RC 20935 (92693) (7/25/97)
Computer Science/Mathematics 32 pages

Research Report

Cutting and Stitching: Efficient Conversion of a Non-Manifold Polygonal Surface to a Manifold

André Guéziec, Gabriel Taubin, Francis Lazarus and William Horn

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich · Austin

Cutting and Stitching: Efficient Conversion of a Non-Manifold Polygonal Surface to a Manifold

André Guéziec, Gabriel Taubin, Francis Lazarus and William Horn
IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
{gueziec,taubin,francis,hornwp}@watson.ibm.com

ABSTRACT:

We present an algorithm for converting a non-manifold polygonal surface to an immersed manifold polygonal surface, orientable if necessary, so that it can be processed by a variety of algorithms that require that the neighborhood of each vertex be topologically equivalent to a disk.

The algorithm manipulates the polygon vertex indices, or surface *topology*, and essentially ignores the vertex coordinates, or surface *geometry*. We identify *singular* edges and vertices and *multiply* singular vertices, or *cut* through singular edges. In the optional *stitching* stage, we join surface boundaries that were cut, or that satisfy a given relation, while guaranteeing that the resulting surface is a manifold. We distinguish between stitching along the same boundary and along different boundaries, and show the difficulties associated with the latter. Cutting and stitching are general tools that are useful independently of converting to a manifold.

The conversion has a linear complexity in the number of vertices, edges, faces, and boundary vertices, as we essentially perform one pass on such elements. We store the conversion information by means of a look up table for the vertices that are multiplied, allowing to restore the original topology.

Key-words : Manifold, Cutting, Stitching.

1. Introduction

Polygonal surfaces are a common choice for representing three dimensional geometric models. Such models are used for generating pictures and animations, and are also used in CAD systems, in Scientific Visualization and Medical Imaging. A computer representation of such surfaces can be created by a human designer using a software geometric modeling tool, or by an algorithm that builds a surface from sensor data, or extracts a surface from volumetric data. For instance, range scanners, measuring arms and other devices for capturing three dimensional geometric data can be interfaced with software that produces polygonal surfaces from their output data. Another example is given by algorithms for extracting iso-intensity surfaces from volume data, that are routinely used in various scientific disciplines. For economic reasons, and because of increasing demand for realism and detail in the surface representation, automatic acquisition is preferred to manual design and its importance is growing.

There are a number of algorithms that operate on polygonal surfaces. A partial list of such algorithms is: (1) algorithms that produce two dimensional graphics for display on computer screens, for printing, or for other purposes; (2) algorithms that compute surface characteristics, e.g. curvature (see Hamann [1]), area, volume of a solid bounded by the surface, various iso-curves on the surface (see Lazarus and Verroust [2]); (3) algorithms used for filtering surfaces, such as smoothing algorithms (see Taubin [3]), or algorithms for computing the surface Laplacian, such as Huiskamp's [4]; (4) algorithms that perform finite element analysis (FEA) on surfaces for various engineering applications such as automobile crash simulation; algorithms that optimize surface meshes for FEA applications (see [5]); (5) algorithms that simplify surfaces, either for allowing real-time interactivity in visualizations, or for efficiency purposes; simplifying means approximating the surface with another surface that contains fewer polygons; a survey of recent simplification algorithms is provided in [6]; (6) algorithms for surface subdivision, such as Doo and Sabin's method [7] or Zorin *et al.*'s method [8]; surface subdivision is a technique for introducing new polygons in the surface, to produce a result that resembles a smooth surface; (7) algorithms for surface compression; surface compression consists of an efficient encoding of polygons defining the surface; surface compression algorithms were proposed by Deering [9] and Taubin and Rossignac [10]; (8) algorithms that (scan) convert a polygonal boundary representation of a potential solid for Rapid Prototyping [11].

Many available algorithms of the previous categories, with the exception of the first category (1), operate exclusively on a *manifold* surface. A manifold surface is such that the neighborhood of every point is *homeomorphic* to a disk except at the surface boundary where the neighborhood of every point is homeomorphic to a half disk. Homeomorphic means that there exists a bijective and bi-continuous mapping between the two sets considered. For the purpose of the present paper, we ignore the coordinates associated with the surface elements, and we look at the property of being a manifold as a purely topological one. (Mathematically, we consider topological immersions of two-manifolds as defined by Massey [12]; we give precise definitions in Section 1.1.) It is justified that algorithms operate in the presence of self-intersections, because of the cost of detecting and eliminating them, or because self-intersections are an integral part of the model (e.g., of a Klein bottle in Euclidean three dimensional space). For convenience, throughout this article, we will call such immersions of two-manifolds *manifold surfaces*. We will write that a surface is a manifold.

The original motivation of this work was to extend an existing surface simplification algorithm [13] as well as a surface compression algorithm [10] that were designed to handle manifold surfaces. In both cases, manifold surface representations were chosen because of algorithmic advantages (simplicity, and speed) and because much potential input data was anticipated to be manifold, for instance a closed iso-intensity surface whose volume and genus had to be defined (and preserved by the algorithm), or the surface model of a spaceship. We observed that such models often were not manifolds, that there was not necessarily a rational explanation for it, and that there was a need to design an efficient algorithm to convert such surfaces to manifolds.

The main contribution of this paper is to provide a complete description of an efficient method for automatically converting a non-manifold surface to an (immersed) manifold surface. Although our ideas are conceptually simple, our experience showed that implementing such algorithms without omitting any special case can be complicated and error prone. The complexity of our algorithm is linear in the number of vertices, edges and faces. Essentially, no floating point operation is necessary. The conversion of a surface to a manifold, when used as a preprocessing step to most algorithms operating on surfaces, will have a negligible cost.

The algorithm is decomposed in two main parts: *cutting* and *stitching*. Cutting is a general method for disconnecting the surface topology along a set of marked edges or singular vertices; cutting is formally defined in Section 3. Here, we perform the conversion by detecting the *singular* vertices and edges of the surface and cutting through such singular elements. An edge is singular if more that two faces are incident to it. A singular vertex is defined in Section 1.1. We describe two different methods for cutting: a global method and a local method. The global method operates on all the faces and vertices of the surface at once, by first breaking all connections between faces and later joining adjacent faces that share a non-marked edge. The local method operates only on singular vertices and vertices incident to marked edges, by counting the number of edge-connected sets of faces incident to a vertex, by *multiplying* the vertex, and by assigning a different copy of that vertex to each connected set. We give the full detail of such methods in Section 3. When cutting, we keep a look-up table of the correspondence between new (multiplied) vertices and old vertices, thus permitting to undo the cutting: the operation is fully reversible.

As illustrated in Fig. 1, several manifolds can be mapped to the original non-manifold by identifying vertices. After cutting through all the singular edges and vertices using the methods of Section 3, we have produced the manifold with the maximum number of vertices and the maximum number of components that cuts through the singular edges and vertices and nowhere else (according to the definition of cutting of Section 3).

To reduce the number of vertices, holes, or components, the cutting operation is followed by stitching. As defined in Section 4, stitching consists of taking two surface boundaries (possibly the same boundary) and joining them by identifying boundary edges, while guaranteeing that the surface is a manifold. We have observed that stitching is a delicate operation, if we wish to obtain a manifold after stitching: for instance, when using the "zipping" method reported in [14], singular edges can be created when zipped multiple times. We distinguish between stitching along the same boundary and stitching along different boundaries. We finish by showing examples and providing sample timings.

Fig. 1 is a diagram illustrating our method. We study the example of two tetrahedra sharing an edge: we subdivide the surface of the tetrahedra into smaller triangles, resulting in the surface of

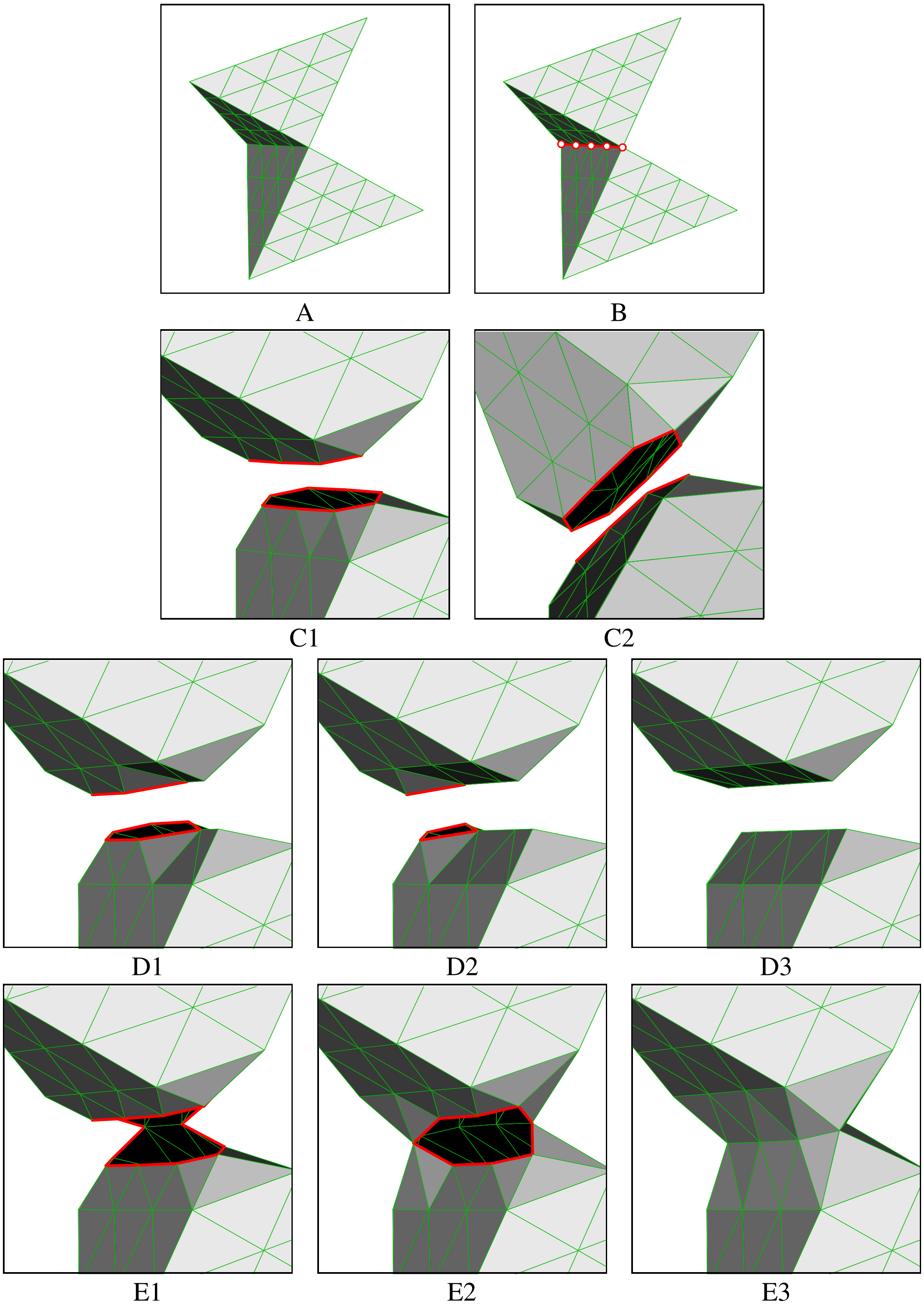


Figure 1: Sketch of the conversion to a manifold surface. A,B,C: cutting through singular edges; For illustrative purposes, topologically disconnected vertices are shown apart; however, in general vertex coordinates are not modified by our algorithm. There are two strategies for stitching: either along the same boundary (D) or along different boundaries (E).

Fig. 1A. We label singular edges and vertices and color them in red in Fig. 1B. After multiplying singular vertices, we have created two disconnected surface components in Fig. 1C, each of which has a boundary of length eight; as explained in Section 3, the three singular vertices in Fig. 1B that are shared by two singular edges are multiplied four times each. The two singular vertices shared by one singular edge only are multiplied twice. After stitching along the same boundaries, we have created in Fig. 1D two disconnected closed surfaces. If instead we had stitched along different boundaries, we could have created in Fig. 1E a single closed surface component. As defined in Section 1.1, all three surfaces C, D, and E are manifolds with the same geometrical realization as A.

1.1. Definitions

For our purposes, a *surface*[1] $S(\{\mathbf{v_i}\}, \{f_j\})$ is defined with a set of vertices $\{\mathbf{v_i}\}$ and a set of faces $\{f_j\}$. Each vertex has coordinates in $\mathbf{R^3}$. Each face is specified with a tuple of at least three vertex indices (we use boldface for vertices which are vectors of coordinates, and normal typesetting for vertex indices which are pointers or integers). The face is said to be *incident* on such vertices. Each vertex must have at least one incident face. The vertex indices in a face must all be different. Otherwise the face is considered *invalid*. An *edge* of a face is defined as a pair (v_i, v_j) of consecutive vertices of that face, modulo circular permutation. The face is said to be incident on the edge, and the edge incident on the vertices v_i and v_j. v_i and v_j are also said to be *adjacent* vertices. Edges sharing a vertex and faces sharing an edge are said to be adjacent edges and faces. There are two possible orderings of the vertices of a face modulo circular permutation, resulting in two orientations for that face.

