Hull** - Let $|M|$ be a polyhedron and let $|M'|$ be its outer hull. With reference to the above definition 3.16, let $O$ be the union of $|M'|$ and all the points in space which are path-connected with $p$ on paths that do not contain points of $|M|$. The Solid Outer Hull $M_{solid}$ of $M$ is the boundary of $O$.

## 3.4 Printability Condition

We wish to classify the set of STL files that unambiguously represent a single *sufficiently connected* solid object. Intuitively, two pyramids that touch at their apex only are not sufficiently connected (i.e. the small connection would either not be printed at all or would be too weak in a structural sense). Though it is tempting to model this class of objects as the class of 2-manifolds, there are cases where singularities occur though the model is solid enough (Fig. 2 (c) and (d)). Notice that these cases cannot be split in parts to be printed separately as one could do with the aforementioned two pyramids. This intuitive concept is formalized by the notion of manifold-connectedness given in Def. 3.18.

Any two-dimensional abstract simplicial complex with singularities (i.e. non-manifold vertices/edges) can be decomposed into a collection of manifold complexes by properly duplicating singular simplexes. This duplication procedure is purely combinatorial and does not consider possible vertex embeddings. The resulting set of combinatorial manifolds is provably unique and is called a *manifold decomposition* of the input complex [Hui et al. 2006]. Without loss of generality, if we assume to deal with a single connected complex with singularities, its manifold decomposition may be either a set of $N > 1$ complexes (e.g. Fig. 2 (a) and (b)) or another single connected complex (e.g. Fig. 2 (c) and (d)).

**Definition 3.18. Manifold-connected Polyhedron** - Let $|M|$ be a polyhedron. $|M|$ is manifold-connected if the manifold decomposition of $M$ is made of a single component.

**Definition 3.19. Printable STL** - An STL model $T$ is printable if there exists a $T$-induced mesh whose realization is a closed and manifold-connected polyhedron that coincides with its outer hull.

This manifold-connectedness constraint is necessary to ensure that a connected model remains connected even in its physical counterpart.

## 4 Conversion Algorithm

An arbitrary STL model must undergo a sequence of checks and possible repairing operations to be guaranteed to be printable according to Definition 3.19. Before entering the details of the repairing process, it is worth providing a high-level classification of all the possible issues that make an STL model not suitable for printing. We distinguish among defects and limitations of (1) the representation, (2) the surface, or (3) the printing device. Representational defects are related to the particular STL file used to encode the surface to be printed; in other words, the visible outer surface itself may be well defined and may unambiguously enclose a solid, but the STL model has issues to be resolved (e.g. degenerate triangles, self-intersections). Representational defects can be fixed without any visible distortion of the outer shape. Conversely, defects of the surface are independent of the representation, and make the intended solid different from the actually enclosed solid. For example, a 3D model designer may represent very thin parts of the object through idealized zero-thickness surfaces, such as for the helmet and cloak in Figure 1. Even after the resolution of self-intersections, such a geometry does not enclose a solid. In these cases of weak design of the model, we must either ask the designer to disambiguate, or guess the intended geometry and perform the necessarily visible distortion. Finally, the model might be perfectly well defined with neither representational nor design defects, but it can still be not suitable for printing due to an incompatibility with the specific printer to be used (e.g. too thin walls, too tiny features, size larger than printing volume). We will not deal with these issues here, but existing works tackle the problem for specific printing devices and scenarios [Pintus et al. 2010] [Luo et al. 2012].

### 4.1 Algorithm overview

In the remainder, our only assumption is that the STL file is syntactically well-formed: no other requirement is necessary. The algorithm first deletes possibly irregular and degenerate triangles and then extracts a layer. The induced mesh of such a pre-filtered STL must be checked for self-intersections. If intersections exist, the mesh must be split accordingly while taking care of not creating new irregular or equivalent triangles. This initial procedure is mainly based on existing methods [Attene 2014] and is briefly described in Sec. 4.2. At this stage the STL model has an induced mesh which might be non manifold but admits a polyhedron. While computing the outer hull $|M|$, the algorithm also splits the outer hull itself in two subsets: $M_{solid}$ and $M_{sheet}$, where $M_{solid}$ represents the union of all the parts of $|M|$ that enclose a volume (see Def. 3.17), and $M_{sheet}$ is the union of all the remaining sheet-like parts of $|M|$ (Sec. 4.3, Algorithm 1). If $M_{sheet}$ is not empty, each component in $M_{sheet}$ is transformed to a thin solid through a thinning procedure (Sec. 4.3.2) and the whole process is repeated. Note that $M_{sheet}$ is necessarily empty on the second iteration. In a last polishing phase, each of the printable components in $M_{solid}$ is isolated (Sec. 4.3.1) and the algorithm terminates.

Figure 2: All these meshes have singularities that make them non manifold. (c) can be obtained from a cylinder by pushing the central points of its two bases towards the same point in the center of the cylinder. For this reason, this configuration is somewhere called a "pinched pie" [Hui et al. 2006]. (d) can be obtained as for (c), but instead of pushing two points we push two edges towards a single common edge. From a topological perspective, (c) and (d) can be seen as the complementary of (a) and (b) respectively, but only (c) and (d) are manifold-connected.
4.2 From STL files to polyhedra

After having deleted all the irregular and degenerate triangles we can easily extract a layer by simply removing possible equivalent triangles. To do this efficiently, we pre-sort all the triangles in lexicographical order so that equivalent triangles are contiguous in the sorted list. The remaining triangles are used to create an induced mesh. If such a mesh has no self-intersections, then it is a valid Euclidean simplicial complex [Ferrario and Piccinini 2011] whose realization is a polyhedron. In other words, the intersection of any two non-disjoint triangles is either an edge or a vertex. Conversely, if self-intersections occur in the induced mesh, new simplices must be created to represent them in the abstract complex too. An efficient approach to perform this operation is described in [Attene 2014]. Other methods include the publicly available software Cork [Bernstein 2013] which is extremely fast but does not guarantee a border edge, the whole mesh. If such a mesh has no self-intersections, then it is a valid Euclidean simplicial complex [Ferrario and Piccinini 2011] whose realization is a polyhedron. In other words, the intersection of any two non-disjoint triangles is either an edge or a vertex. Conversely, if self-intersections occur in the induced mesh, new simplices must be created to represent them in the abstract complex too. An efficient approach to perform this operation is described in [Attene 2014]. Other methods include the publicly available software Cork [Bernstein 2013] which is extremely fast but does not guarantee an intersection-free output in all the cases, and the library LibIGL [Panozzo and Jacobson 2014], which uses Cork as long as it is robust enough and switches to exact arithmetic for the most difficult cases. In any case, the result of such an operation is expected to be a possibly non-manifold mesh without intersections.

