Cutting and Stitching: Converting Sets of Polygons to Manifold Surfaces

André Guéziec, *Senior Member*, *IEEE*, Gabriel Taubin, *Member*, *IEEE*, Francis Lazarus, and Bill Horn

Abstract—Many real-world polygonal surfaces contain topological singularities that represent a challenge for processes such as simplification, compression, and smoothing. We present an algorithm that removes singularities from nonmanifold sets of polygons to create manifold (optionally oriented) polygonal surfaces. We identify singular vertices and edges, *multiply* singular vertices, and *cut* through singular edges. In an optional *stitching* operation, we maintain the surface as a manifold while joining boundary edges. We present two different edge stitching strategies, called *pinching* and *snapping*. Our algorithm manipulates the surface topology and ignores physical coordinates. Except for the optional stitching, the algorithm has a linear complexity and requires no floating point operations. In addition to introducing new algorithms, we expose the complexity (and pitfalls) associated with stitching. Finally, several real-world examples are studied.

Index Terms—Polygonal surface, topological singularities, manifold, cutting, stitching.

1 Introduction

Polygonal surfaces are a common choice for representing three-dimensional geometric models. Models of this type are used for rendering images in a large variety of disciplines including computer-aided design (CAD) systems, feature animation, scientific visualization, and medical imaging. Many polygonal surfaces contain topological singularities that challenge algorithms designed to operate exclusively on *manifold* surfaces.

We consider only the topological properties of a surface. Topological singularities can be intentionally created by algorithms that, for example, intentionally merge vertices to avoid duplicating physical coordinates or intentionally collapse edges and faces to reduce the polygon count. Singularities are also produced unintentionally by faulty algorithms and as a result of "bugs" in software implementations of correct algorithms. Finally, singularities can be introduced by human modelers using modeling software.

Examples of algorithms that require input free of topological singularities include: algorithms for surface subdivision [1]; algorithms that simplify surfaces [2], [3]; algorithms for surface compression [4]; algorithms for progressive transmission [5]; algorithms that generate polyhedra for Rapid Prototyping [6]. Other algorithms, such as surface smoothing [7], yield undesired results when executed on nonmanifold input. The example of surface smoothing is examined in Section 6.2.

- *A. Guéziec is can be reached at 365 America Av., Sunnyvale, CA 94085. E-mail: gueziec@computer.org.*
- *G. Taubin and B. Horn are with the IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.*
- *F. Lazarus is with IRCOM-SIC (UMR CNRS 6615), SP2MI, Bvd. 3, Teleport 2, B.P. 179, 86960 Futuroscope Cedex, France.*

Manuscript received 4 Dec. 1998; revised 16 Feb. 2000; accepted 19 Oct. 2000. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 109343.

We describe techniques to remove topological singularities from input data so that algorithms that require manifold conditions can be safely used. Specifically, we describe a novel and efficient method for automatically converting a nonmanifold surface to a manifold surface. Our techniques target the removal of singularities from existing topological descriptions. However, one of our techniques (snapping) can be used to create manifold topologies from sets of disconnected polygons.

1.1 An Overview of the Algorithm

Our algorithm can be characterized by two high-level operations: *cutting* and *stitching*. The cutting operation involves disconnecting the surface topology along a set of marked edges and marked vertices. In Section 4, 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. The local method operates only on marked vertices. The global method is more appropriate when there are a large number of topological singularities to correct and the local method is more efficient when there are only few singular elements in a generally correct topology. The cutting operation is followed by the stitching operation. Stitching involves joining two boundary edges while guaranteeing that the surface is a manifold. In Section 5, we present two stitching strategies: *pinching* and *snapping*.

Fig. 1 provides an overview and illustration of our algorithm with a practical example. In this figure, we consider two tetrahedra sharing a common edge. We subdivide the surface of the tetrahedra into smaller triangles to create the surface shown in Fig. 1A. In Fig. 1B, regular edges are shown in orange and singular edges are shown in red. The two disconnected surfaces shown in Fig. 1C are created by "multiplying" vertices. In Fig. 1C, each manifold has a boundary, shown in green. The three singular vertices in Fig. 1B that are shared by two singular

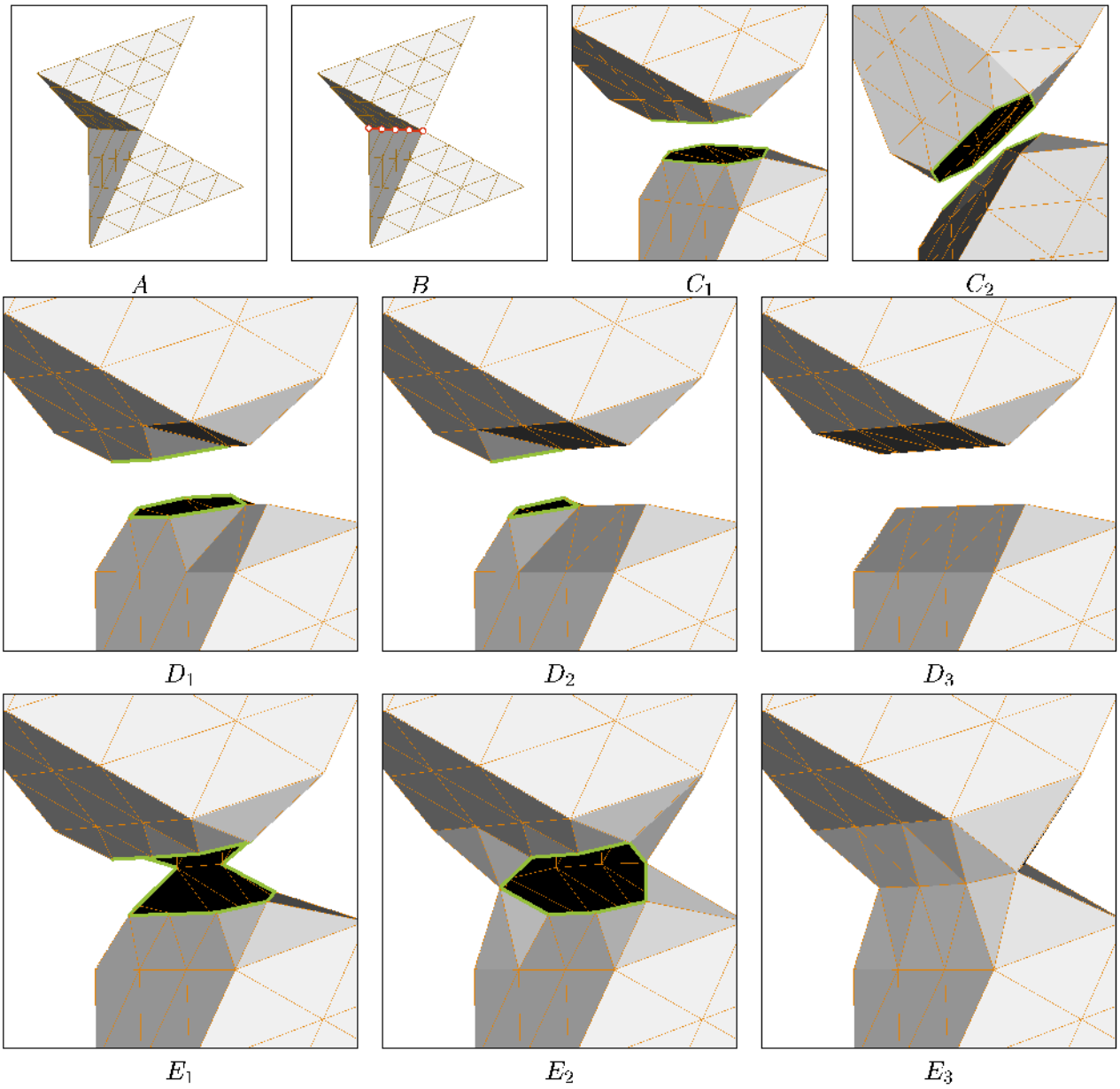

Fig. 1. Converting a nonmanifold surface to a manifold surface. A, B, C: Cutting through singular edges: For illustrative purposes, topologically disconnected vertices are shown physically apart. We implement two stitching strategies: "pinching" edges along the same boundary (D) or "snapping" together edges belonging to different boundaries (E).

edges are each multiplied four times. The two singular vertices shared by one singular edge are only multiplied twice. After stitching along the same boundaries (pinching) we create the two disconnected solids shown in Fig. 1D. However, if we stitch along different boundaries (snapping), we create the single surface with no boundaries shown in Fig. 1E. All three surfaces, C, D, and E, are manifolds with the same geometric realization as A.

1.2 Claim

The application of the cutting and stitching algorithms introduced in this paper will always produce one or more manifold polygonal surfaces from an input polygonal surface as defined in Section 2.

1.3 Overview of the Paper

We define terms related to polygonal surfaces in Section 2. In Section 3, we describe our algorithm for building manifolds from a set of polygons. The cutting and stitching stages of this algorithm are discussed in Section 4 (cutting) and Section 5 (stitching). We examine real-world examples in Section 6. In Section 7, we discuss related work and we present conclusions in Section 8. In Appendices A and B, we discuss data structure and file format issues.

2 Polygonal Surfaces

We define an *abstract polygonal surface* $S(\{v_i\}, \{f_j\})$ as a set of abstract vertices $\{v_i\}$ and a set of faces $\{f_j\}$. Each face is a

cyclically ordered subset of at least three (abstract) vertices.[1] A face is said to be *incident* on its constituent vertices. We will call a vertex with no incident faces a *stand-alone* vertex. A (vertex, incident face) pair is called a *corner*. An *edge* is defined as an unordered pair $\{v_i, v_j\}$ or, equivalently, $\{v_j, v_i\}$, of consecutive vertices in one or more faces. Since all vertices in a face are different, there cannot be self-loop edges with this definition. A face is said to be incident on its constituent edges. The edge $\{v_i, v_j\}$ is said to be incident on the vertices v_i and v_j. v_i and v_j are said to be the *endpoints* of the edge. Two vertices may share at most one incident edge. Two vertices sharing a common edge are said to be *adjacent* vertices. Edges sharing a vertex are said to be adjacent edges and faces sharing an edge are said to be adjacent faces. There are two possible orderings for the vertices of a face, resulting in two orientations for that face.