We next define the *topology* of a surface, as the set of ordered subsets of indices provided by the set of faces $\{f_j\}$, modulo circular permutation. The set of faces uniquely defines a topological space by the following construction: associate to each face with k indices a regular k-gon in the plane, labeling the vertices with the face indices. Then glue together such polygons along shared edges. We call an *abstract surface* the set $\{f_j\}$ of faces.We refer to the *geometry* of a surface as the image of its topology under a map that is an *embedding* restricted to each face. An embedding, or imbedding, of a surface maps its topological space into R^3 homeomorphically to its image. Note that except for triangles, the vertex coordinates do not define any particular embedding of a face unless its edge cycle is coplanar and non self intersecting. We will still use the word geometry to mean the set of vertex coordinates. Cutting and stitching operate on the topology only.

We call the subset of faces of $\{f_j\}$ that share a vertex $\mathbf{v}$ the *star* of $\mathbf{v}$: $\mathbf{v}^\star$. The number of faces in $\mathbf{v}^\star$ is called the *valence* of the vertex $\mathbf{v}$, $v(\mathbf{v})$. To form the *link* of a vertex, $\ell(\mathbf{v})$ we first take all edges belonging to faces in $\mathbf{v}^\star$, and then remove the edges incident on $\mathbf{v}$. The link is a graph made by linking up the remaining edges. See Figure 2. A *regular vertex* has a link formed of one polygonal curve; if the link is closed, then it must be of length at least three; otherwise the vertex is a *singular vertex*. If a regular vertex has a closed link, then it is *interior regular*, otherwise it is *boundary regular*. These cases are illustrated in Fig. 2.

An abstract surface is a *manifold* if each vertex is a regular vertex; otherwise it is a *non-manifold*

[1]The usual algebraic topology terminology reserves the term surface to a manifold simplicial complex [15]. For lack of a better word we extend here the term surface to any collection of faces.

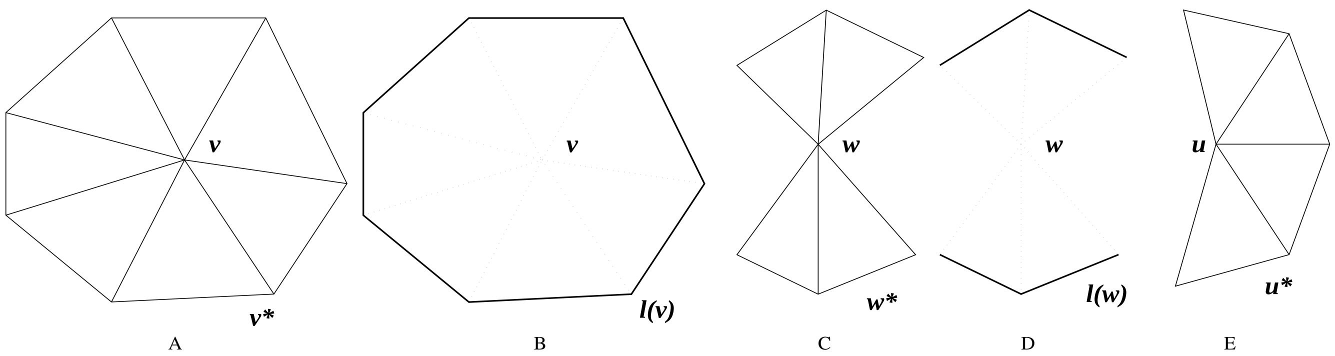


Figure 2: A: the star $\mathbf{v}^\star$ of a regular vertex $\mathbf{v}$ of valence seven. B: the link $\ell(\mathbf{v})$ of the regular vertex $\mathbf{v}$, composed of one simple closed polygonal curve. C: the star $\mathbf{w}^\star$ of a singular vertex $\mathbf{w}$ of valence four. D: the link $\ell(\mathbf{w})$ of $\mathbf{w}$, composed of two disconnected polygonal curves. E: the star $\mathbf{u}^\star$ of a boundary vertex of valence four.

surface. A *topological immersion* is such that the neighborhood of every point is mapped homeomorphically to its image, meaning that self-intersections are tolerated. The output of our algorithm is a topological immersion of an abstract manifold surface [2]. Note that some particular coordinate specifications for polygon vertices will not provide an immersion (for instance when two adjacent triangles intersect in a non-zero area).

An (abstract) manifold surface is *closed* if each edge is shared by exactly two faces. We call an edge incident on one single face a *boundary edge*. The two endpoints of a boundary edge are then necessarily boundary vertices. The surface boundary is obtained by joining boundary vertices with boundary edges, forming a series of closed loops. Other edges are called *adjacent to the boundary* if at least one endpoint is a boundary vertex and otherwise *interior edges* (In a non-manifold surface, we call an edge with a single incident face a boundary edge as well, but the graph obtained by linking such edges may be more complex than a series of closed loops.) Two adjacent faces sharing an edge e have a *compatible orientation* if the two vertices of e listed in one face appear in opposite order in the other face. The surface is *orientable* if each face can be oriented such that any two adjacent triangles have a compatible orientation. An orientable manifold surface such that its faces are all oriented in a compatible way is said to be *oriented*. An orientable manifold surface can be oriented in only two possible ways.

1.2. Background

Researchers and practitioners from various fields have acknowledged that in practice much of the available surface data consists of non-manifold surfaces, with the consequence that many methods cannot be applied [18, 19, 20]. We believe that the primary reason is the ambiguous definition for a polygonal surface in the Engineering and Graphics communities. The present paper is the first attempt that we are aware of for converting non-manifold surfaces to manifold surfaces in full generality.

[2] Abstract manifold surfaces with geometrically coincident vertices or edges have been termed *pseudo manifolds* by certain authors [16, 17]. However, pseudo-manifolds generally have a different definition in mathematics [15].

There is an intuitive notion of what a polygonal surface is, and mathematical definitions. Somewhere in between, the approach of most computer scientists has been to focus on a particular set of applications and to develop a suitable surface data structure. This has resulted in much duplicate effort. For instance, there is the Baumgart Winged-Edge data structure [21], the edge-based structures proposed by Weiler [22] or Karasick [23].

Some surface representations, such as the Winged Edge proposed in the mid 1970s, can only represent a manifold surface; this constraint is also imposed on the associated file format. Kalvin [24] uses the Winged Edge to represent manifold iso-surfaces. But the Winged Edge requires significantly more memory than a representation using just the vertex coordinates and the polygon indices. For concise representations, a consensus for data structures and associated file formats recently employed appears to consist of a set of vertices and a set of polygons. Such formats are less structured, as the VRML format [25]; when using them, it is possible to create a non-manifold unintentionally.

One long term solution to our problem would be to widely adopt non-manifold data structures. Many algorithms could be modified to handle non manifold surfaces in various ways. Note that some particular methods are insensitive to singularities. However, the question of deciding on the relative advantages of manifold versus non-manifold surface representations is outside the scope of this article. Historically, data structures suitable for non-manifold surfaces were advocated in Solid Modeling (see Appendix A.)

1.3. Related Work

For completeness, we discuss the following related work. Again, our method is very different as it operates solely on the surface topology. The first category of methods operate both on the geometry and topology to modify surfaces so that they can represent the boundary of solids (solids are reviewed in Appendix A). This is an important issue for Rapid Prototyping of models represented using the .STL format consisting of topologically disconnected triangles (see [20, 11, 26]). As was duly noted, converting a non-manifold surface to a solid is a difficult task with floating point precision problems, computationally demanding tasks (e.g., polygon intersections), and a number of open problems, as the problem of filling a polygonal hole (boundary curve) with a "reasonable" polygonal surface without creating intersections [27]. Relatedly, Butlin *et al.* [28] attempt to "Repair" CAD data in order to use it for engineering analyses or to simplify data exchange. Barequet and Kumar [29] operate on .STL files; as with the global cutting method of Section 3.3, they first stitch through regular edges, but they can subsequently create a non-manifold after stitching additional edges.

There has been a significant research effort on providing new tools to create and manipulate surface models. The technique of Szeliski *et al.* [30] builds a new polygonal surface from an existing surface by defining a collection of point samples, using point repulsion methods to distribute the points evenly. Subsequently, a manifold surface triangulation of the remaining points is found by extending the techniques of Delaunay triangulation to surfaces. The technique of Welch *et al.* [31] builds a polygonal surface starting from a simple surface, by applying a series of surface operations, that consist of adding, deleting or morphing a portion of surface. They use methods similar to Szeliski's for triangulating surface points. They use mesh cutting techniques, but cut only along simple curves. Both methods build a new lists of vertices and faces, while we manipulate an existing list of face vertex indices. Veron and Leon [32] detect automatically singular vertices and edges but they require user assistance for correcting the singularities.

The last category of related work is in Solid Modeling, to develop data structures for boundary representations of solids. It is related because conversions between manifold and non manifold representations are discussed, for instance in Hoffman [33]. Heisserman [34] developed a method for extracting a manifold boundary representation from a set of intersecting solids. Our approach is different because we do not assume to work with solids, and do not use the notions of interior or complement.

2. Data Structures

Our methods take as an input a surface represented with a list of n_v vertices and with a list of n_f faces. Internally, the vertices and faces are preferably represented using a "vertex array" and a "face array". The vertex array contains the vertex coordinates (three per vertex). The face array contains the vertex indices for each face stored contiguously. We also use a "face start array" to provide the starting index of each face in the face array. We have also implemented object oriented representations that ignore distinctions between lists and arrays, and which can simplify accesses to, and additions and deletions of, vertices or faces. For clarity, we refer to vertex and face arrays, independently of low level data structures.

Additionally, vertices or faces may have a number of continuous or discrete attributes, such as colors, normals, and texture coordinates. Attributes can also be associated with each vertex of each face; (vertex, face) pairs are called *corners*. All attributes are passed on to the output surface. Face and corner attributes are unchanged, as our method preserves corners. When a vertex is multiplied, the same attributes are assigned to all copies.

By looping through the face array, we build a structure of n_e surface edges, recording the number of incident faces. For efficiency, the algorithm requires constant time access in average to the edges indexed by the two endpoint indices. The edge data structure should also provide constant time access to the incident faces. In our implementation, the edges are organized as a hash table indexed by the sorted pair of endpoint indices (smaller vertex index followed by larger vertex index). This hash table, and the list of face incidences for each edge are constructed in $O(n_f)$ time by visiting each face, and for each pair of consecutive vertices modulo circular permutation, by retrieving the edge in the hash table and updating its incidence list or inserting the edge in the hash table if it was not present. Boundary edges have a single incident face. A special marker is used to indicate invalid faces. We have the choice between recording all the incident faces on an edge, or at most two incident faces: this is sufficient if our sole purpose is to convert a surface to a manifold; in this case, we can allocate a fixed space of two incident faces per edge, and stop storing incident faces as soon as two already were encountered.

When cutting and stitching, we need to maintain a partition on the faces of a vertex star or equivalently on all the corners associated to a vertex. Preferably, we use the Union-Find algorithm for this purpose, whose running time is essentially $O(n)$, when n elements are in the partition [35]; once the partition is determined, access to representatives of faces or corners takes constant time. Partitions are also used for orienting faces consistently when stitching in Section 4.3.

3. Cutting

The processes of cutting and stitching [3] are familiar in Topology: in textbooks, they are used for classifying manifold surfaces [12, 36], most often using intuitive definitions. To discuss their properties later, we need to give precise definitions. Cutting and stitching have a rigorous mathematical framework, by associating labels to the face corners of an abstract surface; we refer to Agoston's definitions when faces are simplices ([15], pp. 46-54). For our purposes, given an abstract surface $\{f_j\}, 1 \leq j \leq p$ and the n vertex indices it uses $\{v_i\}$, a *surface cut* is a new abstract surface (set of faces) $\{g_j\}, 1 \leq j \leq p$ over m vertex indices $\{w_i\}$ with $m > n$, and a surjective mapping $c : \{w_i\} \mapsto \{v_i\}$, such that the image by c of the vertex indices in a face g_j are the indices in f_j, $c_{|g_j}$ is one to one (a polygon with k vertices is mapped to a polygon with k vertices: face corners are preserved), and adjacent faces are mapped to adjacent faces [4]. The number $m - n$ of new vertices after a cut is called the *size* of the cut. A cut of size zero is the identity; otherwise it is a *proper cut*. The number of boundary edges created by a cut is called the *length* of the cut.

c induces a mapping c_e on the surface edges. We consider a subset of surface edges, called the set of *marked* edges. A *cut through marked edges* is such that c_e restricted to the inverse image of the marked edges is not bijective and c is bijective on the rest of the surface edges. A *cut through isolated marked vertices* is such that c restricted to the inverse image of the marked vertices is not bijective and c is bijective on the rest of the surface vertices. In the sequel, we study two methods for cutting through marked edges and vertices.

3.1. Local Method for Cutting Through Marked Edges

In this method, we first build a list of marked vertices. We call it a *local* method because it only operates on a selected number of vertices and faces. A vertex is marked if it is an endpoint of a marked edge. We visit each marked vertex in turn; for each marked vertex v, we must specify its star $v^\star$, which can be obtained for each vertex by looping on the face array. We decompose $v^\star$ in subsets that are connected by non-marked edges. This can be performed using standard methods: we collect all non-marked edges incident on v, and maintain a partition on the faces of $v^\star$, considering two faces adjacent if and only if they share a non-marked edge. Here we assume that all faces are valid (have no repeated vertex indices). Invalid faces, violating our definition of a surface, are treated in Section 3.4.1.