4.3 From polyhedra to printable polyhedra

This section describes how to turn a non-manifold simplicial complex into a set of printable models. After a first intuitive description of the process given herebelow, formal definitions are given in Sec. 4.3.1 and used to define the outer hull algorithm in Sec. 4.3.2.

Intuitively, the outer hull $|M|$ can be extracted through a region growing approach: a triangle which is known to be on the outer hull is selected, and from such a seed the remaining parts can be reached by adjacency while staying on the outer side of the complex. Let us see how this procedure works by starting from the simplest case and progressively adding degrees of freedom to the input. If the input complex is a single closed 2-manifold such a procedure reaches all the triangles, and the result is obviously printable according to Def. 3.19. If the complex is a collection of disjoint closed 2-manifolds, the procedure extracts one of the printable components only. Hence we may remove this component from the complex and repeat the procedure to extract a second component, and so on. In the end of the process, each of the so-extracted components which is spatially contained in other components is removed, and the remaining forms a collection of printable models corresponding to the outer hull of the input. If we allow singular vertices, but not singular edges, the aforementioned procedure can still be employed, though particular care must be taken when removing a component from the complex because such a removal might require to create a copy of a singular vertex (e.g. when extracting the first pyramid in Fig. 2(a)).

The last degree of freedom to be unlocked is the possibility to have singular edges and "border" edges (i.e. with only one incident triangle). In this case things become much more complicated because the outer hull might be no longer a collection of printable components and unorientable parts might come into play. Thus, while tracking the outer hull $|M|$, we need to recognize which of its subsets form closed polyhedra ($M_{solid}$) and which other subsets require a thickening ($M_{sheet}$). When our region grows reaching a border edge, the whole patch containing that edge is moved to $M_{sheet}$ and the process restarts (an exact definition of such patches is given in Sec. 4.3.2). In this way, all the dangling open patches attached to the outer hull can be removed one by one. However, this is still not sufficient because, although the outer hull $|M|$ has no longer border edges, it might still contain patches that do not bound any solid (i.e. these patches are connected to other parts of $|M|$ through singular edges). When dealing with singular edges a correct orientation of the region being grown becomes crucial. Hence, at the beginning of the process we first orient the seed, and then propagate such an orientation as the region grows across edges. When a singular edge is encountered, we grow on the triangle which is the "most external" according to the orientation of the triangle $t$ we are coming from: using a metaphor, an ant walking on the outer side of $t$ towards the singular edge would proceed onto such a "most external" triangle. If different propagation directions induce opposite orientations on a same triangle, that triangle and the whole patch containing it are moved to $M_{sheet}$ and the process restarts. This situation, indeed, happens when single sheets of triangles that do not bound any solid are surrounded by other parts of $|M|$ that do. Using this technique, we iteratively build both $M_{sheet}$ and $M_{solid}$. Indeed, if during the growing neither border edges nor incompatible orientations are encountered, the process tracks a closed polyhedron $P$ which can be moved to $M_{solid}$. The triangles of the input complex which are not part of $P$ though being edge-connected with it (i.e. they are spatially contained in $P$) can be safely removed because they are not part of the outer hull. Also, it might happen that $P$ is not manifold-connected (e.g. the growing would cover the whole model in Fig. 2(b)), therefore it might be necessary to split it into its manifold-connected parts [Rossignac and Cardoza 1999]. The following two subsections formalize the aforementioned intuitive concepts and procedure.

4.3.1 Clustered Polyhedra

In general, depending on the number of its incident triangles, an edge can be:

- on boundary (only one incident triangle);
- 2-connected (exactly two incident triangles);
- singular (more than two incident triangles).

**Definition 4.1. Triangle Fan** Let $e$ be an edge, let $T(e)$ be the set of all its incident triangles, and let $t_0$ be one triangle in $T(e)$. A triangle fan at $e$ from $t_0$ is an ordered list $Fan(e, t_0) = \langle t_0, t_1, ..., t_n \rangle$ whose elements are all and only the triangles in $T(e)$, $t_0$ is the first element, and the triangles preserve their radial order around $e$. When $e$ is singular, the pair $(e, t_0)$ admits two triangle fans that can be distinguished by the radial order direction (clockwise and counterclockwise).

Let us assume that the normal $n$ at $t_0$ is well-defined and reliable, and let $v$ be the vertex of $t_0$ that does not belong to $e$. A triangle fan is upward with $t_0$ if $t_0$ is the only element in the fan or if the second element is the first triangle after $t_0$ when turning around $e$ in the direction specified by the vector $n$ applied on $v$. Hence, under the just mentioned assumptions, a pair $(e, t_0)$ has exactly one upward triangle fan and possibly one downward fan (non-upward, if $e$ is singular).

**Definition 4.2. Continuation** - Let $e$ be a non boundary edge, and let $t$ be a triangle of $Fan(e, t_0)$. The continuation of $t$ in $Fan$ is the first triangle after $t$ if $t$ is not the last element of the list, or the first element of $Fan$ if $t$ is the last element.

**Definition 4.3. Edge-connected Polyhedron** - Let $|M|$ be a polyhedron. $|M|$ is edge-connected if any pair of 2-simplexes in $M$ is edge-connected. Two 2-simplexes $s_i$ and $s_j$ are edge-connected if there exists a sequence $s_{a_0} = s_1, ..., s_{a_k} = s_0$ such that $s_{a_{k-1}}$ and $s_{a_k}$ intersect at a common 1-simplex [Hui et al. 2006].

Our algorithm tracks each edge-connected component of the outer hull by starting from a seed triangle. The seed must be guaranteed to be part of the outer hull and its correct orientation/normal must be known without ambiguity (i.e. we must know exactly on which of its two sides we can find the solid). Thus, our first operation is to detect and orient the seed correctly. We select a starting extreme vertex $v_{0}$ as the one having the maximum $x$ coordinate. We then
pick all the edges incident at \( v_0 \) and select the one (let it be \( e_0 \)) whose normalized vector \( n = < n_x, n_y, n_z > \) has the smallest absolute value \( n_z \). Finally, we pick all the triangles incident at \( e_0 \) and select the one (let it be \( t_0 \)) whose normal \( n = < n_x, n_y, n_z > \) has the largest absolute value \( n_z \). If \( n_z \) is negative, we invert the orientation of \( t_0 \).