An abstract polygonal surface can be embedded in $\mathbf{R}^3$ by assigning coordinates to each vertex (and different abstract vertices may be assigned the same coordinates). We use the word geometry to refer to these physical coordinates. There are no particular constraints on the geometry for our methods to apply. For example, the physical coordinates of an embedded face do not have to be planar. Also, properties such as colors, normals, and texture coordinates may be associated with vertices, faces, and corners. We use the word topology to refer to $\{v_i\}$ (abstract vertices) and $\{f_j\}$ (faces). Cutting and stitching only operate on the topology.

In this paper, we are only concerned with abstract polygonal surfaces (with the exception of the Related Work section). For short, we simply write "polygonal surface" or, when there is no ambiguity, "surface," but it is an abstract polygonal surface that we refer to.

We call the subset of faces of $\{f_j\}$ that share a vertex v the *star* of v. The star of v is denoted as $v^\star$. The number of faces in $v^\star$ is called the *valence* of the vertex v. An edge is said to be a *singular edge* if at least three faces are incident on it. Otherwise, it is said to be a *regular edge*. The link of v, or $\ell(v)$, is a graph whose nodes are the faces of $v^\star$. An arc is created between two nodes if the corresponding faces share an edge incident on v. A *regular vertex* is a vertex whose link is either a chain or a cycle.[2] If a vertex is not a regular vertex, then it is said to be a *singular vertex*. A polygonal surface is a *manifold* if each vertex is a regular vertex, otherwise, it is a *nonmanifold* polygonal surface. Both endpoints of a singular edge are singular vertices since their link contains at least one node of degree three. An *isolated singular vertex* is said to be a singular vertex that is not an endpoint of a singular edge. An isolated singular vertex is shown in Fig. 2.

We call an edge with one and only one incident face a *boundary edge*. A connected set of boundary edges is called a *boundary*. A regular endpoint of a boundary edge is called a *boundary vertex*. A regular vertex that is not a boundary vertex is called *interior vertex*. A vertex can thus have one of three types: singular, boundary (regular), interior (regular).

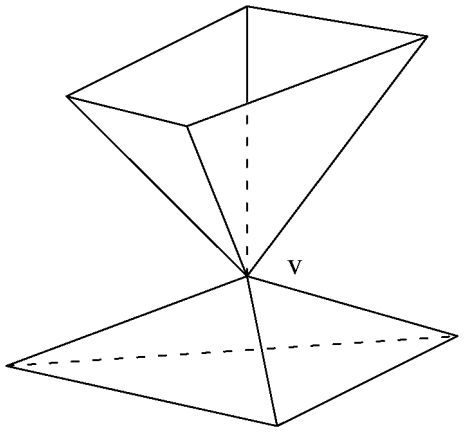

Fig. 2. An isolated singular vertex v.

Two adjacent faces have a *compatible orientation* if the two vertices of each shared edge 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 faces have a compatible orientation. An orientable manifold surface arranged such that its faces are all oriented in a compatible way is said to be *oriented*. A connected orientable manifold surface can be oriented in only two possible ways.

3 Conversion of Nonmanifold Surfaces

Our conversion algorithm is characterized by two high level operations: cutting and stitching. As described in Section 4, cutting involves breaking the surface along singular edges and singular vertices. The result of this operation is a manifold surface. This new surface may contain boundary edges that, under some criteria, are candidates for pairwise merging. As described in Section 5, the goal of the stitching operation is to stitch together these boundary edges to form the final manifold surface. Our conversion algorithm uses the following sequence of steps.

Step I: Find unique edges and identify singular edges. When visiting the face list, we find the unique edges and record each (edge, face) incidence. Edges that are shared by at least three different faces are marked as singular.

Step II: (Local Cutting Method Only) Identify isolated singular vertices. To identify these vertices we begin constructing $v^\star$ for every vertex v that is not an endpoint of a singular edge. We start with an arbitrary face f incident on v and pivot. To pivot, we locate in f an edge e_1 incident on v. From the (edge,face) incidences, we infer a face g that, together with f, shares e_1; we then locate in g an edge incident on v different from e. We continue until we encounter a boundary edge or the first face f. If we encounter a boundary edge, we locate, in the first face, f a second edge $e_2 \neq e_1$ incident on v and pivot in the opposite direction. After completion, we count the number of faces that were visited and compare this number to the valence of v. If the two numbers are different, then we classify the vertex v as an isolated

1. Since a set allows no repetition, there cannot be repeated vertices in a face with this definition. Frequently, input models have "degenerate faces" with repeated vertices. We eliminate such faces before using our algorithms. See Section 6.

2. A cycle is a connected graph where each vertex has degree two and a chain is a connected graph where each vertex has degree one or two. See, for instance, [8].

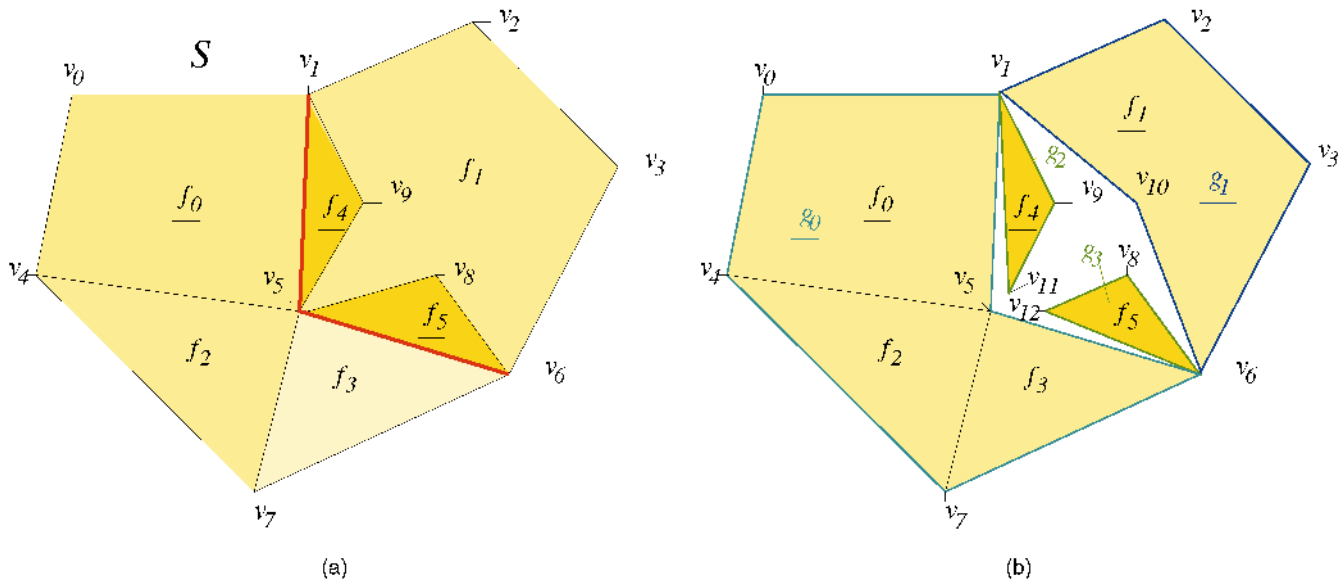

Fig. 3. Local cutting: (a) Star of Vertex v_5 with marked edges in bold. (b) Four sets of faces $g_0, \ldots, g_3$ connected by unmarked edges are identified in $v_5^\star$. v_5 is multiplied into four vertices: v_5, v_{10}, v_{11}, and v_{12}.

singular vertex. This procedure does not require that the surface be orientable.

Step III: Cut. As described in Section 4, we apply either the global cutting method on the marked edges or the local cutting method on the marked edges and on the isolated singular vertices identified in Step II.

Step IV: (Optional) Build an Oriented Manifold. To create an oriented manifold we must verify that each pair of adjacent faces are consistently oriented. This is preferably performed after a first cutting step in order to have a manifold. We build a spanning tree of faces, where two adjacent faces form an arc of the tree only if a consistent orientation can be found across all shared edges, possibly after inverting the orientation of one of the faces. The orientation of each face in the tree is thus consistent with the root face's orientation. We then verify whether face pairs not connected by the spanning tree are consistently oriented. We mark shared edges where the orientation is inconsistent for cutting. (So, the reason for inverting the orientation of some faces is to avoid cutting through every edge where the orientation was originally inconsistent). We then cut along these marked edges to create an oriented manifold.

Step V: (Optional) Stitch. As described in Section 5, we use one of two methods to stitch boundary edges.

4 Cutting

Cutting involves breaking a surface along a collection of marked edges and vertices. New abstract vertices are created and faces are modified to refer to the proper (abstract) vertex. A rigorous definition of cutting (and stitching) can be found in [9]. We propose two methods for cutting through singular edges and vertices: a local method and a global method.

4.1 Local Method for Cutting

We refer to the method described in this section as the *local method for cutting* because it only operates on selected vertices and faces. Starting with the list of singular edges marked in Step I of Section 3, we mark vertices that are endpoints of these edges. We also mark isolated singular vertices that were identified as described in Step II of Section 3. For each marked vertex v_i, we partition the faces of $v_i^\star$ into subsets (connected components) corresponding to equivalence classes under an "is reachable" relation. When constructing the partition, we consider two adjacent faces reachable if and only if they share an unmarked edge which is incident on the candidate vertex.