Once the number of connected components n_c is known, we create $n_c - 1$ additional copies of the vertex v with the same coordinates and same attributes. Each instance of v is labeled from 0 to $n_c - 1$. In $v^\star$, we revisit each face in turn and for each face f, we locate the index of v, and we replace it with the instance of v corresponding to the component number of f. We call this operation *multiplying* the vertex v. Every time a vertex is multiplied, we create $n_c - 1$ new entries in a look up table of vertices; in this look-up table we record that the ancestor of all copies of v is precisely v.

We illustrate the local cutting method on the star of a vertex v_5 with six incident faces $f_0 \ldots f_5$ in

[3] Stitching is sometimes called "pasting" or "sewing"; we believe that the analogy to "stitches" is useful when face corners are identified one pair at a time; we use circular arcs to visualize stitches in Figs. 5 and 7.

[4] if all the face were simplices (triangles), a cut would be a particular type of *simplicial map*.

Fig. 3. In Fig. 3A, marked edges are drawn bold. The non-marked edges incident on v_5 are (v_4, v_5) and (v_5, v_7). Four connected components of faces $g_0 \dots g_3$ are identified in Fig. 3B. Accordingly, four copies of the original v_5 are used: $v_5, v_{10}, v_{11}, v_{12}$. Again, no vertex coordinate is actually modified. We draw topologically disconnected faces as geometrically disconnected for illustrative purposes.

Once all marked vertices have been multiplied, the operation is completed. We now wish to compute the worst case complexity of the operation. The cost to compute the number of connected components of faces incident on every marked vertex is less than the number of marked vertices times the largest valence of a marked vertex. Then, for each vertex $\mathbf{v}$, we need to know the relative position of the corresponding corner in incident faces, in order to replace the corresponding vertex index with an index to the proper copy of $\mathbf{v}$. Instead of searching for the corner, which would incur an additional computational cost, we propose to pre-store for each vertex a list of corner positions in each incident face. Looping through each face allows to build such a list. If we assume that this list is built for each vertex, including non-marked vertices then the cost is proportional to the total number of corners in the surface. Otherwise, if we build the list for each marked vertex only, the cost is proportional to the total number of corners in the stars of the marked vertices. The worst case complexity of the cutting is proportional to the number of marked edges times the largest number of corners in a vertex star.

Determining the Length of the Cut. We now try to characterize the result of the present algorithm with a formula for the length of the cut. Marked boundary edges do not produce a proper cut. A *corolla* of an edge e is a set of (valid) faces incident on either edge endpoint or both endpoints of e, such that each face is adjacent to two faces of the set (corolla) through a non-marked edge adjacent to e (different from e). (see Fig. 4A, B). A marked edge with at least one corolla is called an *isolated interior marked edge*.

In the surface resulting from the cut through marked edges, for each marked edge e and each incident face f not belonging to any corolla, a boundary edge b is created: we consider in turn each face f_i adjacent to e and different from f. As the pair of faces (f, f_i) does not belong to any corolla, at least one of e's endpoints v is such that f and f_i are not connected through non-marked edges incident on v (Fig. 4C): after cutting, v is multiplied, and f and f_i are incident on different copies of v. We cannot increase the length of the cut (introducing new boundaries) without marking other edges: two faces f and g incident on a marked edge e and belonging to a corolla of e cannot be disconnected by cutting, as a boundary of length two would be created, with two edges sharing the same endpoints; this is not allowed. Among all cuts through marked edges, the method of the present section produces a cut of maximum length.

In Fig. 1, C, D and E are all obtained from A with cuts through marked edges. C is obtained using the present method. C could also be obtained from D and from E by cutting through marked edges. The cuts from D to C and E to C have the same length of eight. They have different sizes, of respectively six and eight.

The length of a cut through marked edges can be computed by summing the number of face incidences of the marked edges, which is the maximum possible length, and by subtracting the number of faces belonging to a corolla. This formula is useful to characterize the present algorithm, but it is not practical; the best way of counting the number of boundary edges that are created when cutting is to execute the algorithm.

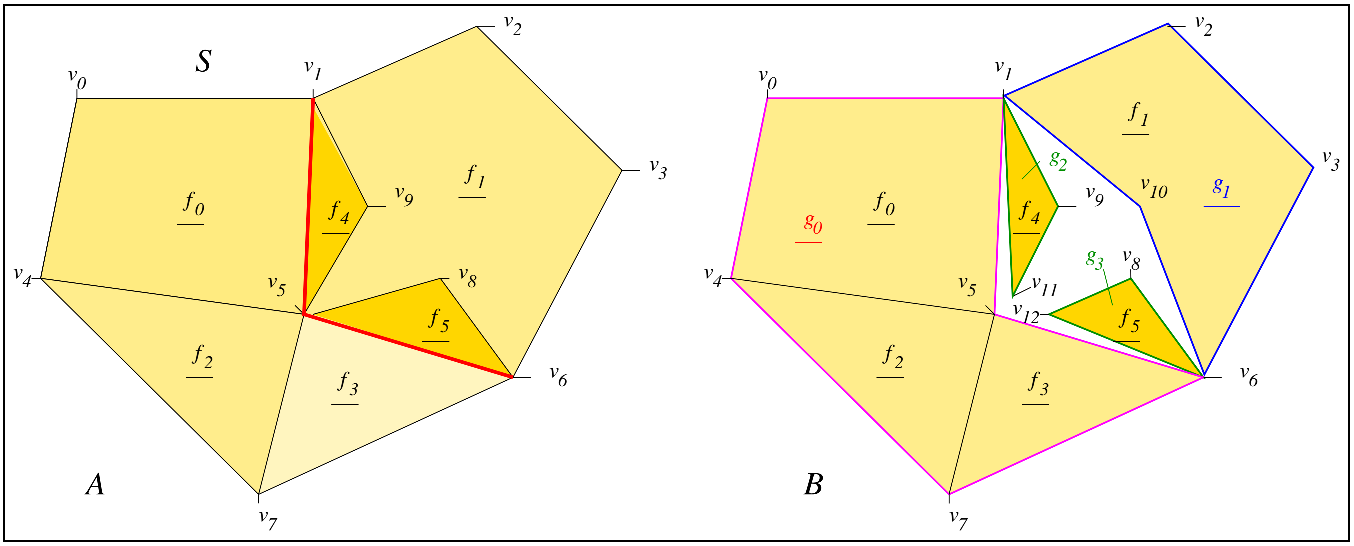


Figure 3: Local cutting. A: star of Vertex v_5 with marked edges in bold. B: multiplying Vertex v_5.

Another property of the cut through marked edges is that any two surface portions that share a collection of marked edges and no other edge or vertex will be disconnected by the cut.

3.2. Local Method for Cutting Through Isolated Singular Vertices

This is a variation of the previous method in which we mark *isolated* singular vertices. Recall that a singular edge is an edge shared by at least three faces. Clearly, an endpoint of such an edge is a singular vertex. A singular vertex that is not an endpoint of a singular edge is called an isolated singular vertex (see Fig. 4D).

To determine whether a vertex v is singular, if it is not an endpoint of a singular edge, we attempt to build its star $v^\star$ by pivoting about v, starting with any face f incident on v. We can specify one incident face for each vertex by looping through the face list. To pivot about v, we locate in f an edge e incident on v. From the edge data structure, we infer a face g that together with f shares e; we then locate in g an edge incident on v different from e. We continue until the edge has a single incident face or until the original face was encountered. We then count the number of faces that were visited and we determine whether this number equals the valence of the vertex v. Note that this method also works if the faces are not consistently oriented: even if all vertices are regular, it is possible that the surface is a non-orientable manifold. It is less expensive to do this test than to arbitrarily mark the vertex and discover during the cutting operation that it won't be multiplied.

The rest of the method is the same as before. Both types of cut can be combined by marking both isolated singular vertices and marked edge endpoints.

3.3. Global Method for Cutting

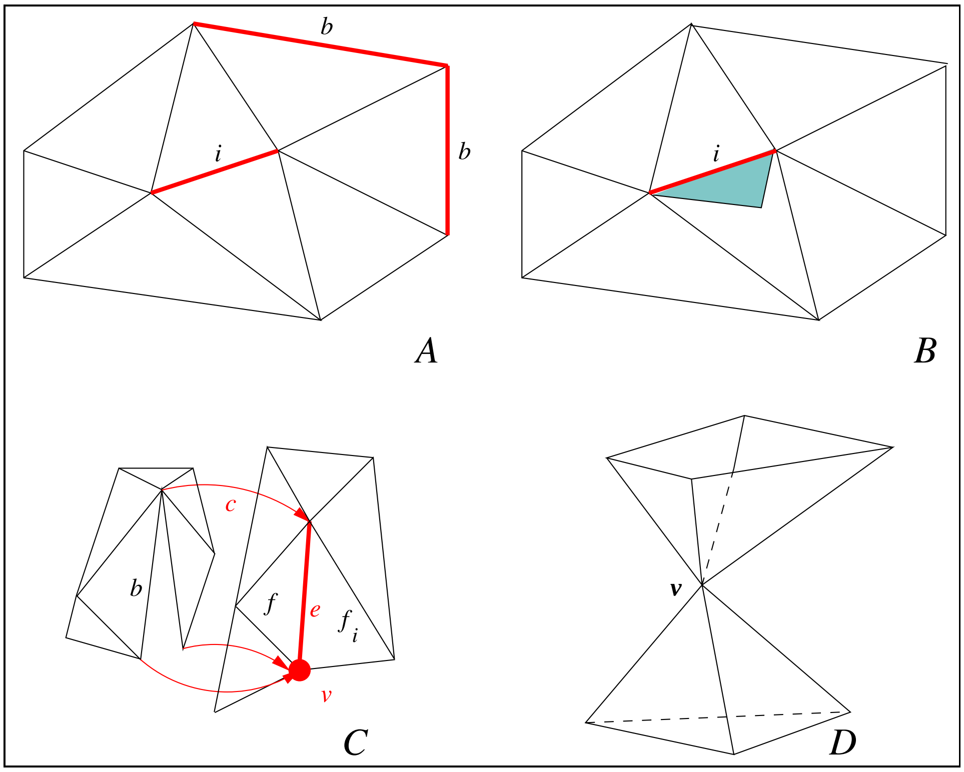


Figure 4: A: edge marks that do not produce a surface cut; Edge i is an *isolated interior marked edge* with one *corolla*; Edges b are marked boundary edges. B: singular edges of type i produce fewer boundary edges after cutting than the number of incident faces. C: after cutting through the marked edges, for each marked edge e and each incident face f not belonging to a corolla, a boundary edge g is created. D: an isolated singular vertex $\mathbf{v}$.

The global cutting method requires the specification of a set of marked edges. We call it a global method because it operates on all the faces and vertices of the surface. The result is a cut through the marked edges as well as a cut through the isolated singular vertices.

This method first creates a new surface Σ from the original surface S by breaking all adjacencies between faces. There are as many vertices in Σ as corners in S. In Fig. 5A, we show the corners $w_0 \ldots w_{20}$ obtained by completely disconnecting $v_5^\star$.

We then define a partition of the face corners as follows: we visit the non-marked edges in S one after the other. For each non-marked edge, we retrieve the faces sharing that edge in S, and the face corners corresponding to the edge endpoints in Σ. For each edge endpoint, we express that the corners corresponding to that endpoint belong to the same group of corners. Once all the non-marked edges are visited, the corner groups correspond to the vertices in the surface resulting from the cut. In a look-up table, we record the mapping from corner groups to the vertices of S before the cut.

We illustrate the result of the global cutting method on $v_5^\star$ in Fig. 5B, where the corner groups $g_0 \ldots g_{16}$ are shown. The example configuration that we use is the same as in Fig. 3; however, as the method is global, all vertices in the configuration are affected, and not only v_5. The worst case

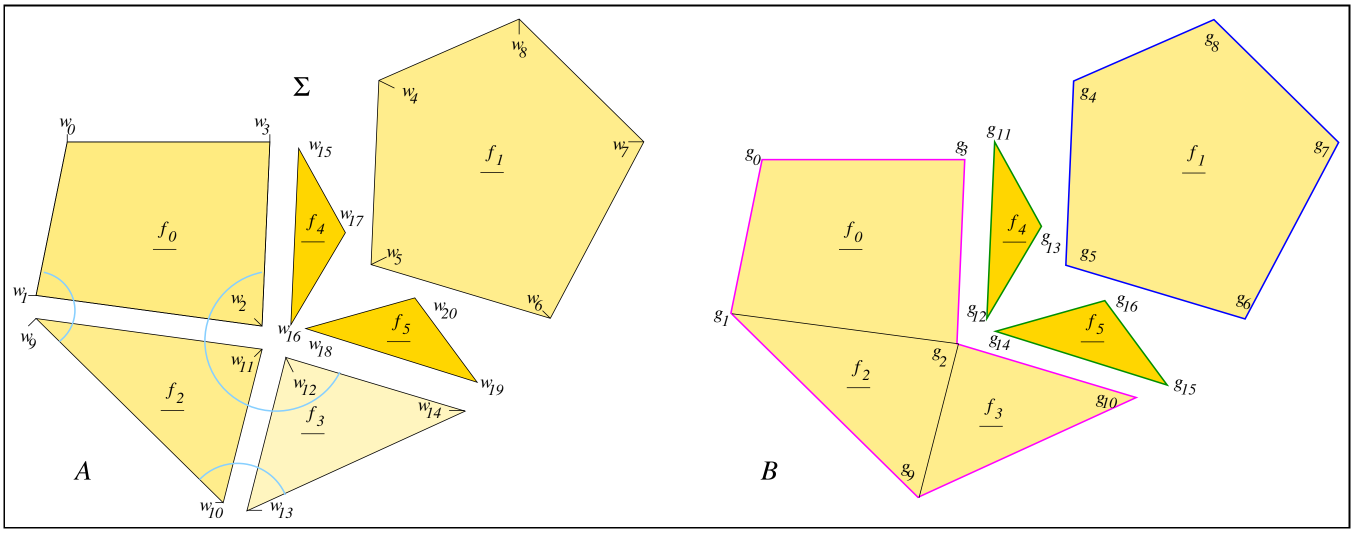


Figure 5: Global cutting. A: corner groups are shown using circular arcs. B: result of global cutting.

complexity of the method is linear in the total number of corners (to first cut the surface entirely) and in the number of non-marked edges.