### 4.3.2 Outer hull extraction algorithm

As mentioned in Sec. 4.3, to treat all the possible cases our algorithm extracts both \( M_{solid} \) and \( M_{sheet} \), and each of these two subsets is made of patches. Each such patch is a cluster of properly connected triangles. Thus, in a first step the abstract simplicial complex is partitioned into these clusters of triangles. These clusters are then virtually connected within an adjacency graph and, based on both their connectivity and their orientability, they are partitioned in three subsets so that, at the end of the process, a cluster can be part of \( M_{solid} \), part of \( M_{sheet} \), or internal (Figure 3).

![Figure 3: A Face Cluster can be part of \( M_{solid} \), part of \( M_{sheet} \), or internal. Internal clusters are deleted, whereas the other two sets form the outer hull.](image)

**Definition 4.4. Face Cluster** - Two triangles \( s_0 \) and \( s_0 \) are 2-connected if there exists a sequence \( s_0 = s_1, ..., s_n = s_n \) such that \( s_{i-1} \) and \( s_i \) share a common 2-connected edge. A set of triangles is 2-connected if any pair of triangles in the set is 2-connected. Let \( |M| \) be a polyhedron. A Face Cluster of \( M \) is a maximal 2-connected subset of triangles of \( M \).

Thus, any pair of triangles in a Face Cluster is 2-connected and, due to maximality, if \( t_1 \) is part of the Face Cluster and \( t_2 \) is 2-connected with \( t_1 \), then \( t_2 \) is part of the Face Cluster too. Note that two triangles can be edge-connected according to Definition 4.3 while not being 2-connected. Conversely, any pair of 2-connected triangles is also edge-connected.

Let us assume that triangles in \( M \) have an orientation. A Face Cluster is oriented if any pair of its 2-adjacent triangles (i.e., having a common 2-connected edge) have a consistent orientation. A Face Cluster is orientable if it is possible to assign an orientation to all of its triangles so that the Face Cluster becomes oriented.

Note that a Face Cluster can be unorientable and non-manifold.

**Definition 4.5. Clustered Polyhedron** - Any polyhedron \( |M| \) can be partitioned into a set of Face Clusters. In such a clustered polyhedron, we say that each triangle belongs to exactly one Face Cluster. If \( t \) belongs to the Face Cluster \( C \), we write \( C = C(t) \).

**Definition 4.6. Homeomorphic Triangle Fans** - On a clustered polyhedron, we say that two triangle fans \( F = < f_1, ..., f_n > \) and \( G = < g_1, ..., g_n > \) are homeomorphic if their triangles belong to the same Face Clusters in the same order. Hence, \( F \) is homeomorphic with \( G \) if \( C(f_1) = C(g_1), C(f_2) = C(g_2), ..., C(f_n) = C(g_n) \).

**Definition 4.7. Cluster Wall** - Let \( |M| \) be a Clustered Polyhedron where each edge is colored "green" if it is 2-connected and "red" otherwise. Also, let us assume that any vertex being an endpoint of a red edge is colored "green" if it has exactly two incident red edges and "red" otherwise. Let \( C \) be an oriented Face Cluster and let \( t_a \) and \( t_b \) be two of its triangles having at least a red edge each, say \( e_a \) and \( e_b \) respectively. We say that the pairs \( (t_a, e_a) \) and \( (t_b, e_b) \) are co-wall if all the following conditions hold:

- \( e_a \) and \( e_b \) are different but they are incident upon a common green vertex \( v \);
- there exists a sequence of \( n \geq 1 \) triangles \( t_1 = t_a, ..., t_n = t_b \) such that any pair \( (t_i, t_{i+1}) \) shares a common green edge incident at \( v \);
- The upward triangle fans \( Fan(t_a, e_a) \) and \( Fan(t_b, e_b) \) are homeomorphic.

A Cluster Wall \( W \) of \( C \) is a sequence of pairs \( W = \{ p_1 = (t_1, e_1), ..., p_n = (t_n, e_n) \} \) such that \( C(t_i) = C(e_i) \) is a red edge of \( t_i \), and \( p_i \) and \( p_{i+1} \) are co-wall for any \( 1 \leq i < n \).

The set of the \( e_i \)'s is the side of the Cluster Wall. Two Cluster Walls are matching if they have the same side.

**Definition 4.8. Cluster Edge** - A Cluster Edge is an equivalence class of matching Cluster Walls. The side of a Cluster Edge is the side of its Cluster Walls.

An oriented Face Cluster is bounded by a number \( n \geq 0 \) of Cluster Walls, each corresponding to a Cluster Edge. Different Cluster Walls in a same Face Cluster may correspond to the same Cluster Edge (e.g., Figure 4(d)). Note that the notion of Cluster Wall is not defined for unorientable Face Clusters. Various Face Cluster configurations are shown in Figure 4.

A Cluster Edge inherits the classification of the edges in its side. Thus, if these edges are on boundary, then the Cluster Edge is also on boundary. Otherwise, if these edges are singular, then the Cluster Edge is singular. Since side edges are not 2-connected by definition, a Cluster Edge cannot be 2-connected.

We say that an oriented Face Cluster is on boundary if any of its Cluster Edges is on boundary. Also, we recall that any unorientable surface with a valid (i.e., non self-intersecting) embedding in \( R^3 \) has a boundary [Griffiths 1976]. Thus, we say that an arbitrary Face Cluster is on boundary if either it is unorientable or any of its Cluster Edges is on boundary.

**Definition 4.9. Continuation** - The notion of continuation defined for triangles (Definition 4.2) is inherited by oriented Face Clusters. Let \( C \) be an oriented Face Cluster, let \( W \) be one of its singular Cluster Walls (i.e., it has a singular Cluster Edge), and let \( (t, e) \) be one of the pairs in \( W \). Let \( t' \) be the continuation of \( t \) in the upward triangle fan \( Fan(e, t) \). The continuation of \( C \) at \( W \) is the Face Cluster \( C' = C(t') \). \( C' \) and \( C \) are consistently oriented if all their matching triangles are consistently oriented. Two triangles \( t' \in C' \) and \( t \in C \) are matching if they have a common edge \( e \) and \( (t, e) \in W \).