Once the number of connected components n_c of $v_i^\star$ is known, we create $n_c - 1$ additional copies of the (abstract) vertex v_i with the same coordinates and same properties. Each instance of v_i is labeled from 0 to $n_c - 1$. The instance of v_i labeled 0 corresponds to the original vertex. Instances labeled 1 to $n_c - 1$ are new. For each face f in $v_i^\star$, we replace v_i with the instance of v_i corresponding to the component number of f (the number of the connected component of v_i it belongs to, between 0 and $n_c - 1$). We call this operation *multiplying* the vertex v_i.

The local cutting method is illustrated in Fig. 3. Here, the method is applied to the star of vertex v_5 and its six incident faces $f_0 \ldots f_5$. In Fig. 3a, marked edges are drawn bold. The unmarked edges incident on v_5 are $\{v_4, v_5\}$ and $\{v_5, v_7\}$.

Four sets (connected components) of faces g_0, g_1, g_2, and g_3 are identified in $v_5{}^\star$ (see Fig. 3b). Four copies of the singular vertex v_5 are thus created, each copy being associated to one face set: v_5, v_{10}, v_{11}, and v_{12}. (In Fig. 3b, topologically disconnected faces are shown to be geometrically disconnected for illustrative purposes. In practice, no physical coordinates are modified.) By construction, each copy of v_5 must be a regular vertex (with no incident singular edge and incident faces all connected by regular incident edges: the link is a chain).

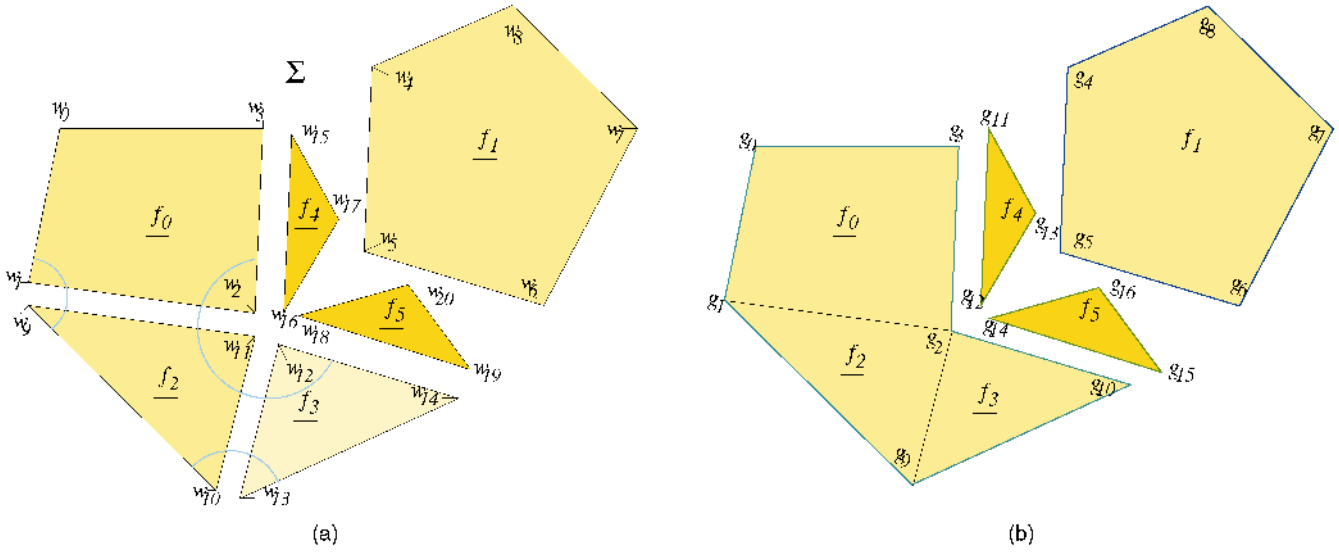

Fig. 4. Global cutting. (a) A new surface is created from the original surface of Fig. 3a by breaking all adjacencies between faces. Corners are then partitioned into groups using the adjacency relationship defined by the unmarked edges (here, $\{v_4, v_5\}$ and $\{v_5, v_7\}$). Corner groups are shown using circular arcs. (b) The result of global cutting. The local method will produce the same result after locally cutting v_1 and v_6.

The cut is completed once all marked vertices have been multiplied. The cost of computing the number of connected components of faces incident on every marked vertex is bounded by the number of marked vertices times the largest valence of a marked vertex. Also, for each vertex v_i, we need to know the relative position of the corresponding corner in incident faces in order to change the corresponding vertex index. The worst case complexity of the local cutting is thus proportional to the number of marked vertices multiplied by the largest valence of a vertex and by the maximum number of vertices in a face.

4.2 Global Method for Cutting

Since the method described in this subsection operates on all of the faces and vertices of the surface, we call it the *global method for cutting*. The method starts by marking all of the singular edges identified in Step I of Section 3. The naive approach of cutting only singular edges fails to detect isolated singular vertices. Instead, we start by creating 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. 4a, we show the corners $w_0 \ldots w_{20}$ obtained by breaking all of the adjacencies in $v_5^\star$ of Fig. 3a. We then partition the corners of Σ into groups using the adjacency relationships defined by the unmarked edges of S. Specifically, for each unmarked edge in S, we retrieve the faces of S that share the edge and for each of the two edge endpoints we identify the pair of corresponding corners in Σ. The identification establishes an equivalence relation which defines a partition on the set of corners. The corner groups (equivalence classes) define a new manifold surface ready to be stitched.

The result of applying the global cutting method on $v_5^\star$ is shown in Fig. 4b. The corner groups $g_0 \ldots g_{16}$ are shown in this figure. The configuration that we use is the same as in Fig. 3a. However, since the method is global, all vertices in the configuration are affected, not only v_5. (The local method will produce the same result as Fig. 4 after locally cutting v_1 and v_6.) The method is linear with respect to the sum of the total number of corners (to break adjacencies) and the number of unmarked edges.

4.3 Comparison of the Two Cutting Methods

The labeling of vertices after the cut is different in both methods, but the surface topology is the same (e.g., in our example v_5 and g_2 refer to the same abstract vertex after cutting, depending upon the method. In a typical implementation, these could be two different integer values assigned to the same vertex). Unlike the local method, the global method implicitly cuts through isolated singular vertices. The global method also implicitly eliminates stand-alone vertices.

There are cases when one method will be preferred over the other. A comparison of the costs for the two cutting methods supports the conclusion that the global method has a lower cost when the cut covers a large portion of the surface. Alternatively, when the number of marked edges or singular vertices is small with respect to the total number of surface edges and vertices, the local method is significantly less costly because it visits only marked edges and vertices.

For a couple of reasons, the global method is slightly simpler to implement. First, as mentioned above, the detection of stand-alone and isolated singular vertices is implicitly performed in the global method. Second, the vertex multiplication process in the local method (Section 4.1) requires searching for the vertex's reference in all its incident faces. This search is avoided in the global method since vertex references in faces are defined during the corner grouping process.

5 Stitching

Stitching simply means identifying two boundary edges. During the global cutting operation, a new surface is defined by identifying corner groups with new vertices.