Comparison of the two methods. The local and global methods differ by the properties of the global method to implicitly cut through isolated singular vertices, and to eliminate *standalone* vertices, which are vertices without incident faces. Also, the global method eliminates the pre-processing of invalid faces as discussed in the next Section 3.4. The labeling of vertices after the cut is different in both methods, but this does not affect the surface topology.

In the global method, we partition corners associated to a given vertex, deciding to join them if there is a non-marked edge whose adjacent faces contains them. This is the same as partitioning the faces incident on that vertex in the local method. If a vertex is multiplied in the local method, its multiplicity is the number of face components connected by non-marked edges; this is also the number of corner groups created for a given vertex in the global method.

There are cases when one method will be preferred against the other: when the number of marked edges or singular vertices is small with respect to the total numbers of surface edges and vertices, the local method has an advantage because it visits only marked edges and vertices. Alternatively, when the cut covers a large portion of the surface, the global method may be more appropriate. The global method is simpler to use when invalid faces are present.

3.4. Application to the Conversion of Non-Manifold Surfaces

We mark singular edges and apply either the global or the local cutting method using the marked edges. When using the local method, isolated singular vertices must be marked as described in Section 3.2, by looping through the vertex list; recall that the global method implicitely cuts through

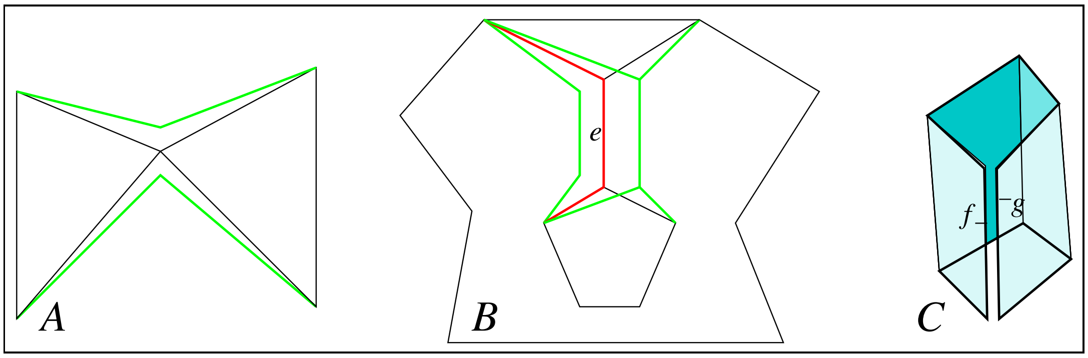


Figure 6: Invalid faces. A: duplicate vertex; the global method automatically produces the correction shown in green. B: face incident on the same edge e multiple times. C: after stitching f and g, the non-planar face C becomes invalid as in B.

isolated singular vertices.

In addition, we present optional pre-processing operations that apply when the input to the conversion is not a surface (according to our definition): for instance, some faces may be invalid, or some vertices may not be referenced in any face.

3.4.1. Pre-Processing Operations. When the local method is used, we label standalone vertices when looping through the face list by counting the number of face incidences for each vertex. We eliminate the standalone vertices, renumber the remaining vertices, and update the vertex indices in faces in a second loop through the face list. The elimination of standalone vertices is implicit with the global method.

Faces of length three (triangles) can only be invalid if they are geometrically degenerate, with two or more coincident vertices; there are more cases for faces of length four or higher as illustrated in Fig. 6. By default, we propose to convert invalid faces as follows; we also offer the possibility of simply removing them if they are not too numerous. Faces can be invalid because vertices are duplicated (Fig. 6A). In this case, it is best to use the global cutting method that will automatically produce the polygon marked in green in Fig. 6A. Another possibility for an invalid face is to be incident on the same edge multiple times (Fig. 6B,C). In this case, we mark the edge and for each endpoint, we inquire whether there is another marked edge of the invalid face incident on that endpoint; if this is not the case, we mark an incident edge; marked edges are shown in red in Fig. 6B; the global method produces the solution marked in green in Fig. 6B. If we still wish to use the local method for converting the surface, we can pre-process the faces to simulate the corrections obtained with the global method.

Labeling Singular Edges. Singular edges are labeled when building the edge data structure, as the edges that are shared by at least three different faces. If we wish to build an orientable manifold, we may mark regular edges for which the incident face orientations are inconsistent.

Building an Oriented Manifold. To build an oriented manifold, we must verify that faces incident on each edge are consistently oriented. We have explored two solutions: the first solution consists of marking every edge for which the face orientations are inconsistent, and of cutting through such edges.

The second solution consists of cutting through singular edges only in a first step, and of propagating the orientation of the resulting manifold using a spanning tree of surface faces. The number of required operations is proportional to the total number of faces. After propagating the orientation, some edges may still have inconsistently oriented incident faces. In a second step, we mark these edges and cut along them. In general, there will be fewer edge marks than with the first solution.

3.5. Why Cutting Works

The previous method works for converting non-manifold surfaces to manifolds because after cutting, by construction each vertex has a star formed of a single component of faces connected through regular edges, which means that the star is topologically equivalent either to a disk or to a half disk. If in addition we have cut through edges whose incident faces are inconsistently oriented, then after cutting each vertex has a consistently oriented star. This method was successfully applied to pre-process non-manifold polygonal surfaces before simplification in IBM Data Explorer [37].

At this stage, every edge is regular, and every edge mapped by c to a non-isolated singular edge is a boundary edge. We cannot break any other connection, or increase the size of the cut without disconnecting faces incident on a regular edge.

4. Stitching

Stitching corresponds to "identifying" boundary edges. Formally, we define a *surface stitch* as the inverse operation of a cut through marked edges of a manifold surface: given a surface Σ we write that a manifold surface S is obtained from Σ by stitching if there is cut through marked edges c that maps S to Σ. Clearly, Σ must be a manifold as well. In Fig. 1, C is a manifold and a surface cut of both D and E; D and E are obtained from C by stitching. We do not consider cuts through isolated vertices, as stitching, the inverse operation, would create singular vertices. Guaranteeing that S is a manifold surface is one of the difficulties of stitching. We define the *length* and the *size* of a surface stitch as respectively the length and the size of the corresponding surface cut. The basic stitching operation is called an *edge stitch*; it turns out that it is an operation that we already performed for the global cutting method of Section 3.3: we consider an edge that was identified in two different faces, and we group the corners corresponding to the edge endpoints in the two faces.

Fig 7B illustrates a sequence of Edge Stitches 1, 2, 3 and 4 that produce a non-manifold surface in Fig 7C. We write that such edge stitches are *incompatible*. Supposing that the input to our conversion algorithm were the surface in Fig 7A, very similar to C (where the red edges would be identified as well as the green edges) we would obtain the surface in B after cutting using either method of Section 3, and unfortunate stitching choices would create in C a non-manifold surface very similar to the original A. We address this problem specifically and develop in Section 4.1 a test that forbids to create a non-manifold surface after stitching. Performing the stitches 1, 2, 3 and 4 in Fig. 7 creates

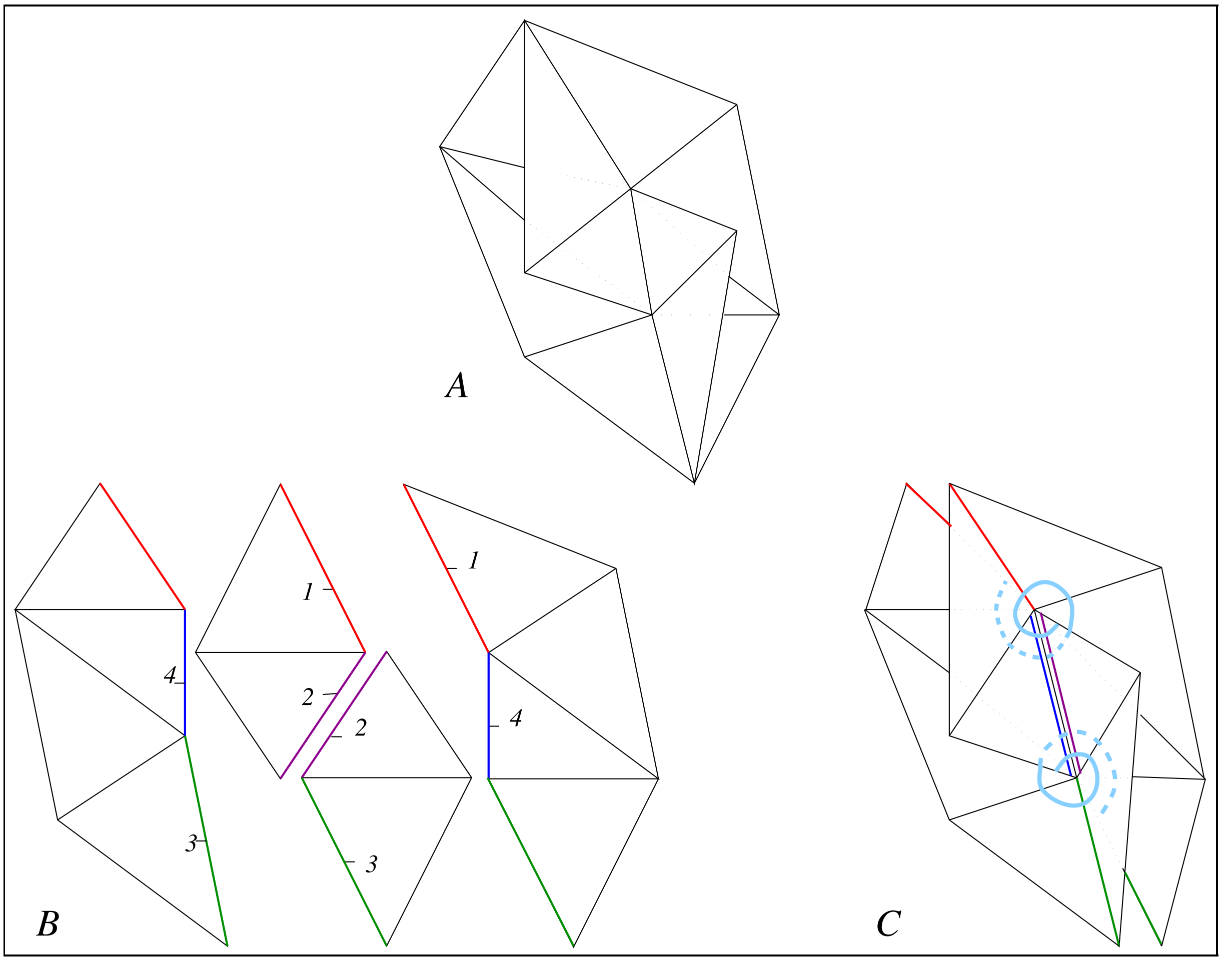


Figure 7: A: a non-manifold surface, resulting in the surfaces of B after cutting. B: an incompatible sequence of edge stitches, resulting in C in a non-manifold. C: spirals indicate which corners are identified (grouped) after stitching.

a singular edge and two singular vertices. However, any combination of three or fewer edge stitches among this four does not introduce any singularity: deciding whether a set of stitches are compatible cannot be solved by testing independently whether pairs of stitches are compatible.

As illustrated in Fig. 1, and discussed in Section 3.1, the default behavior of the cutting stage using either local or global methods is to produce a surface cut of maximum length. We can produce various cuts of smaller size by stitching, unless the additional components and boundaries that were created during the cutting stage are considered useful. In general, we believe that it is preferable to limit the number of additional vertices and boundary edges in a converted surface, because of (1) additional storage requirements, (2) inconsistent topological and geometric properties: a surface may geometrically represent a solid as in Fig. 1C, while its topology indicates that it has a boundary; relatedly, in Solid Modeling, it has been argued that representing geometrically coincident vertices using several copies of the same vertex can yield inconsistent results when computing surface intersections in floating point [38].