Note that in some cases it might happen that \( C = C' \) (e.g., Figure 4(d)).

Based on the just defined concepts, our procedure can be summarized by Algorithm 1.

The rational behind this algorithm is to walk on the outer side of the polyhedron as long as possible. If the while cycle terminates, it means that the visited clusters constitute a closed polyhedron. Conversely, if during the visit either boundaries or incompatible
orientations are encountered, the corresponding sheet-like clusters are added to $M_{\text{sheet}}$. Notice that only the visible sheets are actually added to $M_{\text{sheet}}$, whereas other possible sheets which are enclosed by a volume are never reached. The containment check required to delete each internal Face Cluster is reduced to a single point-in-polyhedron test. Indeed, with reference to line 43 in the algorithm, we just take the barycenter of one of $C'$ triangles and determine if this point is contained in $D$. A hint to understand the link between this algorithm and Def. 3.17 is given in Appendix 6.

In the example of Fig. 5, we start with cluster number 1 which is not on boundary: this cluster has six Walls, two for each of its three adjacent clusters. We select one of these Walls and propagate onto cluster 2. Since cluster 2 is on boundary it is moved to $M_{\text{sheet}}$ and the process restarts from cluster 1. Now we have only four Walls, and propagate on cluster 3 which is moved to $M_{\text{sheet}}$ as for cluster 2. On the third iteration we have two Walls left on cluster 1, and propagate on cluster 4 across one of these Walls. Cluster 4 is not on boundary, and is oriented according to the Wall we are coming from. Then we select the other Wall on cluster 1 and propagate, again, on cluster 4. This time, however, cluster 4 is already visited, and the existing orientation is incompatible with the Wall we are coming from. For this reason, cluster 4 is moved to $M_{\text{solid}}$. At this point the first component of $M_{\text{solid}}$ is complete, and we look for another seed to track the other component. Cluster 5 has two Walls, but both of them propagate onto the same cluster 5 which is already visited. The third seed belongs to cluster 6 which is on boundary and hence it is initially moved to $M_{\text{sheet}}$. However, in the end of the algorithm this cluster is removed because it is contained in the space bounded by the sphere.

It might happen that $M_{\text{sheet}}$ is empty, meaning that the polyhedron has a closed outer hull. In this case, Algorithm 1 provides a nearly-ready solution to our problem because $M_{\text{solid}}$ is the union of manifold-connected components, each printable according to definition 3.19. It might happen that the resulting components share singular vertices or edges, but any such component can be isolated by duplicating these singularities in the abstract complex [Rossignac and Cardoze 1999]. Such a duplication allows separating the manifold-connected components without modifying the geometry of the objects, and each component can be saved to a separate STL file. Each of these STL files represents a printable object according to Definition 3.19, and this object is manifold-connected though not necessarily manifold (e.g. Fig. 2). Hence, each of the meshes in Fig. 2 (a) and (b) would be represented by two files, whereas each of the meshes in Fig. 2 (c) and (d) would be encoded in a single file.

On the other extreme, $M_{\text{solid}}$ might be empty, meaning that the polyhedron does not enclose any volume. Since our objective is to convert any possible STL file into a printable model, the case where $M_{\text{sheet}}$ is not empty requires a further elaboration. We observe that the presence of a non-empty $M_{\text{sheet}}$ must be considered a defect of the surface, and not just a representational defect. Hence, it is necessary to introduce a distortion to fix the problem; note that this corresponds to the practical observation that no real printer can produce a zero-thickness surface.

To solve these issues we consider the minimum thickness $\epsilon$ that the target printer is able to actually build. Then, we take each orientable Face Cluster $C$ in $M_{\text{sheet}}$, orient it in an arbitrary direction, and produce a copy $C'$ with an inverted orientation. $C$ and $C'$ are then stitched along their common boundary, and the resulting closed mesh is inflated through an offsetting at distance $\epsilon/2$ [Qu and Stucker 2003]. Possible unorientable Face Clusters are properly cut to make them orientable [Attene and Falcidieno 2006] before producing $C'$. The union of $M_{\text{solid}}$ and the so-inflated $M_{\text{sheet}}$ undergoes the whole repairing process once more, but this time we are guaranteed that Algorithm 1 will produce an empty $M_{\text{sheet}}$.

Overall, this algorithm guarantees that all the solid parts are fixed without any distortion, whereas a minimum modification is introduced to fix sheet-like parts. Possible solid parts which are thinner than $\epsilon$ can be fixed in a second step through [Wang and Chen 2013]. Essentially, as exact as possible repairing means that if $\epsilon$ is the radius of the filament, the visible part of the repaired model is identical to the visible part of the input in all the points that belong to $M_{\text{solid}}$, whereas it is within an $\epsilon$ tolerance in all the other visible points. Such an as exact as possible repairing of a polyhedron is $M_{\text{solid}}(M_{\text{solid}}(M) \cup \text{Inflated}_\epsilon(M_{\text{sheet}}(M)))$.

For the sake of simplicity, the treatment of visible wire-like features has been omitted in Algorithm 1. Though in principle an STL file should not represent isolated edges, however, wire-like features can be defined as either irregular triangles or triangles having a degenerate realization. To treat these cases, we store all the removed degenerate and irregular triangles as segments in a separated list $D_t$. After the execution of Algorithm 1, each segment in $D_t$ is analyzed: if the entire segment is part of the surface of either $M_{\text{solid}}$ or $M_{\text{sheet}}$, or if it is entirely contained in the volume enclosed by $M_{\text{solid}}$, then it is discarded; otherwise it is inflated into an $\epsilon$-diameter cylinder and undergoes the second iteration of the repairing process along with the inflated $M_{\text{sheet}}$.

5 Results and discussion

A prototype of the repairing algorithm described so far has been implemented in C++ within a Windows 7 environment. All the experiments reported in this paper were run on a standard 2.67 GHz Intel Core i7 PC with 6 Gh RAM. After removal of irregular and equivalent triangles, in the prototype mesh simplexes are subdivided as described in [Attene 2014]: since this approach may create new equivalent triangles (e.g. on exactly coplanar but non equivalent input triangles), an additional post-filtering is performed to remove them. To guarantee a robust though efficient ordering of the triangle fans, fast exact geometric predicates [Shewchuk 1997] are used to sort the planes based on original coordinates only; newly-inserted vertices are not used for this operation. The resulting non-manifold
complex is stored in a proper data structure [De Floriani et al. 2004] and used as input for Algorithm 1.