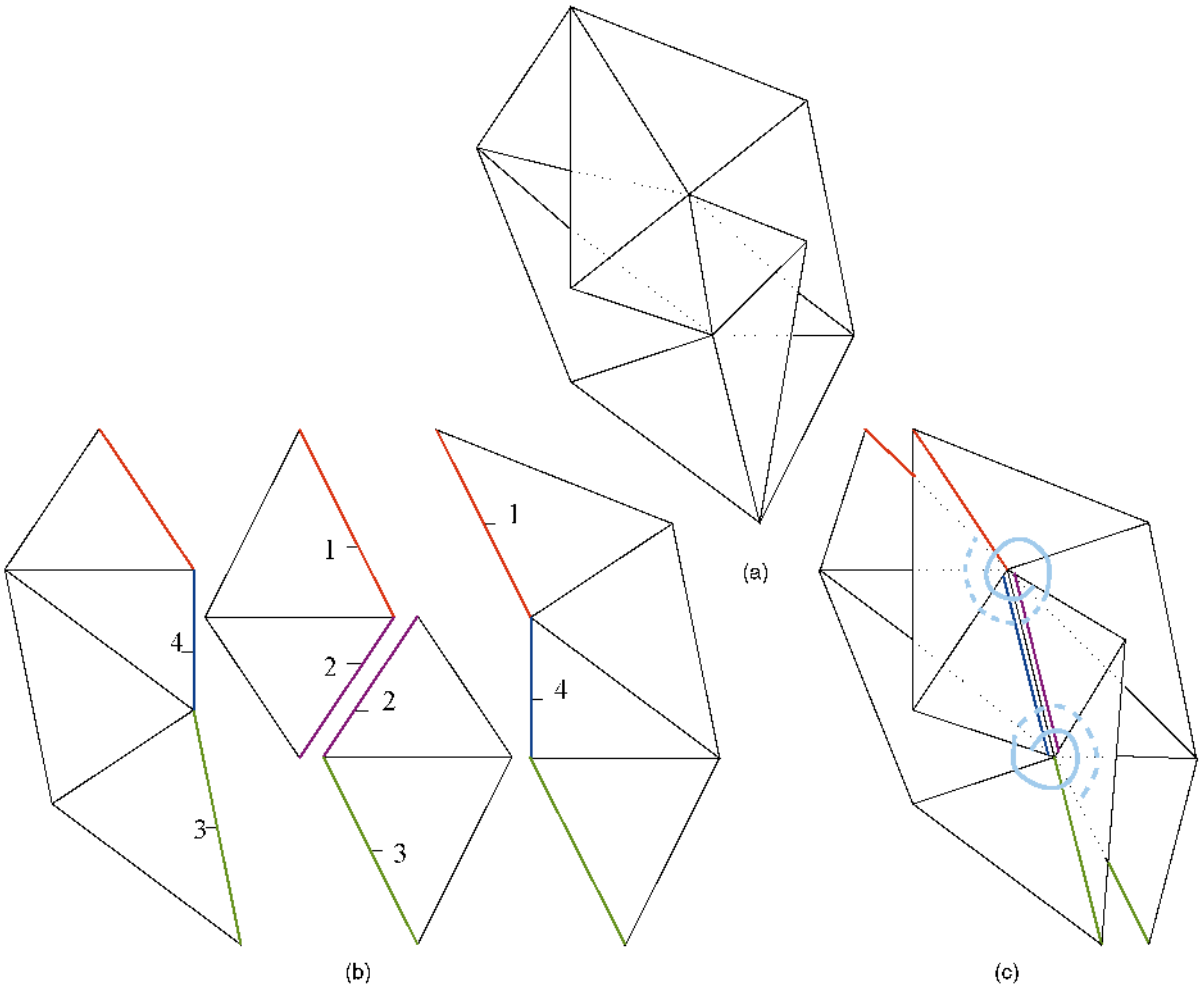

Fig. 5. (a) A nonmanifold surface; after cutting through singular edges, we obtain the surfaces of (b). (b) A sequence of edge stitches (labeled 1, 2, 3, and 4) resulting in a nonmanifold surface in (c). (c) Spirals indicate which corners are identified (grouped) after stitching. Although edges marked with 2 and 4 were not explicitly stitched together, the corner groupings have effectively identified them (only one edge can go through two vertices) and the central edge is now shared by four faces (thus singular).

Stitching can be viewed as a continuation of this grouping process. After each stitch, a new surface can be defined by identifying each corner group with a new vertex.

The application of either the local cutting method or the global cutting method to the surface shown in Fig. 5a will produce the surface shown in Fig. 5b. The sequence of four stitches specified in Fig. 5b will create the nonmanifold surface shown in Fig. 5c. The stitching algorithms in [10] and [11] could produce stitches of this type. Since, in our case, the vertex identifications are induced by edge stitching, no isolated singular vertex can appear; we must only test for the creation of singular edges. We refer to an edge stitch as a *valid stitch* if it does not create a singular edge. Otherwise, it is *invalid*.

How can a singular edge be created? We consider a manifold surface and identify two vertices, v_1 and v_2, when stitching edges: What must we verify to guarantee that the resulting surface contains no singular edge? A singular edge must be incident on the identified pair $\{v_1, v_2\}$. (This is the only way an additional face could become attached to it.) Let us consider v_0, the endpoint of the singular edge different from $\{v_1, v_2\}$. Without loss of generality, suppose v_0 and v_1 are adjacent and consider the three possible cases as in Fig. 6: Case A: v_2 is not adjacent to v_0. v_0 stays regular after identifying v_1 and v_2 (its link is unaffected). Case B_1: v_2 and v_1 are adjacent and both $\{v_0, v_1\}$ and $\{v_0, v_2\}$ are boundary edges: v is regular after identifying v_1 and v_2. Case B_2: A singular edge is created after identifying v_1 and v_2. Thus, the test is: v_1 and v_2 must both be adjacent to some vertex v_0 such that $\{v_0, v_1\}$ and $\{v_0, v_2\}$ are not both boundary edges (B_2).

Stitching strategies. We propose two different greedy strategies for stitching: pinching and snapping. The pinching strategy attempts to stitch boundary edges created during the cutting operation and is discussed in Section 5.1. We will prove that it is impossible to create a nonmanifold surface using this strategy; no explicit test is required when implementing it.

The snapping strategy differs from the pinching strategy in that it attempts to stitch along boundaries other than just those boundary edges created during the cutting operation. For this strategy, special care must be taken to avoid the creation of singular edges. The snapping strategy is discussed in Section 5.2.

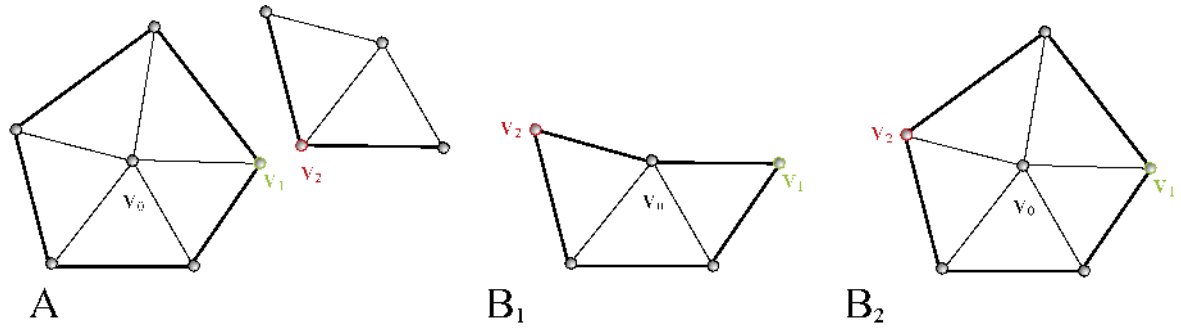

Fig. 6. When identifying two vertices v_1 and v_2, how can a singular edge be created? v_1 and v_2 must both be adjacent to some vertex v_0 such that $\{v_0, v_1\}$ and $\{v_0, v_2\}$ are not both boundary edges (B_2).

5.1 Pinching Strategy

The pinching strategy is illustrated in Fig. 7. Edges are considered to be stitchable only if they have been cut during the cutting operation. We start the pinching operation by computing boundaries (connected sets of adjacent boundary edges). For each boundary, we choose a pair of adjacent stitchable edges and, using their common boundary vertex as a *pivot* vertex, we "pinch" them together. We then check to see if the adjacent pair of edges on the boundary are stitchable. If they are, we stitch them and then continue until the next pair of edges is not stitchable. We then search for an adjacent pair of stitchable edges and repeat the operation.

Using this strategy, it is impossible to create a nonmanifold surface. This can be proven by the following contradiction: As we stitch adjacent edges, exactly one pair of (boundary) vertices, noted v_1 and v_2, is identified when stitching two edges (instead of two pairs of vertices when stitching nonadjacent edges as in Section 5.2). Let us suppose that the identification of v_1 and v_2 would generate a singular edge (we cannot produce an isolated singular vertex, as mentioned previously).

For this to happen, v_1 and v_2 must both be adjacent to some vertex v_0. Also, $\{v_0, v_1\}$ and $\{v_0, v_2\}$ cannot both be boundary edges. (This follows from the above discussion of Fig. 6). Without loss of generality, we assume that $\{v_0, v_1\}$ is not a boundary edge. Because we only pinch edges that were cut, we know that, before cutting, v_1 and v_2 were identified, meaning that $\{v_0, v_1\}$ and $\{v_0, v_2\}$ used to represent the same (singular) edge. The situation is illustrated in Fig. 8A. We also know that a cut was subsequently made through that singular edge (Fig. 8B).

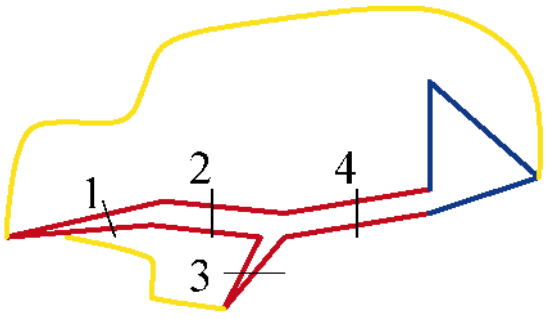

Fig. 7. A typical sequence of stitches produced by following the pinching strategy.

The fact that $\{v_0, v_1\}$ is not a boundary edge implies that some edge was subsequently stitched (pinched) to it. After pinching, the pivot vertex becomes an interior vertex. Since v_1 is a boundary vertex (hypothesis), it follows that v_0 must have been the pivot vertex in a pinching operation and, thus, v_0 must be an interior vertex (Fig. 8C). This configuration is impossible using the pinching strategy. It can only be obtained by stitching nonadjacent edges, which is forbidden with this strategy.

More generally, applying the pinching strategy to a loop of boundary edges (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 and regularizing the solid by computing its interior (see Fig. 1d.) 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.

Pinching does not reduce the number of connected components as the snapping strategy does; the final number of connected components is larger than or equal to the initial number. This may or may not be an advantage. For example, this strategy may help detach and subsequently eliminate small (in terms of the geometry) nonmanifold surface "attachments," such as dangling faces.

5.2 Snapping Strategy

Sometimes it is useful to permit stitching regardless of whether or not two candidate edges had been created during the cutting operation. Also, it may be desirable to permit the stitching of boundary edges that are geometrically close to one another. For example, if a surface is specified as a set of disconnected faces and if the coordinates of vertices of such faces contain small, unintended discrepancies, then it may be desirable to remove these discrepancies and stitch together a manifold representation of the faces. Another example is the creation of suitable input for geometric compression methods that operate most efficiently when the number of connected components is kept to a minimum [4].

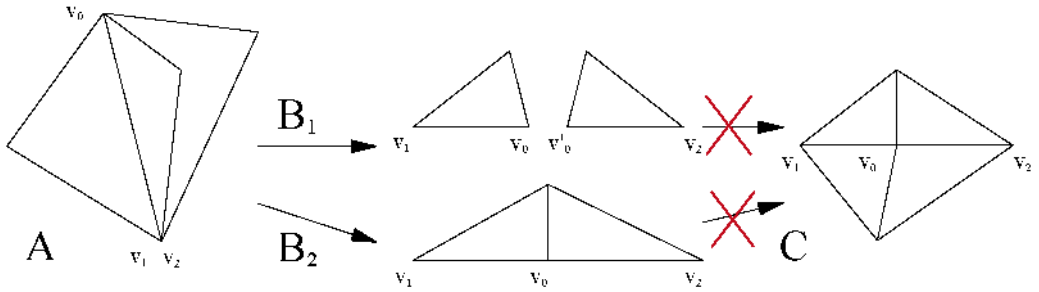

Fig. 8. Validity of pinching: proof by contradiction. A (before cutting): $\{v_0, v_1\}$ and $\{v_0, v_2\}$ were the same singular edge. B (after cutting): v_0, v_1, and v_2 were boundary vertices. C (current situation): How can v_0 be an interior vertex, adjacent to two boundary vertices v_1 and v_2?