We record which edges can be stitched using a relation $\mathcal{R}$ between boundary vertices. Two boundary edges (a, b), (c, d) are *stitchable* if either ($a\mathcal{R}c$ and $b\mathcal{R}d$) or ($a\mathcal{R}d$ and $b\mathcal{R}c$). If $a\mathcal{R}b$ implies that a and b have the same coordinates then the stitching operation will not change the geometry of the surface. Preferably, the relation $\mathcal{R}$ will be: $a\mathcal{R}b \iff c_e(a) = c_e(b)$, whereby we stitch the edges that were previously cut. The c mapping is stored in a look-up table of multiplied vertices described in Section 3. $\mathcal{R}$ can also be provided by a proximity relation, which can be useful for other applications than converting non-manifold surfaces (see Section 4.3).

Given a collection of stitchable edge pairs, potentially interesting goals are to produce a stitch of maximum length or maximum size. In Fig. 1, D and E are both stitches of maximum length, but only E is a stitch of maximum size. This is because D has two components whereas E has only one, and the difference in the number of vertices comes from a different number of components (two versus one) reflected on the Euler number, $n_v - n_e + n_f$ (the number of edges are the same as both stitches are of maximum length, and by definition the number of faces must be the same). In the case of stitching edges that were cut, if there were no isolated marked edges, an upper bound to the length of the stitch can be obtained by rounding down to an even number the number of face incidences for each marked edge (before cutting), and by summing on marked edges. In turns out that the length of the stitch depends upon the order in which edge stitches are chosen: returning to Fig 7, if we start by performing Edge Stitches 1, 2 and 3 in any order, only three edge stitches can be performed. Here, an upper bound is 4 edge stitches; it can be reached by performing for instance Edge Stitches 1, 2, 4 and stitching a pair of green edges (not the 3-3 pair). Thus choosing Edge Stitches 1, 2 and 3 is a sub-optimal choice for a longest stitch: we may need to backtrack to find a better solution.

Armed with theses observations, for this article we propose two different greedy strategies for stitching. Strategy I consists of keeping the components that were created after cutting, and of stitching within the same boundaries. When stitching through the same boundary, as we will prove, it is impossible to create a non-manifold. Strategy II consists of stitching along different boundaries, whereby a test needs to be developed to avoid creating singular edges, as illustrated in Figs. 7,8 and 11.

4.1. Testing whether an Edge Stitch is Valid

A *valid* edge stitch is such that no singularities are created after stitching. We test whether two edges $e_1 = (v_1, v_3)$ and $e_2 = (v_2, v_4)$ may be stitched by joining v_1 and v_2 and v_3 and v_4. (For convenience, we make no distinction between vertices or vertex indices and use normal typesetting for both.) Testing that an edge stitch is valid amounts to observing the star of both edge endpoints after the stitch, and verifying that they are still homeomorphic to a disk or half disk. A necessary and sufficient condition is that there should be no common vertices in the stars of the vertices v_1 and v_2 and v_3 and v_4, except for vertices incident on a single face (boundaries of the link). The test is visualized in Fig. 8. We retrieve the vertices of the star of v_1, for instance v_3, v_5, v_6, v_7 and v_8, and the vertices of the star of v_2: $v_4, v_9, v_{10}, v_{11}, v_{12}$ in Fig. 8. Such vertices can be ordered starting respectively from v_3 and from v_4 independently of whether the surface is orientable or not. Our test prescribes that $v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$ must all be different, except maybe for v_8 and v_{12} and for v_3 and v_4. An algorithm implementing this test is described in Section 4.3. If in addition we require the resulting surface to be oriented we may have to verify that the orientations of the faces incident on e_1

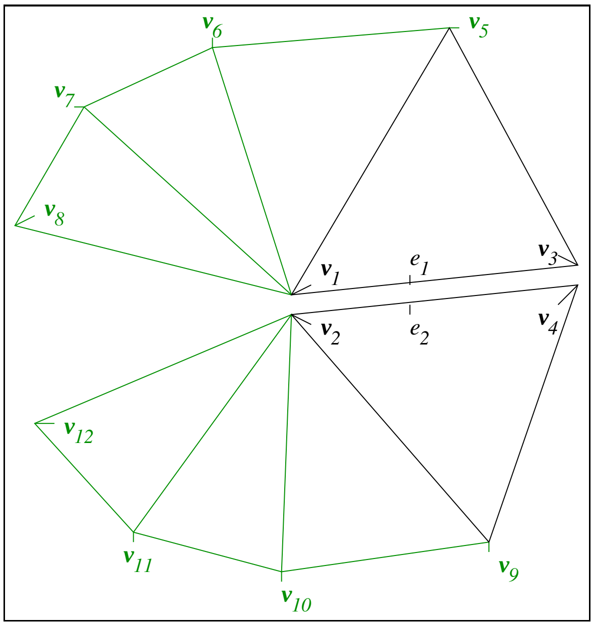


Figure 8: Examining the stars of boundary vertices to determine whether stitching Edges e_1 and e_2 introduces any singularity.

and e_2 are consistent, as explained in Section 4.3.

The reader may wonder why this test was not mentioned in Section 3.4 that develops the global cutting method for conversion of a non-manifold surface to a manifold surface. The reason is that it is not necessary. In Section 3.4 we stitch only edges that were regular edges before cutting. A common vertex index in the stars could only result from an adjacent singular edge, which using our assumption, was cut. Also, there is no ambiguity in which edge stitches to perform, as only two faces share each regular edges.

4.2. Strategy I: Stitching Along the Same Boundary Obtained after Cutting

We organize the boundary edges per surface component. For each component, we choose a pair of adjacent stitchable edges and stitch them: we diminish by two the length of the boundary. Starting from the first edge stitch, we then verify whether the adjacent pair of edges on the boundary are stitchable and if so, we stitch them. We continue until the next pair of edges is not stitchable. We repeat the operation of searching for an adjacent pair of stitchable edges.

One advantage of this strategy is that the test of Section 4.1 can be omitted. Firstly, as we stitch adjacent edges, we only have to worry about one pair of vertices v_1 and v_2 to be identified. Secondly, suppose that when trying to identify v_1 and v_2 we would observe that they are both adjacent to v_0, such that (v_0, v_1) and (v_0, v_2) are not both boundary edges. Assuming that we stitch only edges that were cut ($a\mathcal{R}b \iff c_e(a) = c_e(b)$), we know that before cutting, v_1 and v_2 were identified, meaning that (v_0, v_1) and (v_0, v_2) were the same singular edge: accordingly, a cut was made through that singular edge. Without loss of generality, we assume that (v_0, v_1) is not a boundary edge, meaning that some

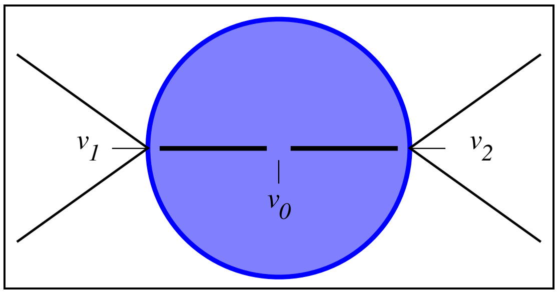


Figure 9: Strategy I: impossible configuration. After cutting, v_0 was a boundary vertex; now it is an interior vertex adjacent to two boundary vertices v_1 and v_2: this configuration cannot be obtained by following Strategy I of pinching adjacent boundary edges.

edge was stitched to it after cutting. Using Strategy I, when stitching an edge, at least one endpoint must be an interior vertex. v_1 is a boundary vertex so v_0 must be an interior vertex. v_0 is an interior vertex that is adjacent to v_1 and v_2, which are both boundary vertices. After cutting, v_0 was a boundary vertex; now it is an interior vertex adjacent to two boundary vertices v_1 and v_2. As shown in Fig. 9, v_0 has been stitched in a manner that does not follow Strategy I. More generally, applying Strategy I to a loop of boundary edges (or to a boundary), results in a loop of boundary edges to which trees of stitched (formerly singular) edges are attached.

If the original surface before cutting represents a solid, this strategy for stitching has the effect of breaking all connections of zero width, thus regularizing the solid by computing its interior (see Appendix A). This is not true if singular edges form a graph on the surface that is not a forest: each loop of singular edges would yield two disconnected boundaries after cutting (without necessarily disconnecting the underlying surface: consider a cut through the shorter generating circle of a tore).

Having several components is useful for selecting connected portions of surface and operating on them independently from the rest of the components. In some cases, most components except a few can be rejected because they correspond to noise or have no impact on a visualization. We believe that this is the appropriate strategy in the example of Fig. 14, of a polygonal approximation to an iso-surface representing the jaw bone of a primate fossil. Although it may be reasonable to create non-manifold iso-surfaces in the general case, in this case a simple model used for visualization would consist of one connected manifold surface homeomorphic to a sphere, with potentially a few boundary curves. It turned out that the model represented in Fig. 14 had about five hundred singular vertices. After cutting, most of the nine connected components were inside the largest component and thus invisible, assuming an opaque surface. All but one component could be removed after cutting, and the boundaries of the remaining component stitched. Another justification for keeping the components obtained after cutting is that some algorithms perform better when the surfaces are separated in many components. This is the case of Rossignac and Borrel's surface simplification using vertex clustering [39].

4.3. Strategy II: Stitching Along Different Boundaries

Different stitching strategies will, in general, result in different surface topologies. In the previous section, stitching is only allowed between edges that belong to the same boundary and are identified prior to cutting. The number of connected components resulting from the cutting operation is left unchanged after the stitching operation. This strategy leads to a simple and efficient algorithm. However, it is sometimes preferable to allow stitching between edges that were not previously identified. For instance, since there is an overhead cost for every connected component, surface compression techniques such as the ones used in [9, 10] tend to give better results for surfaces with few connected components.

For the compression method of Taubin and Rossignac [10] the ideal stitching strategy would minimize both the number of vertices and the number of connected components. For example, suppose the surface is specified by a set of disconnected faces. Further suppose that the coordinates of vertices of such faces contain small, unintended discrepancies. For this case, it may be useful to stitch edges whose vertices are geometrically close to one another. Since vertices with different coordinates can be identified, the geometry of the resultant surface might change. However, the number of faces and the number of vertices per face will not be affected. Finding an optimal stitching strategy for the purpose of compression is a difficult task and we were unable to find such a strategy. We next present the strategy that was implemented; we describe tests to insure that the stitches are valid. Our methods were successfully applied for pre-processing 650 VRML models before compression.

Stitching Strategy. We start by specifying when a pair of boundary edges is stitchable and the order in which such pairs will be stitched. We consider two edges to be stitchable if each of their corresponding endpoints are located within an ϵ distance. We choose ϵ to be a fraction of the length of the shortest edge. To avoid a quadratic number of comparisons between boundary edges, we cluster the edges in an octree-like structure constructed using the distance between edge centers. To build the structure, we first compute a bounding box containing all the edge centers and then recursively subdivide it into two parts on the longest side. The boxes are enlarged by $\epsilon/2$, and, as a result, edges may appear in several boxes as shown in Fig. 10. With this enlargement, neighboring boxes need not be visited when looking for a stitching candidate. The recursion is stopped when either the side of a box becomes smaller than ϵ or the number of edges in a box is less than a fixed number p. In practice, we use $p = 20$.

We consider in turn each pair of edges in each leaf box of the octree. When we encounter a pair of edges whose endpoints meet the ϵ distance criterion we verify whether the edge stitch is valid, and if so, we perform the stitch. To minimize the number of connected components we visit the octree twice: in the first pass we only try to stitch edges from different connected components; after this pass all the potentially stitchable edges must belong to the same connected component; in the second pass we attempt to stitch any pair of edges.

Tests for Determining Valid Stitches. As explained in Section 3.3, the global cutting method operates by grouping corners. At the end of this process, a surface is defined by identifying every corner group with a new vertex. An edge stitch can be viewed as a continuation of the grouping

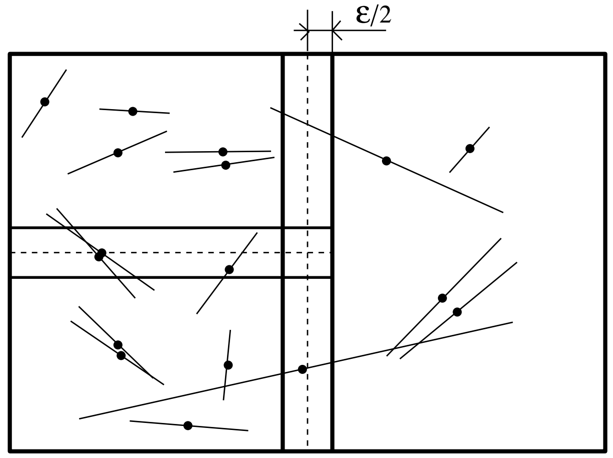


Figure 10: Bounding box subdivisions with a maximum of ten edges per box.

process. After each stitch, a new surface can be defined by identifying each corner group with a new vertex. The forthcoming tests verify that the surface resulting from a proposed stitch is a manifold. Since the vertex identifications are induced by edge stitching, the link of every vertex always belongs to a single connected component. As a consequence, no isolated singular vertex can appear. Thus, we must only test for the creation of singular edges. We next describe an algorithm implementing the test presented in Section 4.1: because of the cost of finding candidate edges for stitching, we allow to merge only one of the two pairs of endpoints when this creates no singularity; accordingly the test is decomposed in several steps.