The prototype implementation was first run on some challenging models (Figure 6) on which state-of-the-art algorithms fail or give too rough results. Then, experiments were run on all the 1814 models of the Princeton Shape Benchmark [Shilane et al. 2004]. The prototype succeeded in all the cases, and the repairing proceeded at an average speed of 11K triangles per second. The speed depends on a number of factors, but the total elapsed time is largely dominated by the self-intersection removal phase: all the aspects that determine the efficiency for this phase are already described in [Attene 2014] where results are reported for models made of up to millions of triangles. Herewith we just observe that the self-intersection removal must be run twice if $M_{\text{sheet}}$ is not empty, and the second iteration usually deals with more triangles. Hence, during the experimentation we also measured the average speed on models with empty $M_{\text{sheet}}$ (16K tri/sec) and on all the other models where two iterations were required (5K tri/sec). All the models were first scaled to have a bounding box maximum extension equal to 100 mm, and a value of 0.4 mm was used for $\epsilon$ (this is the extruder nozzle diameter of the 3D printer used for the experiments).

Nowadays NetFABB [Microsoft and NetFABB 2013] and Meshmixer [Autodesk 2011] are probably among the most popular and powerful systems to repair models for 3D printing applications; it is therefore worth comparing our method against these two approaches. Our evaluation protocol rates each method according to the following scoring table:

- Score = 1: Failure. The software did not produce an output at all, or produced something which is unprintable or completely different from the input (e.g. NetFABB creates a single tetrahedron out of the model in Figure 6);
- Score = 2: Incomplete. The output is printable and resembles the input enough, but some of the parts which were visible in the input are no longer visible;
- Score = 3: Distorted. The output is printable and all the visible parts of the input are represented, but the output is unnecessarily distorted;
- Score = 4: As-exact-as-possible. The output is printable and all the visible parts of the input are represented exactly or with a minimum distortion due to the specific 3D printer at hand;
- Score = 5: Exact. The output is printable and exactly represents all the visible parts of the input.

Based on such a protocol, the behavior of each algorithm could be rated on a per-model basis. As shown in Table 1, our method outperforms the competing algorithms in most difficult cases.

Unfortunately, NetFABB is run remotely on a proprietary hardware whose characteristics are not public. Hence, computational efficiency could be fairly compared against MeshMixer only. The "Repair selected" function provided by MeshMixer v10.9.246 was used, and in all the tests the default parameter setting for MakerBot printers was employed. Among the hard models with a non empty $M_{\text{sheet}}$, in our dataset the Darth Vader is the most complex (15294 triangles), and MeshMixer took 110 seconds to produce the fixed version shown in Figure 1. On the same computer, our algorithm needed less than 5 seconds. Furthermore, it is worth mentioning that the output model produced by MeshMixer is made of more than one million triangles, whereas our result is much sim-
null
follows: the input STL undergoes the repairing process up to the procedure inspired on [Bischoff and Kobbelt 2005], and works as expected. The prototype exploits a stitching approach, that is run once again. To demonstrate this approach, a prototype interactive system has been implemented so that inaccurately-modeled objects (e.g. Fig. 8) could be repaired as expected. In a final step, the selected areas are disconnected, we mark all the eight cells in \( C \) as potential stitches; if not, the same check is repeated on each of the eight blocks of cells that share one of \( C \)'s vertices; specifically, if \( v \) is one of the eight vertices of \( C \), we consider the block \( C' \) made of the eight cells incident at \( v \), and if the mesh restricted to \( C' \) is disconnected, we mark all the eight cells in \( C' \) as potential stitches. The collection of all the potential stitches is displayed, and the user selects those that must lead to an actual stitch. These selected regions are triangulated, that is, each square forming the boundary of the region is split into two triangles. These new triangles are added to the mesh and the repairing is launched once again, but this time the thickening process and the second iteration are included. During this last repairing step, however, triangles which are part of the stitching areas are kept selected, and their subtriangles (i.e. those triangles that are generated by resolving their intersections) inherit such a selection. In a final step, the selected areas are smoothed by iteratively moving their internal vertices towards their respective centers of mass. This process is summarized in Fig. 11. It is first iteration of Algorithm 1. If \( M_{\text{sheet}} \) is empty we already have the result and no ambiguity requires user interaction. Otherwise, the user is warned and asked to set a value for \( \gamma \). A uniform grid is created to intersect model with cubical cells of size \( \gamma/2 \). Each cell \( C \) which is intersected by at least one boundary curve is analyzed: if the mesh restricted to the volume of \( C \) is disconnected (i.e. \((M_{\text{solid}} \cup M_{\text{sheet}}) \cap C \) is disconnected), then \( C \) is marked as potential stitch; if not, the same check is repeated on each of the eight blocks of cells that share one of \( C \)'s vertices; specifically, if \( v \) is one of the eight vertices of \( C \), we consider the block \( C' \) made of the eight cells incident at \( v \), and if the mesh restricted to \( C' \) is disconnected, we mark all the eight cells in \( C' \) as potential stitches. The collection of all the potential stitches is displayed, and the user selects those that must lead to an actual stitch. These selected regions are triangulated, that is, each square forming the boundary of the region is split into two triangles. These new triangles are added to the mesh and the repairing is launched once again, but this time the thickening process and the second iteration are included. During this last repairing step, however, triangles which are part of the stitching areas are kept selected, and their subtriangles (i.e. those triangles that are generated by resolving their intersections) inherit such a selection. In a final step, the selected areas are smoothed by iteratively moving their internal vertices towards their respective centers of mass. This process is summarized in Fig. 11. It is

Figure 9: An original STL file with open surfaces used to model sheet-like features (a) and three repaired versions produced by NetFABB cloud service (b), Autodesk Meshmixer (c), and our algorithm (d). On the right (e), a physical replica of the model in (d) is shown.

Figure 10: A tessellated model of a Klein bottle (a). This model is not embeddable in \( R^3 \), hence its surface is necessarily self-intersecting (red triangles). Our method could cut and reorient the surface without distortions (b), so that the model could be successfully printed (c).

5.1 User interaction

The repairing algorithm described in this paper can be part of more generic systems which involve the user in the ill-posed task of distinguishing between intentional thin sheets and unintentional disconnections in the outer surface. Such a system may run the repairing process once and, before thickening \( M_{\text{sheet}} \), may use a threshold distance \( \gamma > \epsilon \) to determine which portions of the boundary could be stitched to other parts of the surface. At this point, the user is called into play to select which of these potential stitchings should actually take place: if the user selects at least one such stitching, the geometry is modified accordingly and the whole repairing is run once again.