5.2.1 An Overview of Snapping

We start the snapping algorithm by deciding 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 $\epsilon/2$ such that neighboring boxes need not be visited when looking for a stitching candidate, as shown in Fig. 9. The subdivision stops 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 check to see if the edge stitch is valid. If it is valid, then we perform the stitch. To minimize the number of connected components, we perform two passes on the octree. In the first pass, we only try to stitch edges from different connected components. After this pass, all the stitchable edge pairs must belong to the same connected component. In the second pass, we attempt to stitch any pair of edges.

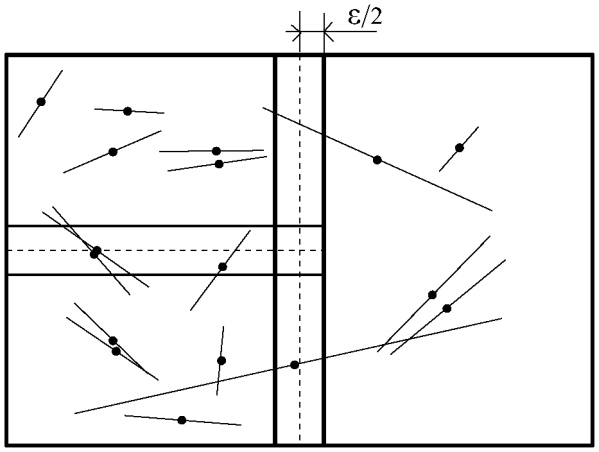

Fig. 9. Stitchable edge pairs fall inside the same box.

5.2.2 Tests for Determining Valid Stitches

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*. A stitch 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. 10, 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. There are three cases:

Case I. The stars of v_0 and v_1 do not intersect the stars of v'_0 and v'_1. This case is illustrated in Fig. 10a; the stitch is valid; it can be performed.

Case II. Either (v_0, v'_1) or (v_1, v'_0) is a current edge. Fig. 10b shows this configuration. The stitch cannot be performed since it creates a self-loop edge which is prohibited in our surface model.

Case 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 Fig. 10c, Fig. 10d, Fig. 10e, and Fig. 10f. Fig. 10c 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. 10d and Fig. 10e. However, the configuration of Fig. 10f 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 nonmanifold 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.

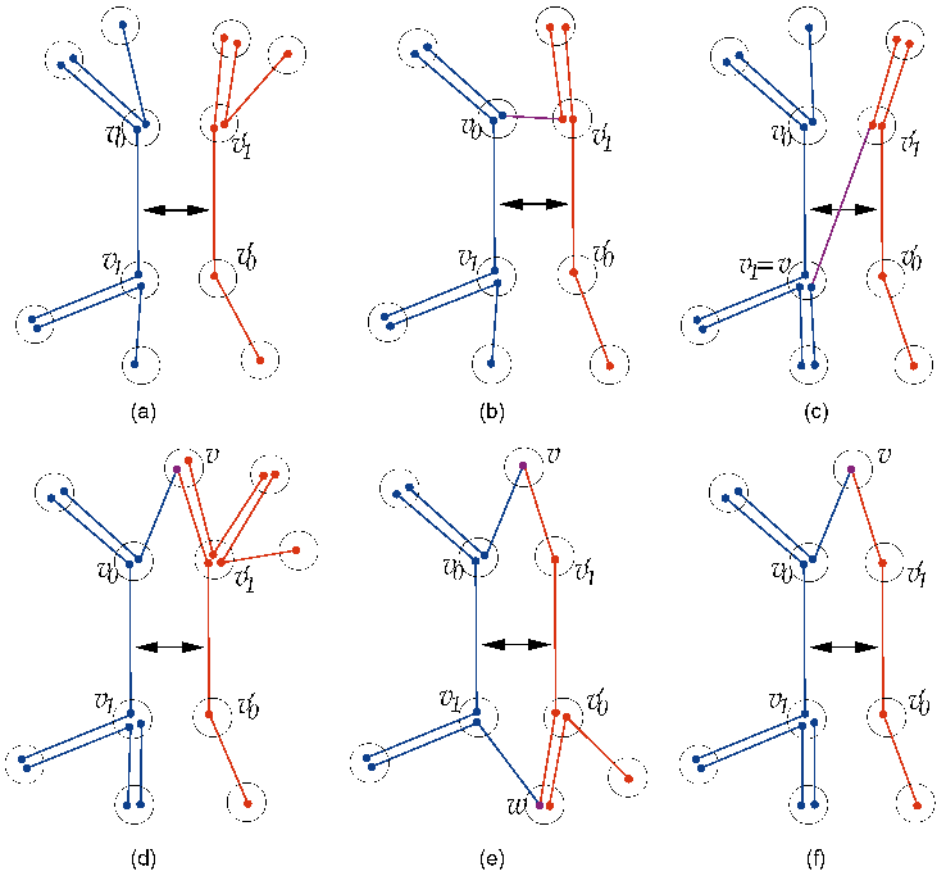

Fig. 10. Different configurations for a proposed stitch between (v_0, v_1) and (v'_1, v'_0).

This is the case for the configurations shown in Fig. 10c and Fig. 10e. A procedure called $\mathrm{CompareStars}(v, v')$ evaluates the effect of merging two corner groups, v and v' and classifies the merging as one of three types: (Type 1) creates at least one singular edge, (Type 2) creates no singular edge and creates no implicit stitch, or (Type 3) 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 $\mathrm{CompareStars}(v_0, v'_1)$ and perform the merging if there is an implicit stitch (Type 3).

Step II. We evaluate $\mathrm{CompareStars}(v_1, v'_0)$ and perform the merging if there is an implicit stitch (Type 3).

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

Step IV. If neither merge was performed and if both merges would not create any singular edges (Type 2), then perform both merges.

It is important to perform the first two steps sequentially. In the case of Fig. 10c, the above procedure will merge v_0 with v'_1 in Step I. However, $\mathrm{CompareStars}(v_1, v'_0)$ will prevent the second merging in Step II. Fig. 10d shows another case where order is important. $\mathrm{CompareStars}()$ works by maintaining a list of current edges incident to the corners of a group. $\mathrm{CompareStars}()$ verifies whether any edge is repeated in the 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.

5.2.3 Orientability

If we want to have an oriented surface, then we start by using the cutting methods of Section 4 to enforce the orientability of the input surface. Next, we consistently orient 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 provides 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.

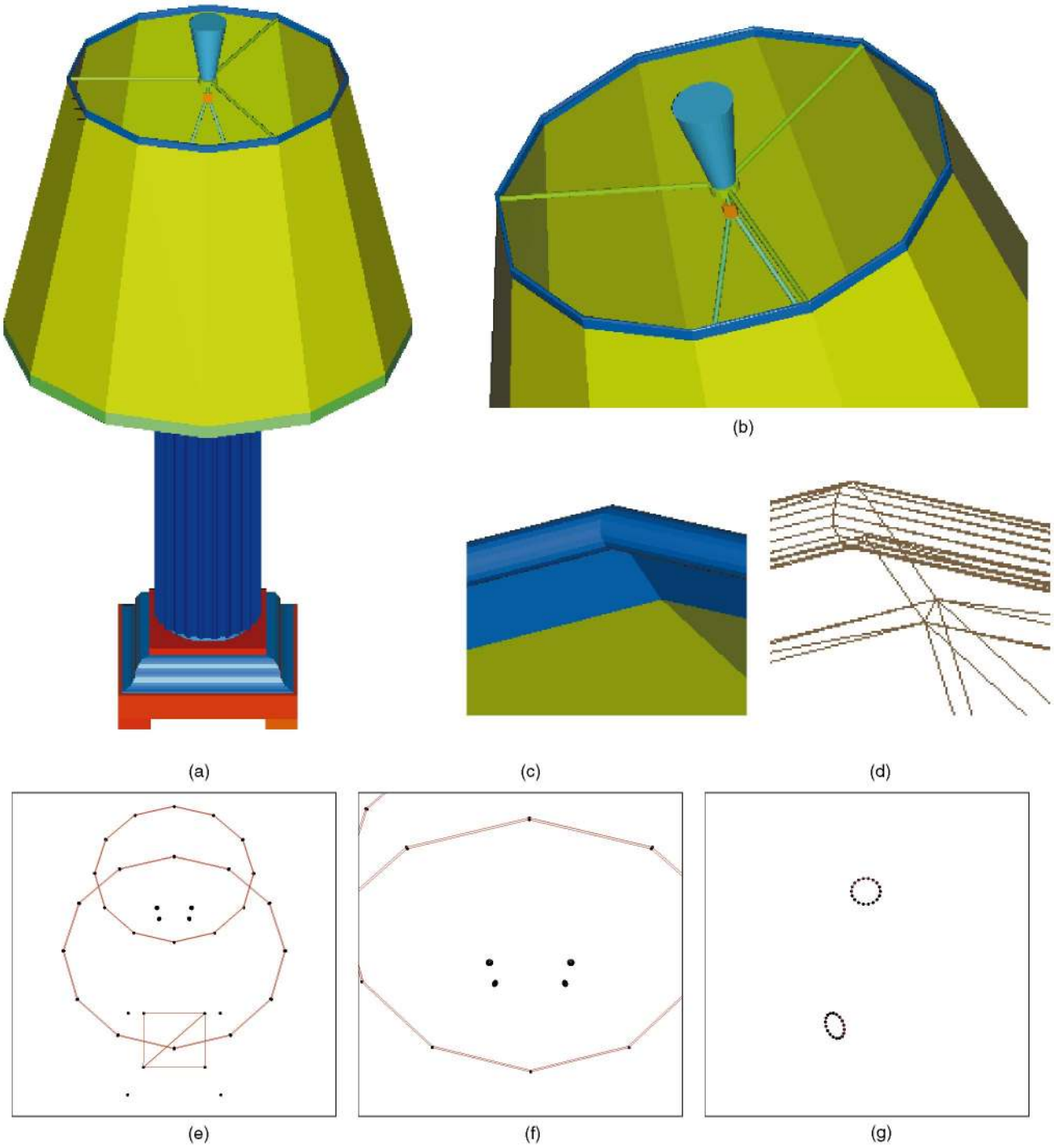

Fig. 11. Lamp model. (a) General view. (b), (c), (d) Successive details showing edges shared by more than two faces. (e), (f), (g) Singular edges are shown in red and singular vertices in black in increasingly detailed views.