At each step of the stitching process, each vertex of the surface corresponds to a group of corners and each edge corresponds to two groups of corners. To avoid any confusion with the edges of the original cut surface we call such edges *current edges*. Therefore, a stitch is performed by stitching two current edges; this in turn is performed by merging each of the two pairs of corner groups that define the endpoints of two current edges. Since the surface is a manifold before the stitch, every current edge is incident to one or two faces. This condition must also hold after the stitch. Only current edges incident to one of the four vertices involved in the stitch may be affected by the stitch. These edges must, by definition, belong to one of the stars of the four vertices. Suppose that we wish to stitch the two current edges (v_0, v_1) and (v'_1, v'_0) by merging v_0 with v'_1 and v_1 with v'_0. As shown in Fig. 11, several configurations may occur. In this figure circles represent groups of corners (vertices) and lines represent boundary edges of the cut surface. A current edge is represented by two circles connected by at least one edge. The manifold property requires that no more than two edges connect the same circles. The configurations can be partitioned into three classes:

Class I The stars of v_0 and v_1 do not intersect the stars of v'_0 and v'_1. This case is illustrated Fig. 11A; the stitch is valid; it can be performed.

Class II Either (v_0, v'_1) or (v_1, v'_0) is a current edge. Fig. 11B shows this configuration. The stitch cannot be performed since it would create a self-loop edge which is prohibited in our surface model.

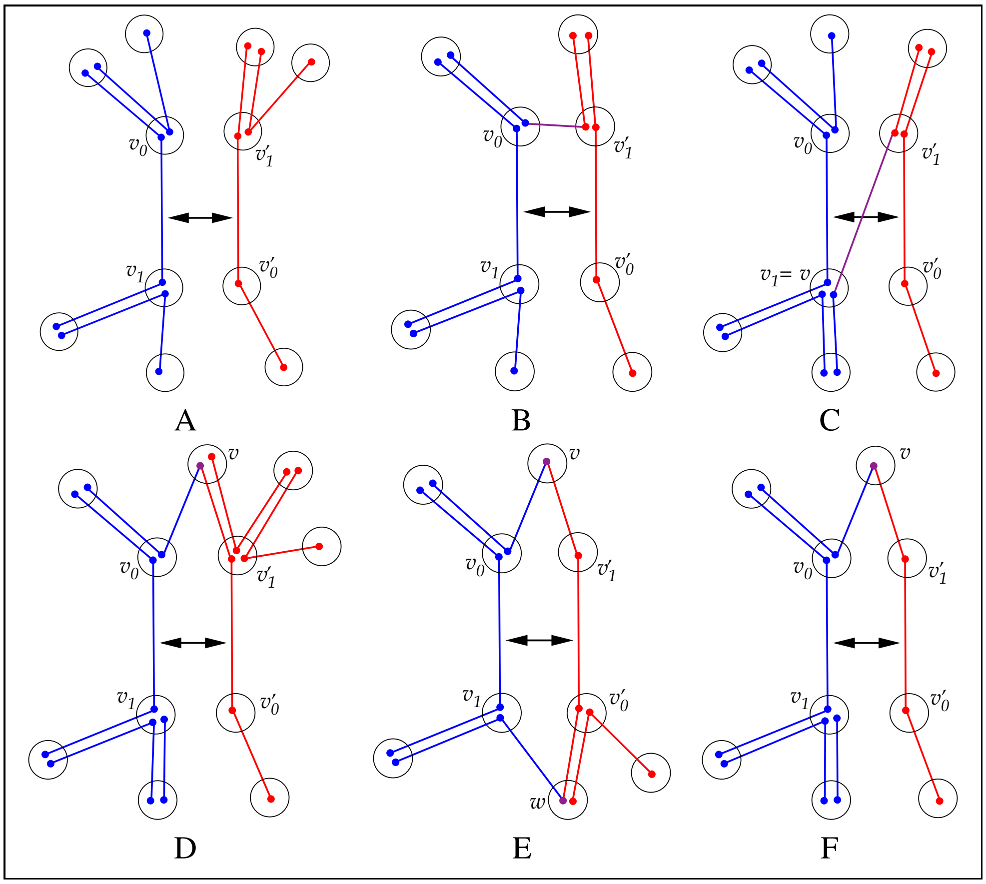


Figure 11: Different configurations for a proposed stitch between (v_0, v_1) and (v_1', v_0').

Class III There are two current edges of the form (v, v_0) and (v, v_1') or of the form (v, v_1) and (v, v_0'). Several such configurations are shown in Figs. 11C,D,E and F. Fig. 11C illustrates the case where $v = v_1$. Here the stitch is invalid, since the stitched edge would be incident to three faces. For similar reasons, stitches cannot be performed for the configurations of Fig. 11D and Fig. 11E. However, the configuration of Fig. 11F yields no singular edge. In this last case, stitching (v_0, v_1) and (v_1', v_0') implies stitching (v, v_0) and (v, v_1'). We call this last stitch an *implicit stitch*. In contrast, we refer to the stitch between (v_0, v_1) and (v_1', v_0') as an *explicit stitch*. Explicit stitches that yield a non-manifold surface are rejected. However, in the process of rejecting a proposed explicit stitch we may encounter a valid implicit stitch, in which case we merge the corresponding corner groups. This is the case for the configurations shown in Figs. 11C and 11E.

Firstly, we reject the stitch if (v_0, v_1') or (v_1, v_0') is an edge. Next, we verify whether Edges (v_0, v_1) and (v_1', v_0') belong to different connected components. If they do we perform the stitch by merging v_0 with v_1' and v_1 with v_0'. Otherwise, we use a procedure called $STAR\_COMPARE(v, v')$ to evaluate the effect of merging two corner groups, v and v'. $STAR\_COMPARE$ classifies a proposed merging as one of three types: (1) creates at least one singular edge, (2) creates no singular edge and creates no implicit stitch, (3) or creates no singular edge and creates one or two implicit stitches. Given two stitchable edges (v_0, v_1) and (v_1', v_0') we perform the following steps:

Step I We evaluate $STAR\_COMPARE(v_0, v_1')$ and perform the merging if there is an implicit stitch (3).

Step II We evaluate $STAR\_COMPARE(v_1, v_0')$ and perform the merging if there is an implicit stitch (3).

Step III If one of the two mergings was performed (3) and if the other does not create a singular edge and no implicit stitch (2) then perform the merging.

Step IV If neither merging was performed and if both mergings would not create any singular edges (2) then perform both mergings.

It is important to perform the two first steps sequentially. As shown Fig. 11C, although neither merging creates a singular edge by itself, the explicit stitch leads to a singular edge. In this case, the above procedure will merge v_0 with v_1' in Step I. However, $STAR\_COMPARE(v_1, v_0')$ will prevent the second merging in Step II. Figure 11D illustrates another case where order is important. When a merging does not create an implicit stitch we cannot perform the merging before we know that the other merging can also be performed. For instance, in Fig. 11D, if we merge v_1 with v_0' we would create a singular vertex since v_0 cannot be merged with v_1' and the link of the new vertex corresponding to v_1 merged with v_0' would have two connected components.

$STAR\_COMPARE$ works by maintaining a list of current edges incident to the corners of a group (an edge can appear at most once); this list is accessed through the corner representative. Before any edge is stitched, the list is initialized with all incident edges to a corner group, boundary or not. For each current edge, a flag indicates if it is a boundary edge. $STAR\_COMPARE$ verifies whether any edge is repeated in two lists. for each repetition, if both edges are boundary edges, then there is an implicit stitch, otherwise, a singular edge would be created when merging.

Orientability. If we wish to have an oriented surface, firstly, we enforce the orientability of the input surface using the cutting methods of Section 3. Secondly, we orient consistently the faces of the different surface connected components: we maintain a partition on the faces into connected components; each face also carries an orientation bit indicating whether the ordering of its vertices (its orientation) should be kept or reversed. The orientations of the various components are subject to change when stitching. The orientation bit of a face composed with the orientation bit of the component representative provide the *current* orientation of a connected component.

When stitching two disconnected components, we update the current orientations to make them consistent across the stitched edges: we update the orientation bit of the representative of one of the components. When stitching edges of the same component, implicit stitches do not affect the orientability, but explicit stitches may affect the orientability: In Step IV, we retrieve the current orientations of the faces incident on the (boundary) edges (v_0, v_1) and (v_1', v_0') and we make sure that they are consistent. Otherwise, we do not perform the stitch.

5. Examples

5.1. Conversion to a Manifold Surface

Invalid faces and singular vertices and edges are frequent in real world geometric data. The following examples illustrate some of these singularities and the benefits of using our methods. We show three examples of conversions, performed using the cutting algorithm followed with either Strategy I or Strategy II for stitching. As the effect of the conversion is of pure topological nature, it is essentially "invisible" in a display; however, we use the following artifices to show the various steps of our methods: we use different colors for boundary edges, regular edges and singular edges; we highlight singular vertices; in certain illustrations, we may disconnect geometrically adjoining boundary edges; we may also use different colors for painting the faces belonging to different connected components;

The first example is a polygonal CAD model of a desk lamp in Fig. 12A. In Fig. 12B,C,D we show successive close-up views of the lamp shade, where several singular edges were detected. In Fig. 12, which is obtained after conversion, we paint disconnected surface components using different colors. The original model had 5054 triangles and 2810 vertices; we discovered 125 singular edges and 128 singular vertices, shown in Fig. 13. After conversion, obtained by cutting through singular edges and vertices, there were 5052 triangles and 3058 vertices. The conversion took less than one second using an IBM RS6000 58f. No stitching could be performed using Strategy I.

The second example is a polygonal approximation of an iso-surface extracted from a CT-scan of a fossil monkey jaw, illustrated in Fig. 14A. The original model had 75842 triangles and 37624 vertices. We discovered 462 singular edges and 563 singular vertices; singular edges are shown in Fig. 14B and singular vertices in Fig. 14C. Invalid triangles with duplicate vertex indices contribute to the singular edge and vertex count: they are incident to the same edge (consecutive vertex index pair) twice, and provided the triangle shares that edge with neighboring triangles, the edge is singular. Such singularities resulted from flaws in the algorithm extracting the iso-surface, whose intended purpose was to build closed manifold surfaces. (When building an iso-surface from regularly sampled data, invalid triangles may be created when data values are equal to the chosen iso-level. A solution for avoiding invalid triangles is provided in [40].) After removing invalid triangles, we discovered 2 remaining non adjacent singular edges, one of which is shown in Figs. 14E,F and 10 singular vertices.

After cutting through singular edges and vertices, there were 75371 triangles and 37636 vertices. The conversion took 6 seconds with an IBM RS6000 58f workstation, including the removal of invalid faces. No stitching could be performed using Strategy I. For this example, it is preferable to use the local cutting method rather than the global cutting method, since the number of singular edges and vertices is very small after removing invalid faces.

The third example is a polygonal model of the space ship Enterprise with 12539 triangles and 15011 vertices. Fig. 15A is a global view of the model, where disconnected surface components are painted with different colors. We discovered 594 singular edges and 1878 singular vertices. After removing invalid triangles, there were 435 remaining singular edges and 1689 remaining singular vertices. After conversion and stitching using Strategy II, there were 12552 triangles and 7429 vertices. The conversion took 21 seconds using an IBM Power PC 42T. Fig. 15B shows a detail of