To demonstrate this approach, a prototype interactive system has been implemented so that inaccurately-modeled objects (e.g. Fig. 8) could be repaired as expected. The prototype exploits a stitching procedure inspired on [Bischoff and Kobbelt 2005], and works as follows: the input STL undergoes the repairing process up to the
Algorithm 1 The outer hull extraction algorithm.\( A \leftarrow B \) with \( B \subseteq M \) indicates that all the triangles in \( B \) are moved from \( M \) to \( A \). Upon creation, orientable Face Clusters are arbitrarily oriented.

Require: A polyhedron \( |M| \)
Ensure: \( M_{solid} \) and \( M_{sheet} \)

1: Create a clustered polyhedron out of \( M \) and init \( M_{solid} = M_{sheet} = \emptyset \)
2: if \( M \) is empty then
go to (41)
end if
3: Determine and orient a seed triangle \( t_0 \) and its Face Cluster \( C_0 = C(t_0) \)
4: if \( C_0 \) is on boundary then
7: \( M_{sheet} \leftarrow C_0 \) and go to (2)
8: else
9: Orient \( C_0 \) consistently with \( t_0 \)
end if
10: Mark \( C_0 \) as visited
11: List \( L = C_0 \)
12: while \( L \) is not empty do
14: if \( C_0 \) is the first element of \( L \)
15: remove \( C_0 \) from \( L \)
for all Cluster Wall \( W \) of \( C_0 \) do
17: \( C_c \) = continuation of \( C_c \) at \( W \)
18: if \( C_c \) is on boundary then
19: \( M_{sheet} \leftarrow C_c \), unmark all the visits, and go to (2)
end if
21: if \( C_0 \) and \( C_c \) are inconsistently oriented then
22: if \( C_c \) is marked as visited then
23: \( M_{sheet} \leftarrow C_c \), unmark all the visits, and go to (2)
else
25: Invert the orientation for \( C_c \)
end if
27: end if
28: if \( C_c \) is not marked as visited then
29: Mark \( C_c \) as visited
30: \( L = L \cup C_c \)
end if
32: end for
33: end while
for all unvisited cluster \( C \) adjacent to a visited cluster \( C_0 \) do
35: Delete \( C \) from \( M \)
end for
for all visited cluster \( C \) do
38: \( M_{solid} \leftarrow C \)
end for
for all cluster \( C \) in \( M_{solid} \cup M_{sheet} \) do
41: for all cluster \( D \) in \( M_{solid} \) do
42: if \( C \) is contained in the space bounded by \( D \) then
43: Delete \( C \) (from either \( M_{solid} \) or \( M_{sheet} \))
end if
end for
end for

Figure 11: A model made of seven disconnected sheets with some unintended boundaries (a) is repaired and partitioned into \( M_{solid} \) and \( M_{sheet} \). The potential stitching areas are shown as voxel collections around the boundaries (b). After the user selection, the (triangulated) stitching areas are considered to be part of the input, and the repairing is re-run on such an integrated input: during this last repairing step, the algorithm keeps track of the triangles which were part of the stitching areas and keeps them selected (c). Vertices which are in the interior of a selected area are iteratively moved towards their centers of mass, so as to perform a local Laplacian smoothing (d). The resulting model is a single solid with two through holes representing the ‘Y’ character.

5.2 Applications

The technical contribution given in this paper has a number of potential applications, and this section briefly describes a few examples.

Important to consider that this example system contains the main algorithm but is not as-exact-as-possible. However, it shows how easily the main algorithm can be customized to implement extremely powerful and flexible repairing systems.
which cannot be due to numerical approximation. This is an important aspect because it means that, in practice, the connectivity of the STL models can be effectively reconstructed without the need of arbitrary tolerances.

Buoyancy and structural analysis - Besides the aforementioned target application, the determination of the outer solid is important when dealing with objects that must be immersed in fluids to assess, e.g., buoyancy characteristics [Wang and Whiting 2016]. In these cases, indeed, it is important both to understand which is the exact volume that cannot be flooded by the surrounding fluid, and to analyze the buoyant equilibrium to assess the stability of the floating object. Besides buoyancy, the repaired model is a well-defined solid that can be tetrahedrized, and thus can serve as a support for a number of simulations that can be run before printing the actual physical object.

Texturing and Shape Analysis - When a design model has sheets which are intended to be visible on both sides, texture mapping may become an issue. In this case, indeed, texture coordinates should be defined for both the sides separately, but unfortunately standard graphic pipelines do not support this feature. Our approach represents a perfect solution to this problem because all the visible portions of the object become solid. Clearly, in this case the inflation radius must be sufficiently small so that the distortion introduced is acceptable for the designer. Also, the described repairing tool is an extremely simple means to widen the applicability of many shape analysis algorithms that require their input to be a well-defined solid (e.g., computation of Reeb graphs [Biasotti et al. 2008], shape descriptors [Haibin and Jacobs 2007], generic 3D skeletons [Tagliasacchi et al. 2016]).

5.3 Limitations

Our algorithm is not meant to treat models with internal cavities. Surfaces that bound cavities, indeed, are not reachable from infinity, and therefore are not part of the outer hull according to our definition. Our definition can probably be extended to include these cases as well, but many additive manufacturing technologies cannot be used in any case to create such models because support material would remain trapped within the cavities. Furthermore, independently of the printing technology, we assume that the input STL is a raw representation of the object to be manufactured. Based on this, we can properly define the outer hull even if the orientation of triangles is unreliable. On the other hand, this generality makes us unable to distinguish an actual cavity from a topological artefact in the input. Hence, extending our definition to an outer and inner hull which includes cavities seems to be rather difficult: it is easy to select a point at “infinity” to define the entire outer hull, but it is not clear how one should define the cavities while assuming an unreliable orientation of the triangles. If the input can be guaranteed to have reliably oriented triangles, such a definition may take advantage from the notion of generalized winding numbers [Jacobson et al. 2013]. In all the other cases, one may think of involving the user in the disambiguation process: by providing a point which is known to be part of a cavity, the cavity itself can be defined as in Def. 3.16 while considering this point instead of $p$. Any of these solutions, however, would require to adapt Algorithm 1.