6 Examples

6.1 Conversion of Nonmanifold Surfaces

We illustrate cutting and stitching with three practical examples. Since cutting and stitching are purely topological processes, their effects are not necessarily visible unless further processing occurs. The following visualization techniques are used to illustrate our methods: Different colors are used for boundary edges, regular edges, and singular edges; singular vertices are highlighted; geometrically adjoining boundary edges are disconnected (in some figures); and different colors are used for faces that belong to different connected components (in some figures).

The first example is a polygonal CAD model of a desk lamp and is shown in Fig. 11a. The original model contains 5,054 triangles and 2,810 vertices, including 125 singular edges and 128 singular vertices. After cutting through singular edges and vertices, 5,052 triangles and 3,058 vertices result. Fig. 11a shows the various connected components after conversion using different

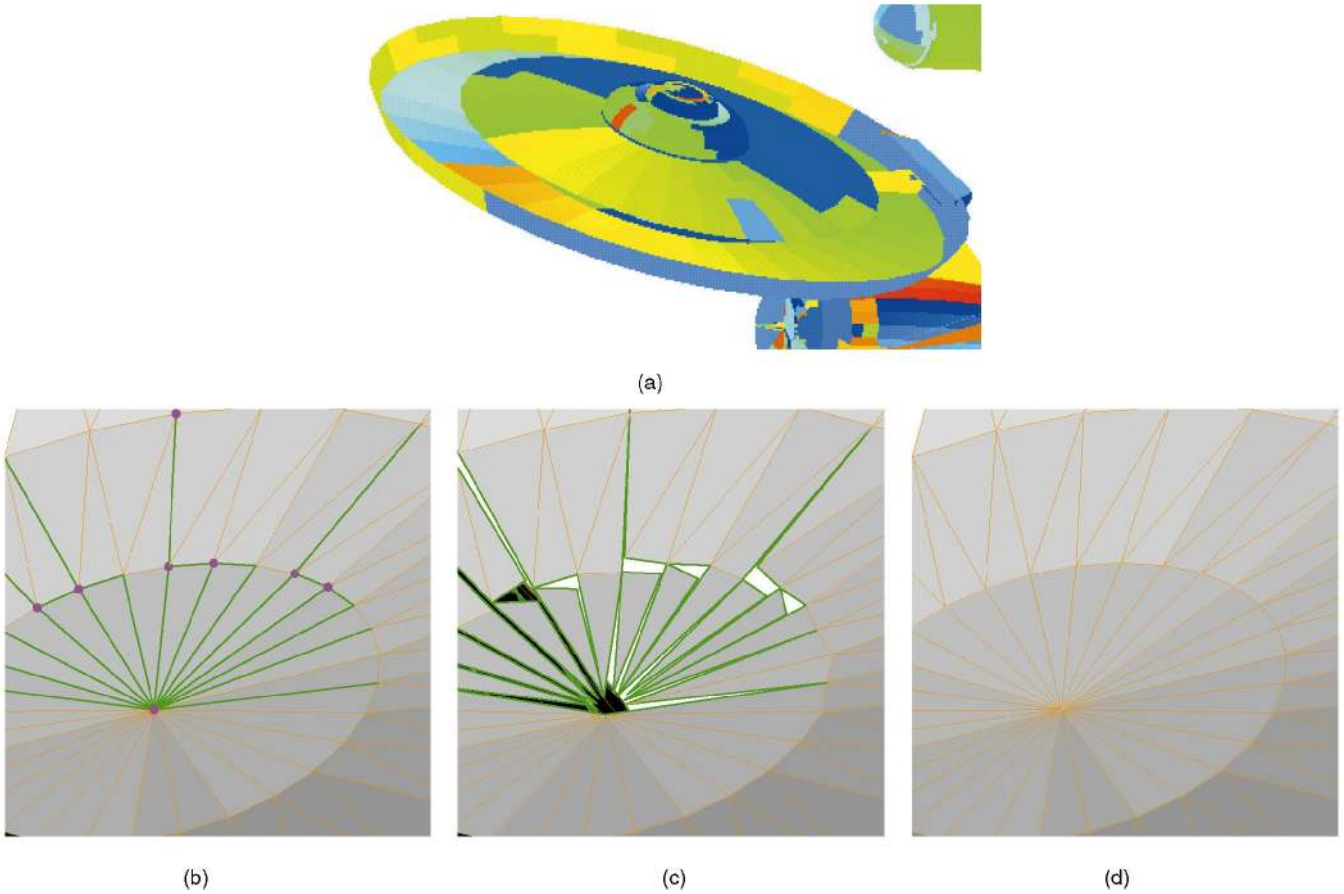

Fig. 12. Spaceship dish model. (a) Disconnected surface components are drawn using different colors. (b) Detail: Regular edges are shown in orange; boundary edges are shown in green; singular vertices are shown in magenta. No singular edge is visible in this area. (c) After cutting through singular edges and vertices. (Gaps are introduced here for illustrative purposes. Vertex coordinates are not affected.) (d) After stitching using the snapping strategy.

colors. The conversion took less than one second on an IBM RS6000 580.

The second example is a model representing a portion of a spaceship with 12,539 triangles and 15,011 vertices. Fig. 12a shows how the model is formed of a multitude of diconnected components. Disconnected surface components are drawn using different colors. The original model, after removing degenerate triangles, has 435 singular edges and 1,689 singular vertices. After cutting and stitching using the snapping strategy, we obtained 12,552 triangles and 7,429 vertices. The conversion took 21 seconds using an IBM Power PC 42T.

Asking a user to decide on how to locally connect polygons of a surface for thousands of different instances is not practical. This example thus makes a strong case for automating topology correction methods. Another benefit of the conversion process is to halve the number of vertices for this model, thus saving storage space and speeding up rendering (using triangle strips, for instance, or, more simply, because fewer geometric transformations are necessary).

The third example is a polygonal approximation of an isosurface extracted from a CT-scan of a fossil monkey jaw and is shown in Fig. 13a. The original model has 75,842 triangles and 37,624 vertices. The model contains 462 singular edges and 563 singular vertices; singular edges are shown in Fig. 13b and singular vertices in Fig. 13c. In this example, singularities originated from an incorrect algorithm. After cutting through singular edges and vertices, we obtained 75,371 triangles and 37,636 vertices. The conversion, including the removal of degenerate triangles, took 6 seconds on an IBM RS6000 580 workstation.

We next report statistics on the conversion of 332 VRML models in preparation for geometric compression using the method described in [4]. As this compression method incurs an overhead per connected component, we want to join disconnected surfaces and minimize the number of connected components; the snapping strategy for stitching can be very useful in this situation. The results are gathered in Table 1. We note that, on this data collected from the World Wide Web, the ratio between the number of connected components and the number of models can be very high (in excess of 1,000 to one); our methods can reduce this ratio considerably (less than five components per model on average) while building manifold surfaces and keeping surface properties (color, normal, texture coordinates) unchanged.

6.2 How Topology Affects Geometry: Surface Smoothing and Singularities

While cutting and stitching operate only on the topology, their effects can become clearly visible (and change results

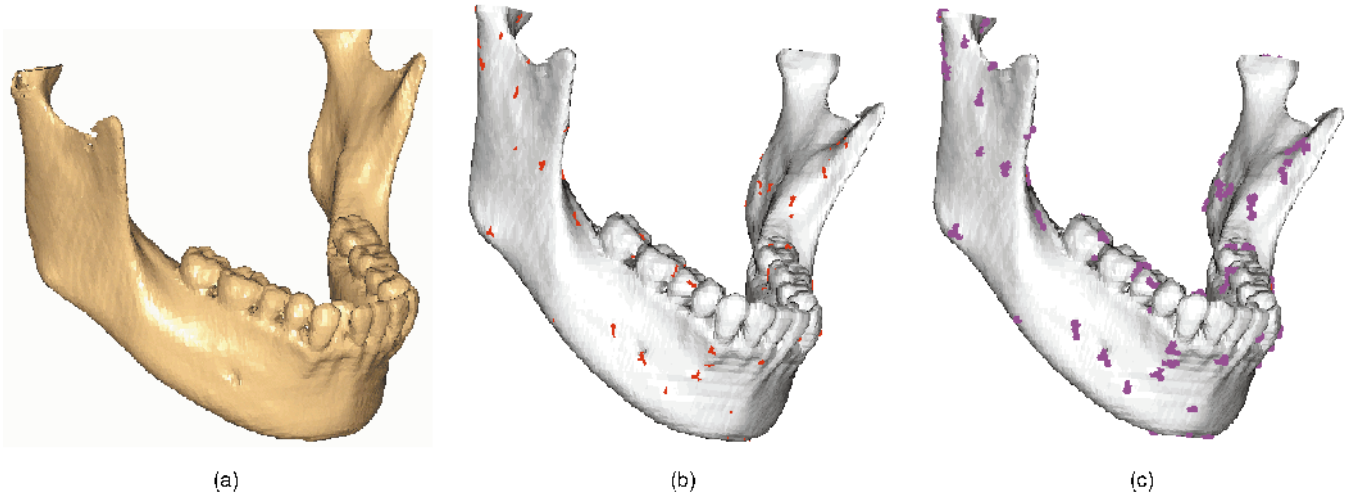

Fig. 13. Fossil monkey jaw (nonmanifold isosurface). (a) General view. (b) Singular edges are shown in red. (c) Singular vertices are shown in magenta.

dramatically) if the geometry of the surface is modified by a subsequent process.