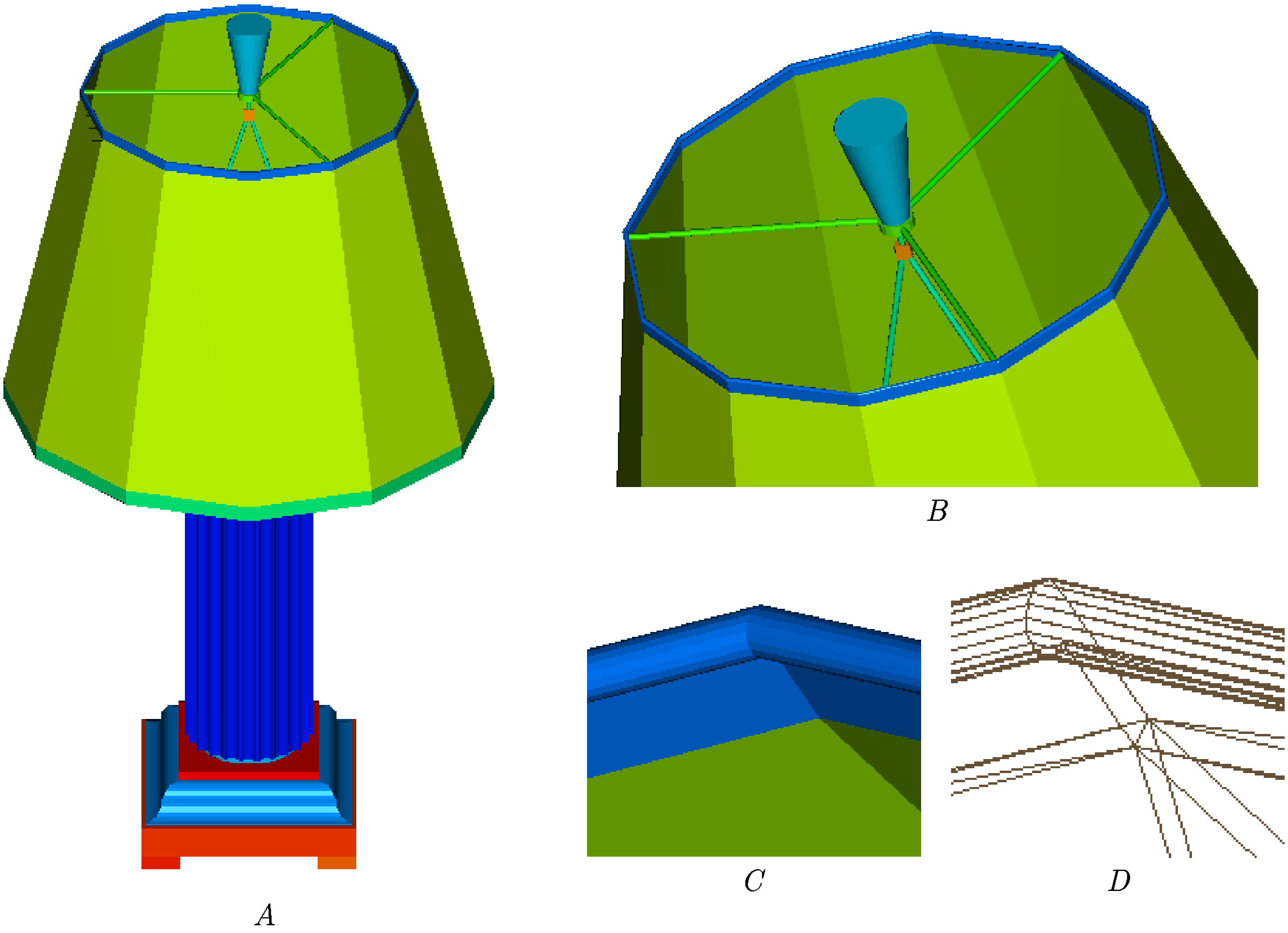


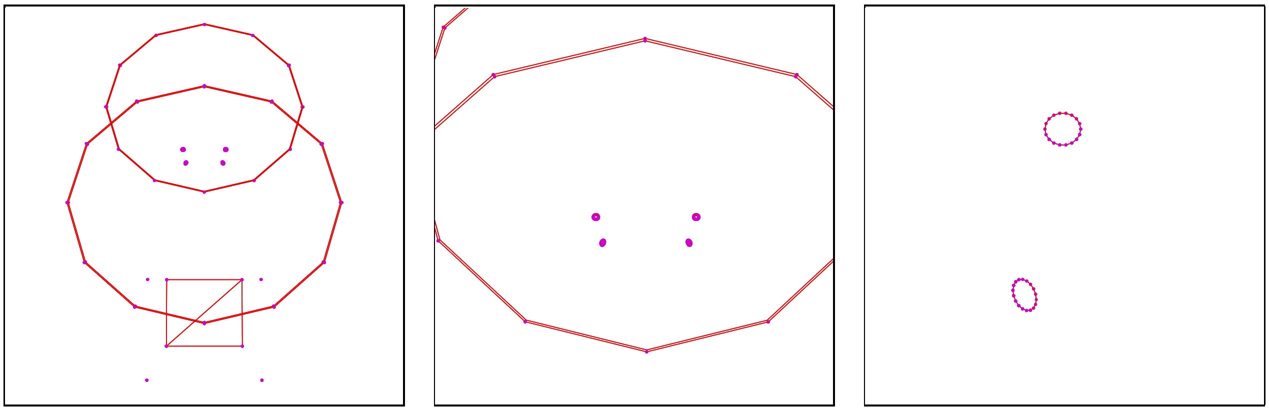
Figure 12: Lamp model. A: general view. B, C, D: successive details showing edges shared by more than two faces.

the model before eliminating invalid faces; Fig. 15C shows the same view after eliminating invalid faces; Fig. 15C shows the result of cutting through singular edges and vertices, and Fig. 15C shows the result of stitching using Strategy II.

We believe that this example is a good advocate for automatic conversion methods: asking a user to decide on how to locally connect the surface about one thousand five hundred times seems tedious and meaningless (this number may actually be lower, as some decisions may affect simultaneously a collection of adjacent singular vertices and edges).

5.2. Digression: Surface Smoothing and Singularities

We now describe another application for our methods: an algorithm can terminate normally in the presence of singular edges and vertices but deliver unintended results. We consider the surface subdivision and smoothing algorithm of Taubin [3]: designed for use on a manifold surface, it operates on a non-manifold as well; however, the resulting surface is not smooth in the vicinity of the singular



A B C

Figure 13: Singular edges of the Lamp model of Fig. 12 are shown in red; singular vertices are shown in magenta. A, B and C are increasingly detailed views.

edges and vertices. We use the example of two spheres sharing two edges, providing a non-manifold model in Fig. 16A. In Fig. 16B we attempt to subdivide and smooth A, and notice the non-smooth behavior in the vicinity of the singular edges. In Fig. 16C, we first cut through the singular edges using the methods of Section 3 and then subdivide and smooth. In Fig. 16D, we smooth after cutting and stitching using strategy I. In Fig. 16E, we smooth after cutting and stitching using strategy II. Fig. 16F illustrates another solution when stitching using strategy II as well.

6. Conclusion

In this paper, we have formally defined cutting and stitching through a surface, and have applied cutting and stitching to the automatic conversion of a polygonal surface to a manifold polygonal surface, that can potentially exhibit self-intersections (which are not treated). The result of the conversion is a surface with the same geometrical realization (expect maybe using Strategy II for stitching), but potentially with a different topology (or connectivity). As discussed in Sections 3 and 4, several surface cuts yield a manifold surface. Starting with a cut of maximum length, we can apply different stitching strategies to reduce the number of boundary edges and vertices and connected components.

We have successfully applied this conversion to extend a simplification algorithm[13] as well as a surface compression algorithm [10], so that they can process non-manifold surfaces. This method may not be suitable if the original surface was intended to be a non-manifold, and if topological singularities (singular edges and vertices) are an integral part of the model. Otherwise, it is general and handles any type of topological singularity without human intervention.

Aside from reducing the number of boundary edges after converting to a manifold, another application of stitching is to join topologically disconnected but geometrically adjoining surface components, which we found useful for optimizing surfaces before compressing them using Taubin and Rossignac's method.

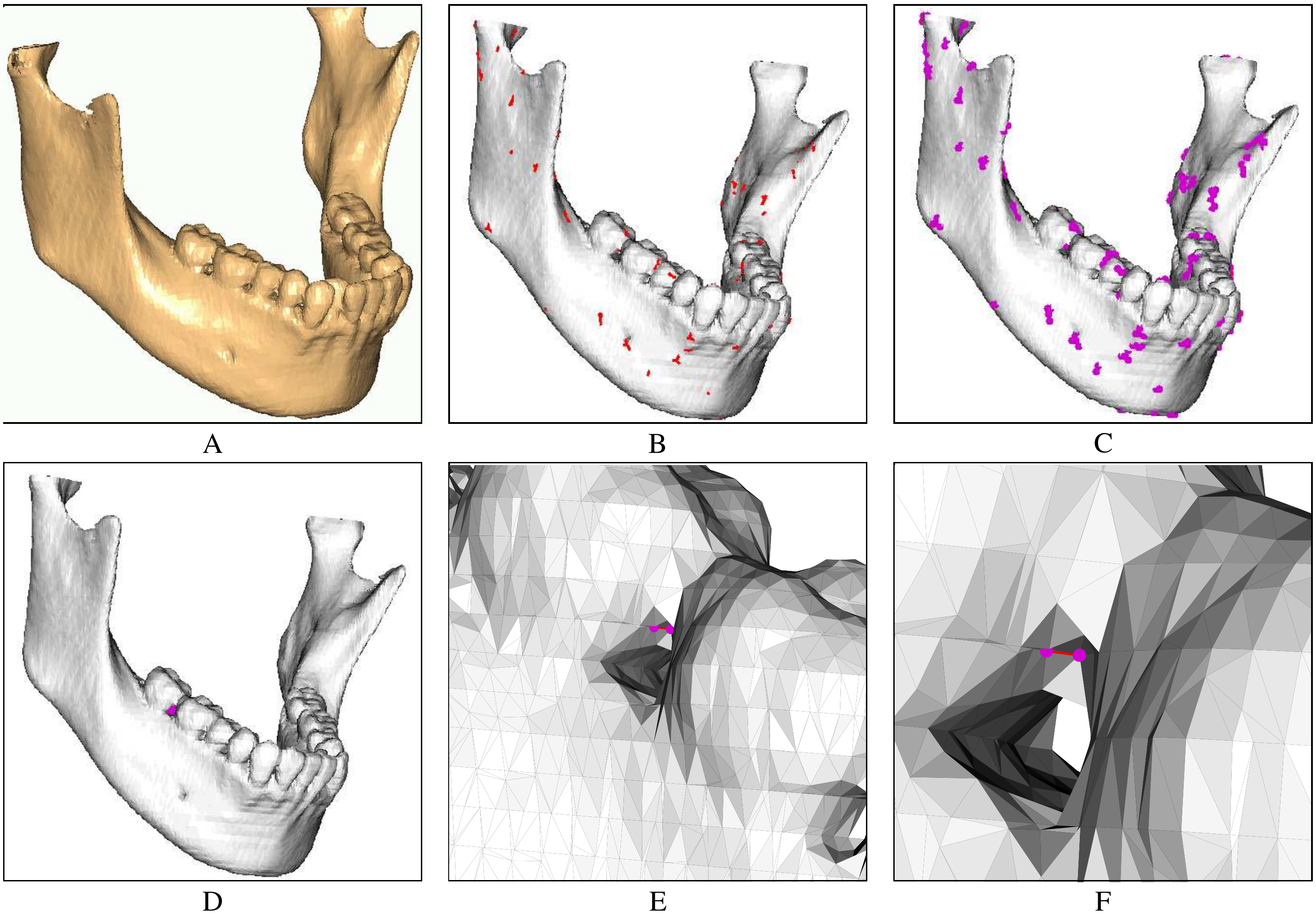


Figure 14: Non-manifold iso-surface. A: general view. B: singular edges are shown in red. C: singular vertices are shown in magenta. D: singular elements after removing invalid triangles. E, F: close-up views of D.

A. Simplicial Complexes and Solids

Simplicial complexes are frequently referenced in the Computer Graphics and Solid Modeling literature. For completeness, we review them briefly. A *n-simplex* is defined as the convex hull of $n+1$ linearly independent vertices: a 0-simplex is a vertex, a 1-simplex a segment, a 2-simplex a triangle. A *face* of a simplex σ is a simplex defined using a subset of the vertices defining σ. A (geometric) simplicial complex is a collection of simplices, such that each simplex face is in the collection and any two simplices can only intersect at a face of each of them. The dimension of a complex is the highest dimension of its simplices. A homogeneous simplicial complex of dimension n is such that every simplex is either of dimension n or incident to a simplex of dimension n (in dimension two every edge and every vertex are incident to at least one triangle ([15, 33]). If a surface is made exclusively of triangles, such that two triangles can only intersect at a common edge or vertex,