6 Conclusion and future work

We have formally defined the class of printable STL files and have shown how to convert a generic STL to a printable model with no or minimum distortion. In particular, we have shown that the solid parts of the input can be fixed with no visible deformations, whereas zero-thickness surfaces must be necessarily made solid to become printable, and hence visible in the eventual physical prototype.

An interesting objective for future research is the automatic distinction of intentionally designed thin features and unintentional open surfaces due to inaccurate modeling. A potential inspiration here may come from the notion of generalized winding numbers [Jacobson et al. 2013], though this method cannot be readily exploited because it heavily relies on the triangle orientation. Alternatively, thresholds and reasoning on distance fields can represent a good starting point.

Acknowledgements

This work is partly supported by the EU H2020-FoF-2014-2015/H2020-FoF-2015 Project N. 680448 (CAXMan) and by the international joint project on Mesh Repairing for 3D Printing Applications funded by Software Architects Inc (WA, USA). Thanks are due to the SMG members at IMATI for helpful discussions.

References

ASTM ISO / ASTM52915-13, 2013. Standard specification for additive manufacturing file format (amf) version 1.1.

ATTENE, M., AND FALCIDIENO, B. 2006. Remesh: An interactive environment to edit and repair triangle meshes. In Procs. of Shape Modeling International, IEEE.

ATTENE, M., CAMPEN, M., AND KOBBLT, L. 2013. Polygon mesh repairing: an application perspective. ACM Comp. Surv. 45, 2.

ATTENE, M. 2014. Direct repair of self-intersecting meshes. Graphical Models 76, 658–668.

AUTODESK. 2011. Meshmixer. http://www.meshmixer.com.

BERNSTEIN, G. 2013. Cork boolean library. https://github.com/gilbo/cork.

BIASOTTI, S., GIORGI, D., SPAGNUOLO, M., AND FALCIDIENO, B. 2008. Reeb graphs for shape analysis and applications. Theoretical Computer Science, Computational Algebraic Geometry and Applications 392, 1-3.

BISCHOFF, S., AND KOBBLT, L. 2005. Structure preserving CAD model repair. Computer Graphics Forum 24, 3, 527–536.

BISCHOFF, S., PAVIC, D., AND KOBBLT, L. 2005. Automatic restoration of polygon models. ACM Transactions on Graphics 24, 4, 1332–1352.

CAMPEN, M., AND KOBBLT, L. 2010. Exact and robust (self-)intersections for polygonal meshes. Computer Graphics Forum 29, 2, 397–406.

CHEN, Y., AND WANG, C. C. 2013. Regulating complex geometries using layered depth-normal images for rapid prototyping and manufacturing. Rapid Prototyping Journal 19, 4.

CIGNONI, P., CORSINI, M., AND RANZUGLIA, G. 2008. Meshlab: an open-source 3d mesh processing system. ERCIM News 73, 4, 45–46.

DE FLORIANI, L., DAVID, G., AND HUI, A. 2004. A data structure for non-manifold simplicial d-complexes. In Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, Eurographics Association, SGP ’04, 119 – 128.
FABMETHEUS. 2012. Skeinforge. http://fabmetheus.crsndoo.com/.

FABRI, A., and PION, S. 2009. Cgal: The computational geometry algorithms library. In Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems, ACM, 538–539.

FERRARIO, D. L., and PICCININI, R. A. 2011. Simplicial Structures in Topology, vol. CMS Books in Mathematics. Springer-Verlag New York.

FOMENKO, A. T., and MATVEEV, S. V. 2013. Algorithmic and computer methods for three-manifolds. Springer Science and Business Media.

GARLAND, M., and HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '97, 209–216.

GLASER, L. C. 1970. Geometrical Combinatorial Topology, vol. 1. Van Nostrand Reinhold, 450 West 33rd Street, New York.

GRIFFITHS, H. B. 1976. Surfaces. Cambridge University Press.

HACHENBERGER, P., KETTNER, L., and MEHLHORN, K. 2007. Boolean operations on 3d selective nef complexes: Data structure, algorithms, optimized implementation and experiments. Computational Geometry 38, 1–2, 6–49.

HAIBIN, L., and JACOBS, D. W. 2007. Shape classification using the inner-distance. Pattern Analysis and Machine Intelligence, IEEE Transactions on 29, 2.

HUANG, P., WANG, C. C. L., and CHEN, Y. 2013. Intersection-free and topologically faithful slicing of implicit solid. ASME Transactions - Journal of Computing and Information Science in Engineering 13, 2.

HUANG, P., WANG, C. C. L., and CHEN, Y. 2014. Algorithms for layered manufacturing in image space. ASME Advances in Computers and Information in Engineering Research 1, 377–410.

HUI, A., VACLAVIK, L., and FLORIANI, L. D. 2006. A decomposition-based representation for 3D simplicial complexes. In Fourth Eurographics Symposium on Geometry Processing, Eurographics Association.

JACOBSON, A., KAVAN, L., and SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. ACM Transactions on Graphics 32, 4, 33:1–33:12.

JU, T. 2004. Robust repair of polygonal models. ACM Transactions on Graphics (Proc. SIGGRAPH) 23, 3, 888–895.

JU, T. 2009. Fixing geometric errors on polygonal models: A survey. Computer Science and Technology 24, 1, 19–29.

LUO, L., BARAN, I., RUSINKIEWICZ, S., and MATUSIK, W. 2012. Chopper: Partitioning models into 3d-printable parts. ACM Transaction on Graphics 31, 6.

MICROSOFT, and NETFABB. 2013. 3D model repair service. https://netfabb.azurewebsites.net/.

MOORE, D. M. 2013. Instant Slic3r. Packt Publishing Ltd.

PANOZZO, D., and JACOBSON, A. 2014. LIBIGL: A C++ library for geometry processing without a mesh data structure. In SGP 2014 Graduate School Course Notes.

PINTUS, R., GOBBIETTI, E., CIGNONI, P., and SCOPIGNO, R. 2010. Shape enhancement for rapid prototyping. The Visual Computer 26, 831–840.

QU, X., and STUCKER, B. 2003. A 3d surface offset method for stl-format models. Rapid Prototyping Journal 9, 133–141.

ROSSIGNAC, J., and CARDOZE, D. 1999. Matchmaker: manifold breps for non-manifold f-sets. In Procs of 5th ACM symposium on Solid modeling and applications, ACM Press, New York, NY, USA, 31–41.