In this section, we observe the combined effects of cutting and stitching with the surface smoothing algorithm of Taubin. Simply stated, a smoothing iteration in Taubin's algorithm involves two steps. Laplacian smoothing is used in a first step. To prevent shrinkage, in the second step, the surface is locally inflated by pushing each vertex in the opposite direction of the first (Laplacian) smoothing step. A detailed description of the algorithm can be found in [7]. Taubin's algorithm is designed for use on a manifold surface and will also handle a nonmanifold surface. However, for a nonmanifold surface, the resulting surface is not smooth in the vicinity of the singular edges and vertices.

A nonmanifold model consisting of two spheres sharing edges is shown in Fig. 14a. Using Taubin's algorithm, we attempt to subdivide and smooth the nonmanifold. The result is shown in Fig. 14b. This figure illustrates the nonsmooth behavior in the vicinity of the singular edges. In Fig. 14c, we first cut through the singular edges using the methods of Section 4 and then subdivide and smooth. In Fig. 14d, we smooth after cutting and stitching using the pinching strategy. In Fig. 14e, we smooth after cutting and stitching the snapping strategy. Fig. 14f illustrates a different outcome produced using the snapping strategy. For further reading on the topic of smoothing, Hubeli and Gross's paper [12] discusses smoothing algorithms specialized for nonmanifold meshes.

TABLE 1
Statistics on Conversion and Stitching Using the Snapping Strategy Applied to VRML 2.0 Models Collected on the World Wide Web

Source of VRML data / Statistics	www.microsoft.com/vrml	www.acuris.com	www.3dcafe.com
Models	200	23	109
Indexed Face Sets	665	128	421
Connected Components	2746	29461	4525
Vertices	28621	51015	59119
Singular Vertices	708	22960	4691
Edges	52610	31113	149690
Singular Edges	601	0	1976
Change in Components	-1106	-29033	-2709
Change in Vertices	-3656	-40078	-1663
Change in Edges	-2167	-10968	1191
Total CPU Time	41s	37s	1m51s

Timings were measured in minutes and seconds on an IBM RS6000 590 in debug mode; they include parsing of VRML files and scene graph operations.

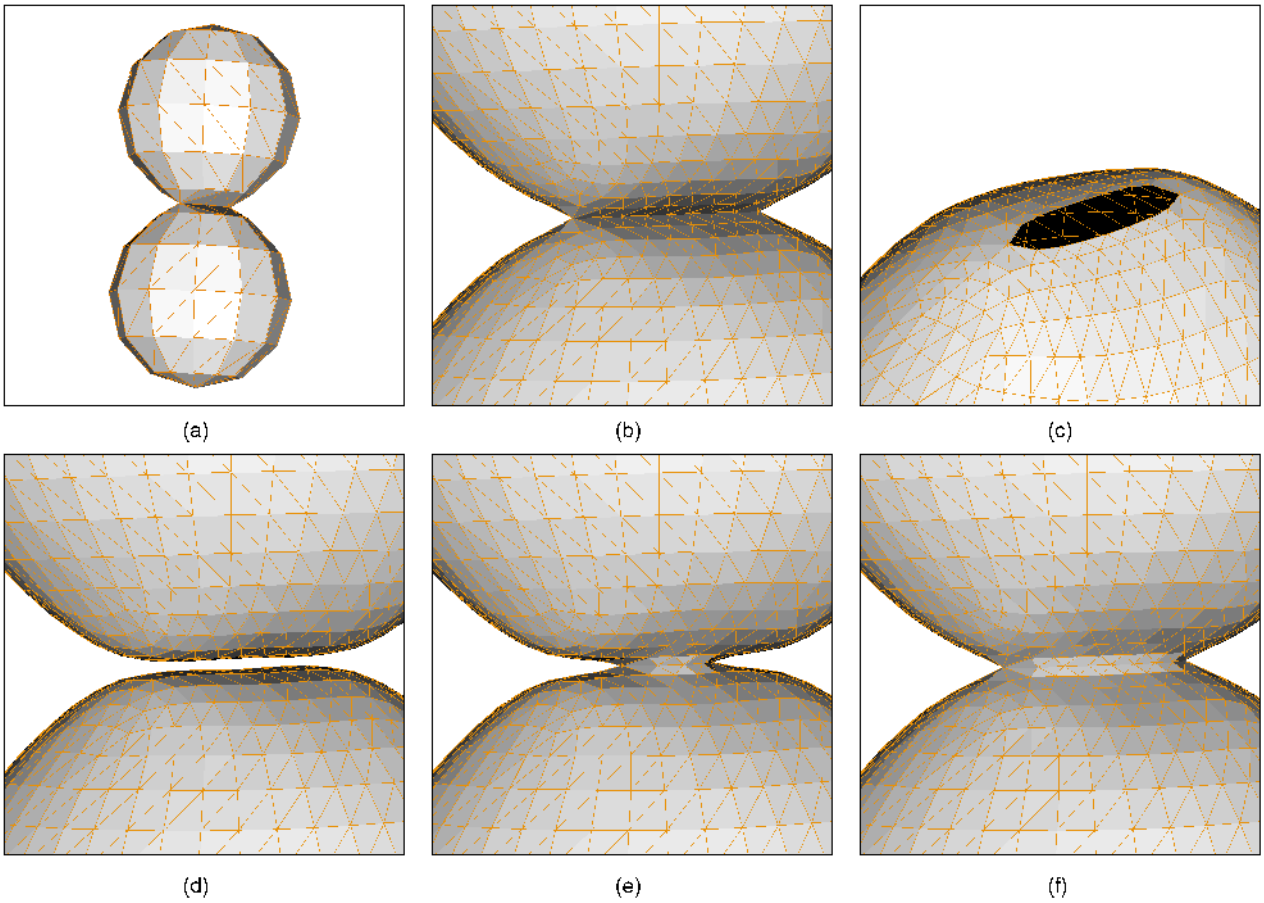

Fig. 14. Surface smoothing and singularities. (a) Nonmanifold model of two spheres sharing two edges. (b) After subdividing and smoothing. (c) After cutting, subdividing, and smoothing. (Only the bottom sphere is shown.) (d) After pinching boundary edges, subdividing, and smoothing. (e) After snapping boundary edges and smoothing. (f) Another result of edge snapping.

7 Related Work

Our method is different from most of the previous work as it operates solely on the surface topology. Following the publication of an excerpt of the present work [13], Rossignac and Cardoze [14] introduced the MatchMaker algorithm for minimizing the number of vertex replications when converting nonmanifold solids to manifold solid representations. MatchMaker analyzes the neighborhood of a singular vertex, and assigns (face, face) adjacencies through singular edges, thus "cutting less" than the present method. In MatchMaker, an edge cannot be uniquely identified with a pair of vertices: For instance, two edges (and four faces) can share the same two endpoints. Thus, the edge list must be provided as an output of the algorithm, while, in our case, the polygon-vertex incidences completely specify the edges.

Dey et al. [15] study the conditions for creating singularies when contracting edges of simplicial complexes: The conditions are related to what is described in Section 5. (In Dey's work, simplices are defined as convex hulls or points and, consequently, are not abstract. Our definitions are more general.) For 2-complexes (or surfaces), such conditions were also described in [16] and [17].

We group other related work into three categories.

The first category of related work covers methods that operate both on the geometry and topology to modify surfaces so that they can represent the boundary of solids [18]. This is an important issue for models represented using the rapid prototyping STL format [19]. This format consists of topologically disconnected triangles (see [20], [21], [22], [6], [23]). To build solids from STL representations, Bohn and Wozny [24], [25] use a topology-based approach complemented with heuristics for closing gaps.

Converting a nonmanifold surface to a solid is a difficult task involving floating point precision problems, computationally demanding tasks, and a number of open problems. An example of an open problem is the problem of filling a polygonal hole with a reasonable polygonal surface without creating intersections [26]. Recently, to address this open problem, Barequet proposed a method using "geometric hashing" to discover matching boundaries and close gaps [27]. (To learn more about "geometric hashing," the reader may consult other articles in the magazine where [27] was published.) In a related work, Butlin and Stops [28] repair CAD data for input to engineering analyses or to simplify data exchange. Barequet and Kumar [11] use methods similar to the global cutting method of Section 4.2 on STL files to stitch through regular edges. But, they can subsequently create a nonmanifold after stitching additional edges. Murali and Funkhouser [29] use polygon faces to partition the volume into cells and determine the likelihood for each cell to be solid. From a set of cells

considered solid, they produce a manifold boundary representation.

The second category covers methods that create and manipulate surface models. Szeliski et al. [30] build new polygonal surfaces from an existing surface by defining a collection of point samples and then using point repulsion methods to evenly distribute these points. A manifold surface triangulation of the distributed points is then computed. Welch and Witkin [31] build polygonal surfaces starting from simple surfaces by applying a series of surface operations that consist of adding, deleting, or morphing a portion of surface. They use mesh cutting techniques, but cut only along simple curves. Both methods build new lists of vertices and faces, while we manipulate an existing list of face vertex references. Veron and Leon [32] detect automatically singular vertices and edges, but they require user assistance to correct the singularities.