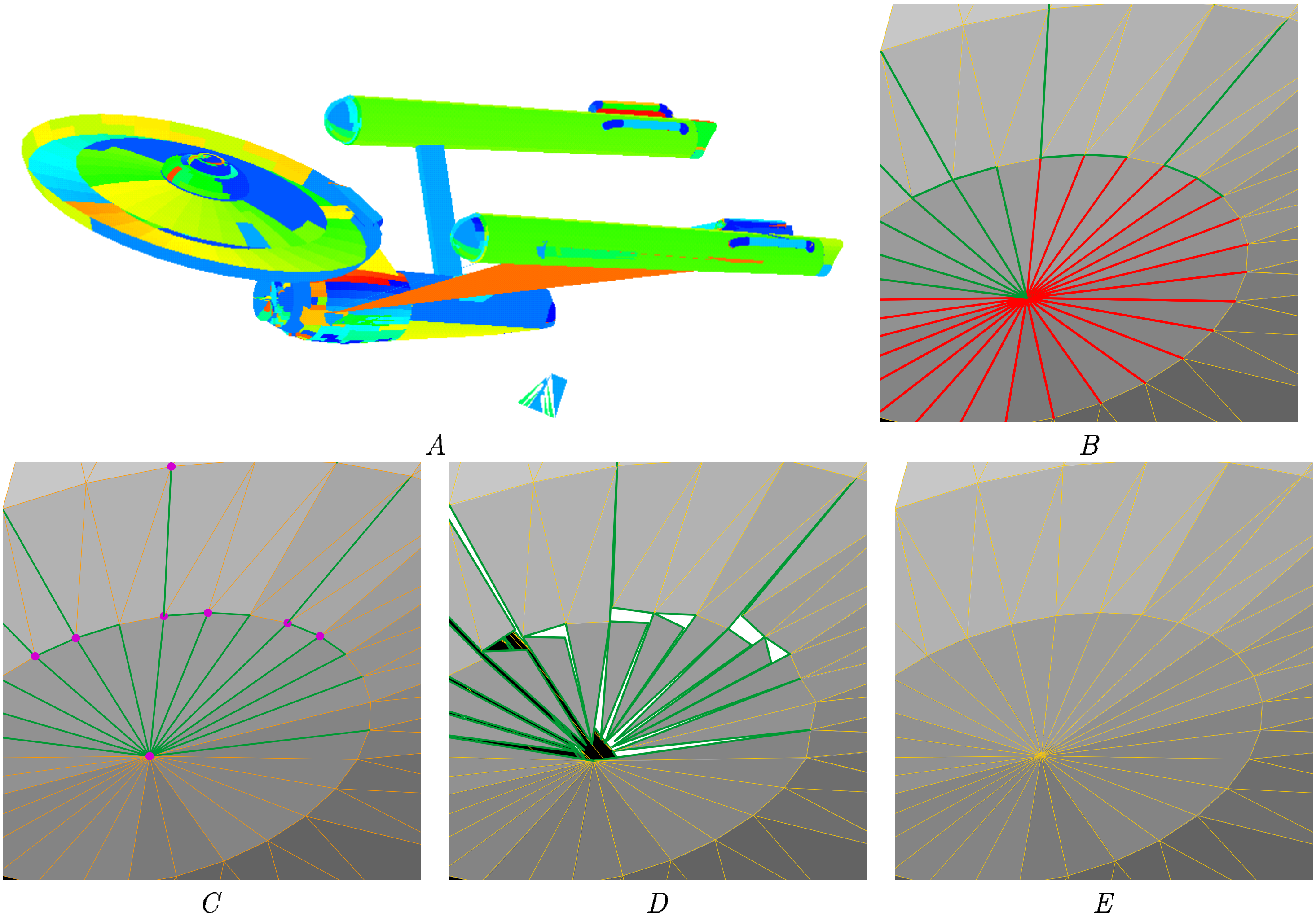


Figure 15: Enterprise model. A: general view: disconnected surface components are painted using different colors. B: detail: singular edges are shown in red; boundary edges are shown in green; regular edges are shown in orange. C: same as B, after removing invalid triangles; singular vertices are shown in magenta; no more singular edges are visible in this area. D: same as C, after cutting through singular edges and vertices. For illustrative purposes, adjoining boundary edges are shown disconnected; in reality, endpoints are very close to each other (with respect to the unit length, defined as the shortest edge length in the model). E: same as D, after stitching using Strategy II.

a surface defines a *simplicial complex* of dimension two.

A *non-manifold solid* is a homogeneous simplicial complex of dimension three whose boundary is not a manifold, meaning that solid bodies can share isolated vertices or networks of edges. A *manifold solid* is a homogeneous simplicial complex of dimension three whose boundary is a (necessarily closed) manifold simplicial complex of dimension two.

In CAD and Constructive Solid Geometry research, regularized sets or r-sets (see Requicha [41])

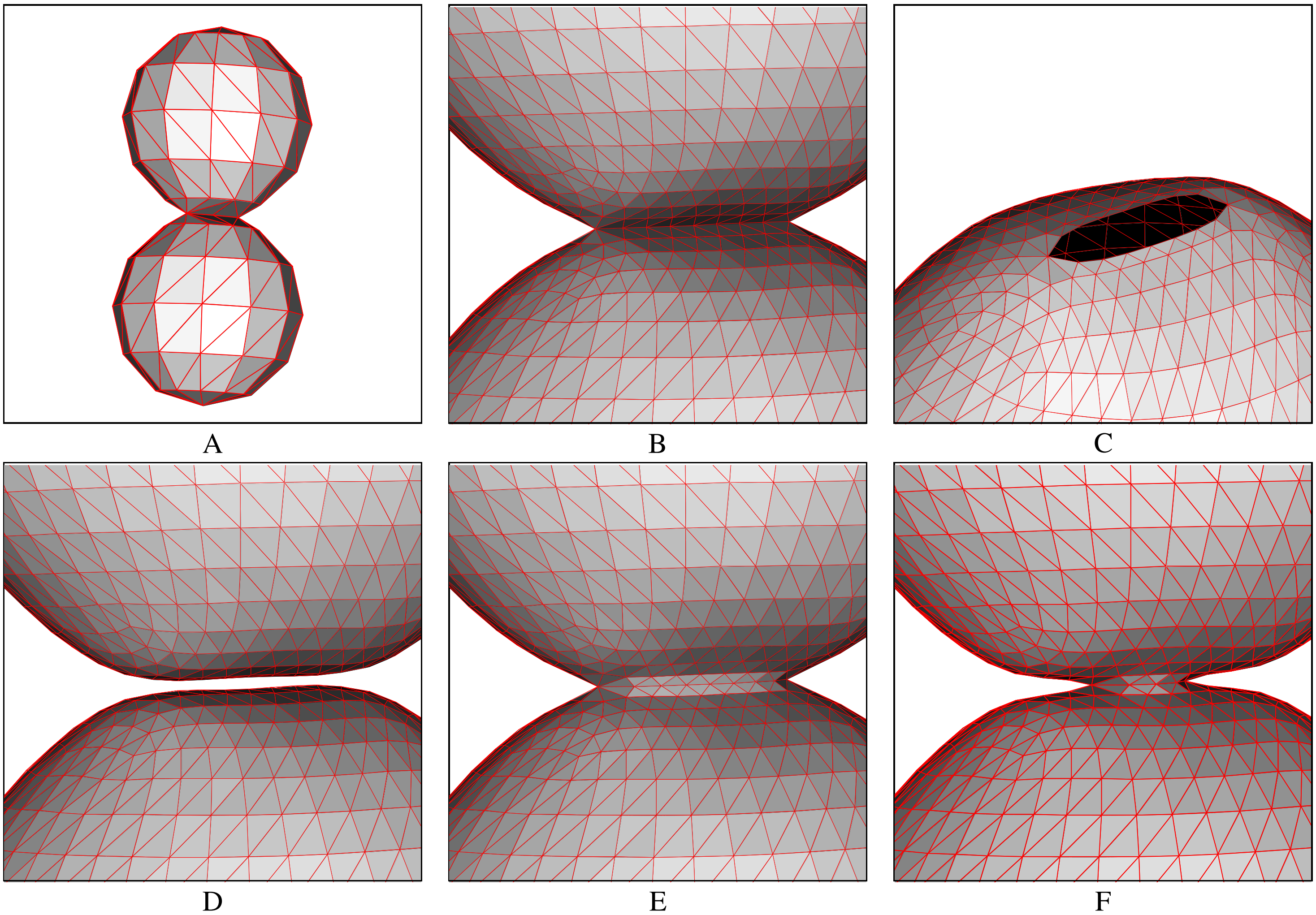


Figure 16: A: non-manifold model of two spheres sharing two edges. B: after subdividing and smoothing. C: after cutting, subdividing and smoothing. D: after stitching with Strategy I, subdividing and smoothing. E: after stitching with Strategy II, subdividing and smoothing. F: another result of stitching using Strategy II.

have been presented as preferable models for solid objects. An r-set is essentially what Hoffmann refers to as a non-manifold solid [33]. An r-set can exhibit connections of zero width. R-sets have over other models the theoretical advantage of representing a closed set over regularized boolean intersection and complement (and union) operations. Non-manifold representations for r-set boundaries have the reported advantage of allowing more consistent implementations of boolean operations (see Hoffmann *et al* [38, 33]). Desaulnier and Stewart [42] study the relationship between manifold solids and r-sets. Boolean operations are known to present severe difficulties in the presence of roundoff errors and uncertainties in vertex positions. Non-manifold representations have an advantage because, in general terms, geometrically coincident vertices or edges can be represented without duplication.

References.

[1] B. Hamann. Curvature approximation for triangulated surfaces. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling*, Computing Suppl. 8, pages 139–153. Springer-Verlag, New York, 1993.

[2] F. Lazarus and A. Verroust. Décomposition cylindrique de Polyèdre et courbe squelette. *Revue de CFAO et D'Informatique Graphique*, 11(4):363–377, 1996.

[3] G. Taubin. A signal processing approach to fair surface design. In *Siggraph*, pages 351–358, Los Angeles, August 1995. ACM.

[4] G. Huiskamp. Difference formulas for the surface laplacian on a triangulated surface. *Journal of Computational Physics*, 95:477–496, 1991.

[5] A. Yamada, K. Shimada, and T. Itoh. Energy minimizing approach to meshing curved wire-frame models. In *5th International Meshing Roundtable*, pages 179–189, 1996.

[6] A. Guéziec. Surface simplification inside a tolerance volume. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, New York, March 1997. revised version of RC 20440.

[7] S. Doo and M. Sabin. Analysis of the behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10(6):356–360, 1978.

[8] D. Zorin, P. Schroder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Siggraph'96 Conference Proceedings*, pages 189–92. ACM, Addison Wesley, August 1996.

[9] M. Deering. Geometric compression. In *Siggraph'95 Conference Proceedings*, pages 13–20, Los Angeles, August 1995.

[10] G. Taubin and J. Rossignac. Geometry Compression through Topological Surgery. Technical Report RC-20340, IBM Research Division, January 1996. `http://www.research.watson.ibm.com/vrml/binary/pdfs/ibm20340.pdf`.

[11] J.H. Bohn. Removing zero-volume parts from cad models for layered manufacturing. *IEEE Computer Graphics & Applications*, 15(6):27–34, November 1995.

[12] W.S. Massey. *Algebraic Topology: An Introduction*. Graduate Texts in Mathematics. Springer Verlag, 1967.

[13] A. Guéziec. Surface simplification with variable tolerance. In *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery*, pages 132–139, Baltimore, November 1995.

[14] X. Sheng and I.R. Meier. Generating topological structures for surface models. *IEEE Computer Graphics & Applications*, 15(6):35–41, November 1995.

[15] M. K. Agoston. *Algebraic Topology. A First Course*. Pure and Applied Mathematics. Marcel Dekker, Inc., New York, 1976.

[16] J.H. Bohn and M.J. Wozny. A topology based approach for shell closure. In P.R. Wilson et al., editor, *Geometric Modeling for Product Realization*, pages 297–319. Elsevier, 1992.

[17] M. Mäntylä. Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics*, 5(1):1–29, jan 1986.

[18] A. Varshney. *Hierarchical Geometric Approximations*. PhD thesis, University of North Carolina at Chapel Hill, 1994.

[19] D. Luebke. Hierarchical structures for dynamic polygonal simplification. Technical Report

TR96-006, University of North Carolina at Chapel Hill, January 1996.

[20] M.C. Bailey. Tele-manufacturing:rapid prototyping on the internet. *IEEE Computer Graphics & Applications*, 15(6):20–26, November 1995.

[21] B. G. Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, Stanford University, 1974.

[22] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *Computer Graphics & Applications*, 5(1):21–40, January 1985.

[23] M. Karasick. *On the Representation and Manipulation of Rigid Solids*. PhD thesis, Cornell University, 1989.

[24] A. Kalvin. *Segmentation and surface-based modeling of objects in three-dimensional biomedical images*. PhD thesis, New York University, June 1991.

[25] *The Virtual Reality Modeling Language Specification, Version 2.0*, June 1997. http://www.vrml.org/consort/Specs.html.

[26] V. Chandru, S. Manohar, and C.E. Prakash. Voxel-based modeling for layered manufacturing. *IEEE Computer Graphics & Applications*, 15(6):42–47, November 1995.

[27] G. Barequet and M. Sharir. Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design*, 12(2):207–229, 1995.

[28] G. Butlin and C. Stops. CAD data repair. In *5th International Meshing Roundtable*, pages 7–12, 1996.

[29] G. Barequet and S. Kumar. Repairing cad models. In *Visualization 97*, Phoenix, AZ., oct 1997. IEEE.

[30] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Curvature and continuity control in particle-based surface models. In *Geometric Methods in Computer Vision II*, volume 2031-15, pages 172–181. SPIE, July 1993.

[31] W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. In *Siggraph'94 Conference Proceedings*, pages 247–256, Orlando, July 1994. ACM.

[32] P. Veron and J.C. Leon. Static polyhedron simplification using error measurements. *Computer Aided Design*, 29(4):287–298, April 1997.

[33] C.M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, California, 1989.

[34] J.A. Heisserman. *Generative Geometric Design and Boundary Solid Grammars*. PhD thesis, Carnegie Mellon University, may 1991.

[35] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. Mac Graw Hill, 1989.

[36] M. Henle. *A Combinatorial Introduction to Topology*. W.H. Freeman and Company, 1979.

[37] G. Abram and L. Treinish. An extended data-flow architecture for data analysis and visualization. In *Visualization 95*, pages 263–270, Atlanta, GA., oct 1995. IEEE.

[38] C.M. Hoffmann, J.E. Hopcroft, and M.S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics & Applications*, 9(6):50–59, November 1989.

[39] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering. In B. Falcidieno and T.L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, 1993.

[40] A. Guéziec and R. Hummel. Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):328–342, December

1995.

[41] A.A.G. Requicha. Representations for rigid solids: Theory, methods and systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.

[42] H. Desaulniers and N.F. Stewart. An extension of manifold boundary representations to the r-sets. *ACM Transactions on Graphics*, 11(1):40–60, January 1992.

[43] S.J. Rock and M.J. Wozny. Generating topological information from a "bucket of facets". In et. al H. L. Marcus, editor, *Solid Freeform Fabrication Symposium Proceedings*, pages 251–259, University of Texas at Austin, aug 1992.