SHEWCHUK, J. R. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. From Discrete and Computational Geometry 18, 3, 305–363.

SHILANE, P., MIN, P., KAZHDAN, M., and FUNKHAUSER, T. 2004. The princeton shape benchmark. In Shape Modeling International, Genova, Italy.

SI, H., and GAERTNER, K. 2005. Meshing piecewise linear complexes by constrained delaunay tetrahedralizations. In Procs. 14th Intl. Meshing Roundtable, 147–163.

STAVA, O., VANEK, J., BANES, B., CARR, N., and MECCH, R. 2012. Stress relief: improving structural strength of 3d printable objects. ACM Transaction on Graphics 31, 4.

STILLWELL, J. 1993. Classical Topology and Combinatorial Group Theory (2nd ed). Springer-Verlag New York.

TAGLIASACCHI, A., DELAME, T., SPAGNUOLO, M., AMENTA, N., and TELEA, A. 2016. 3d skeletons: A state-of-the-art report. Computer Graphics Forum 35, 2.

TELEA, A., and JALBA, A. 2011. Voxel-based assessment of printability of 3d shapes. In Proceedings of the 10th International Conference on Mathematical Morphology and Its Applications to Image and Signal Processing, Springer-Verlag, ISMM’11, 393–404.

VANEK, J., GALICIA, J. A. G., BENES, B., MECCH, R., CARR, N., STAVA, O., and MILLER, G. S. 2014. Packmerger: A 3d print volume optimizer. Computer Graphics Forum 33, 6, 322–332.

WANG, C., and CHEN, Y. 2013. Thickening freeform surfaces for solid fabrication. Rapid Prototyping Journal 19, 6, 395–406.

WANG, C., and MANOCHA, D. 2013. Efficient boundary extraction of bsp solids based on clipping operations. IEEE TVCG 19, 1, 16–29.

WANG, L., and WHITING, E. 2016. Buoyancy optimization for computational fabrication. Computer Graphics Forum 35, 2.

YAU, H.-T., KUO, C.-C., and YEH, C.-H. 2003. Extension of surface reconstruction algorithm to the global stitching and repairing of STL models. Computer-Aided Design 35, 5, 477–486.

Appendix - On the correctness of Algorithm 1

The concept of $M_{solid}$ formalized in Def. 3.17 captures the intuitive notion of solid without dangling sheets. The Euclidean space is split in two parts: one which is path-connected with $p$ which we call $M_{ext}$, and one which is not which we call $M_{int}$, and $M_{solid}$ is the piecewise linear interface between these two parts. Note that $M_{int}$ can be made of multiple connected components, just as $M_{solid}$. In Def. 3.17, any dangling sheet in $|M|$ is surrounded on both sides by points which are path-connected with $p$, so that the boundary operator applied to the set $O$ has the effect of cancelling it from
Thus, each face of $M_{\text{solid}}$ can be assigned a unique orientation that depends on which of its two sides is occupied by $M_{\text{ext}}$. For any non-singular edge, the two faces that share that edge must necessarily have a consistent orientation, but what happens around singular edges? If $e$ is a singular edge, all the triangles that share $e$ can be radially ordered around it. We call the set of these triangles $T(e)$. Let us construct a small enough circumference $C$ orthogonal to $e$ and centered at its midpoint. Herewith *small enough* means that all and only the triangles in $T(e)$ intersect $C$. Let $t_0$ be one of the triangles in $T(e)$, and let $c_0$ be the intersection of $t_0$ and $C$. By definition, $c_0$'s neighboring points are in $M_{\text{int}}$ on one side of $t_0$ and on $M_{\text{ext}}$ on the other, and since the plane of $C$ is orthogonal to that of $t_0$, we can induce an unambiguous orientation on $C$. In other words, we can start moving on $C$, starting from $c_0$, in the direction of the points that belong to $M_{\text{ext}}$ as determined by $t_0$. Along the movement we will encounter another point of intersection generated by another triangle in $T(e)$, let it be $t_1$. Since we are coming from $M_{\text{ext}}$, $t_1$'s orientation is unambiguously defined and must be consistent with $t_0$'s orientation. In practice, for the sake of their orientation, the triangles $t_0$ and $t_1$ behave as if they were sharing a non-singular edge.

Note that the concept of orientability for manifolds with singularities is not new [Fomenko and Matveev 2013], and it is key to fully understand why Algorithm 1 works. Basically, Algorithm 1 visits all the triangles that belong to the outer hull and tries to propagate a consistent orientation even across singular edges, so as to enclose a well-defined subset of the Euclidean space (i.e. $M_{\text{int}}$ or a portion of it). The propagation fails when it reaches either a boundary triangle or a triangle which was previously oriented in the opposite direction: in these cases the triangle cannot be part of $M_{\text{solid}}$, and is therefore moved to $M_{\text{sheet}}$. The use of Face Clusters instead of single triangles allows to pre-aggregate triangles, and provides a significant advantage in terms of efficiency: the order of the visits does not influence the final result, but Algorithm 1 restarts from line 2 whenever an element is moved to $M_{\text{sheet}}$ thus, if the number of these elements is high (e.g. because we consider every single triangle independently) the process might become too slow. On an edge-connected input, Algorithm 1 locates and moves to $M_{\text{sheet}}$, one at a time, all the Face Clusters that cannot be consistently oriented with the seed triangle. When the last of these Face Clusters is moved to $M_{\text{sheet}}$, Algorithm 1 iteratively visits all the remaining clusters which are then all consistently oriented with the seed, and the "while" cycle terminates. Due to their consistent orientation and to the choice of the seed triangle, these visited Face Clusters split the space in $M_{\text{ext}}$ and $M_{\text{int}}$, and all the clusters which were moved to $M_{\text{sheet}}$ must necessarily belong to $M_{\text{ext}}$. Since these $M_{\text{sheet}}$ clusters do not split the space, $M_{\text{ext}}$ is path connected with a point at infinity, and hence the visited clusters actually form $M_{\text{solid}}$.

If the input is not edge-connected, lines 1-39 split an edge-connected component into a tentative pair $(M_{\text{solid}},M_{\text{sheet}})$, but each cluster therein can be contained in the space bounded by larger components, and thus may not be part of the actual $M_{\text{solid}}$ or $M_{\text{sheet}}$. Lines 41-47 perform such a final filtering and ensure that only the most "external" clusters are kept in the output.