The last category of related work covers methods for building boundary representations of solids for use in Solid Modeling CAD. This category and the current work both address conversions between manifold and nonmanifold representations (e.g., [33]). Historically, much of the discussion on manifold and nonmanifold connectivities was performed in a CAD context. In solid modeling, nonmanifolds have the advantage of being able to represent a closed set over regularized Boolean intersection, complement, and union operations. Nonmanifold issues have been used to motivate the use of r-sets to represent solid objects [34]. An r-set is essentially what Hoffmann refers to as a nonmanifold solid [33]. r-sets can exhibit connections of zero width. Desaulnier and Stewart [35] have studied the relationship between manifold solids and r-sets. These nonmanifold representations allow more consistent implementations of Boolean operations [36]. Heisserman [37] developed a method for extracting a manifold boundary representation from a set of intersecting solids. Our methods differ from these works in that we we do not assume that our surfaces are boundaries of solids and we do not use the notions of interior or complement. This difference permits our methods to be applied to nonorientable manifolds such as Klein bottles or Moebius strips.

8 Conclusion

We have used cutting and stitching to automate the conversion of a set of polygons to a manifold polygonal surface. All the properties (colors, normals, texture coordinates) of the original surface can be passed on without degradation to the final surface. We have successfully used our methods to convert nonmanifold surfaces for subsequent processing by algorithms for surface simplification [3] and compression [4]. Some of these methods are also incorporated into the Open Visualization Data Explorer (OpenDX) open-source project [38], [39].

Our method may not be suitable if the original surface was intended to be a nonmanifold, that is, if topological singularities (singular edges and singular vertices) are an integral part of the model. Otherwise, the method is general and handles any type of topological singularity without user intervention. Finally, the method does not address geometric issues such as self-intersecting surfaces.

Stitching may be also used without cutting to join topologically disconnected but geometrically adjacent surface components. We have found this procedure useful for optimizing surfaces before geometric compression.

Appendix A
Data Structures and File Formats

Some data structures for surfaces, such as the Winged Edge proposed in the mid-1970s [40], can only represent a manifold surface; the Winged Edge also imposes this constraint on an associated file format. For instance, Kalvin uses this property to guarantee building a manifold isosurface [41].

Many 3D model singularities occur because some geometric models are stored and exchanged using an unstructured file format. Examples include the vertex and polygon lists in VRML [42] and polygon lists in the STL format. The future adoption of geometric compression in data exchange formats [4] may require topological consistency to achieve maximum compression.

Appendix B
Data Structures in Our Implementation

Our methods take as an input a surface represented by a list of n_v vertices and a list of n_f faces. Internally, the vertices and faces are 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 references for each face stored contiguously. We also use a "face start array" to provide the starting index of each face in the face array.

By looping through the face array, we build a structure of n_e edges while recording the number of incident faces. The edges are organized as a hash table indexed by the sorted pair of endpoint references (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, 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. The hash table permits, on average, constant time access to the edges. The edge data structure contains pointers to provide direct access to the incident faces.

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. We use the Union-Find algorithm for this purpose, whose running time is essentially $O(n)$, when n elements are in the partition [43]; once the partition is determined, access to representatives of faces or corners takes constant time.

Acknowledgments

The authors wish to thank Swami Narayanan, Vlad Pochop, and Manny Ko of Multigen-Paradigm for their feedback. Many thanks to the TVCG reviewers for their suggestions and careful reviews.

REFERENCES

[1] S. Doo and M. Sabin, "Analysis of the Behaviour of Recursive Division Surfaces Near Extraordinary Points," *Computer Aided Design,* vol. 10, no. 6, pp. 356-360, 1978.

[2] A. Varshney, "Hierarchical Geometric Approximations," PhD thesis, Univ. of North Carolina at Chapel Hill, 1994.

[3] A. Guéziec, "Locally Toleranced Polygonal Surface Simplification," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 2, pp. 178-199, Apr.-June 1999.

[4] G. Taubin, W.P. Horn, F. Lazarus, and J. Rossignac, "Geometry Coding and VRML," *Proc. IEEE,* vol. 86, no. 6, pp. 1228-1243, June 1998.

[5] H. Hoppe, "Efficient Implementation of Progressive Meshes," Technical Report MSR-TR-98-02, Microsoft Research, Redmond, Wash., Jan. 1998.

[6] J.H. Bohn, "Removing Zero-Volume Parts from CAD Models for Layered Manufacturing," *IEEE Computer Graphics and Applications,* vol. 15, no. 6, pp. 27-34, Nov. 1995.

[7] G. Taubin, "A Signal Processing Approach to Fair Surface Design," *Proc. Siggraph,* pp. 351-358, Aug. 1995.

[8] F. Harary, *Graph Theory.* Addison-Wesley, 1969.

[9] M.K. Agoston, *Algebraic Topology: A First Course.* New York: Marcel Dekker, 1976.

[10] X. Sheng and I.R. Meier, "Generating Topological Structures for Surface Models," *IEEE Computer Graphics and Applications,* vol. 15, no. 6, pp. 35-41, Nov. 1995.

[11] G. Barequet and S. Kumar, "Repairing CAD Models," *Proc. Visualization '97,* pp. 363-370, Oct. 1997.

[12] A. Hubeli and M. Gross, "Fairing of Non-Manifolds for Visualization," *Proc. Visualization 2000,* pp. 407-414, Oct. 2000.

[13] A. Guéziec, G. Taubin, F. Lazarus, and W.P. Horn, "Converting Sets of Polygons to Manifold Surfaces by Cutting and Stitching," *Proc. Visualization '98,* pp. 383-390, Oct. 1998.

[14] J. Rossignac and D. Cardoze, "Matchmaker: Manifold Breps for Non-Manifold r-Sets," *Proc. Fifth Symp. Solid Modeling and Applications (SMA '99),* pp. 31-41, June 1999.

[15] T.K. Dey, H. Edelsbrunner, S. Guha, and D. Nekhayev, "Topology Preserving Edge Contraction," *Publications de l'Institut Mathematique (Beograd),* vol. 60, no. 80, 1999, also, Technical Report RGI-Tech-98-018, Raindrop Geomagic Inc. , Research Triangle Park, N.C., 1998.

[16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," *Proc. Siggraph,* pp. 19-25, July 1993.

[17] A. Guéziec, "Surface Simplification with Variable Tolerance," *Proc. Second Ann. Int'l Symp. Medical Robotics and Computer Assisted Surgery,* pp. 132-139, Nov. 1995.

[18] M. Segal and C.H. Sequin, "Partitioning Polyhedral Objects into Non-Intersecting Parts," *IEEE Computer Graphics and Applications,* vol. 8, no. 1, pp. 53-67, Jan. 1988.

[19] SLA CAD, *Stereolithography Interface Specification,* 3D Systems Inc., Valencia, Calif., 1989.

[20] S.J. Rock and M.J. Wozny, "Generating Topological Information from a 'Bucket of Facets,'" *Solid Freeform Fabrication Symp. Proc.,* H.L. Marcus, ed., pp. 251-259, Aug. 1992.

[21] I. Makela and A. Dolenc, "Some Efficient Procedures for Correcting Triangulated Models," *Proc. Symp. Solid Freeform Fabrication,* pp. 126-134, July 1993.

[22] M.C. Bailey, "Tele-Manufacturing: Rapid Prototyping on the Internet," *IEEE Computer Graphics and Applications,* vol. 15, no. 6, pp. 20-26, Nov. 1995.

[23] V. Chandru, S. Manohar, and C.E. Prakash, "Voxel-Based Modeling for Layered Manufacturing," *IEEE Computer Graphics and Applications,* vol. 15, no. 6, pp. 42-47, Nov. 1995.

[24] J.H. Bohn and M.J. Wozny, "A Topology Based Approach for Shell Closure," *Geometric Modeling for Product Realization,* P.R. Wilson et al., eds., pp. 297-319, 1993.

[25] J.H. Bohn, "Automatic CAD-Model Repair," PhD thesis, Rensselaer Polytechnic Inst., Troy, N.Y., Aug. 1993.

[26] G. Barequet and M. Sharir, "Filling Gaps in the Boundary of a Polyhedron," *Computer Aided Geometric Design,* vol. 12, no. 2, pp. 207-229, 1995.

[27] G. Barequet, "Using Geometric Hashing to Repair CAD Objects," *IEEE Computational Science and Eng.,* vol. 4, no. 4, pp. 22-28, Oct. 1997.

[28] G. Butlin and C. Stops, "CAD Data Repair," *Proc. Fifth Int'l Meshing Roundtable,* pp. 7-12, 1996.

[29] T.M. Murali and T.A. Funkhouser, "Consistent Solid and Boundary Representations from Arbitrary Polygonal Data," *Proc. Symp. Interactive 3D Graphics,* pp. 155-161, Apr. 1997.

[30] R. Szeliski, D. Tonnesen, and D. Terzopoulos, "Curvature and Continuity Control in Particle-Based Surface Models," *Geometric Methods in Computer Vision II,* vol. 2031-15, pp. 172-181, July 1993.

[31] W. Welch and A. Witkin, "Free-Form Shape Design Using Triangulated Surfaces," *Siggraph '94 Conf. Proc.,* pp. 247-256, July 1994.

[32] P. Veron and J.C. Leon, "Static Polyhedron Simplification Using Error Measurements," *Computer Aided Design,* vol. 29, no. 4, pp. 287-298, Apr. 1997.

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

[34] A.A.G. Requicha, "Representations for Rigid Solids: Theory, Methods and Systems," *ACM Computing Surveys,* vol. 12, no. 4, pp. 437-464, Dec. 1980.

[35] H. Desaulniers and N.F. Stewart, "An Extension of Manifold Boundary Representations to the r-Sets," *ACM Trans. Graphics,* vol. 11, no. 1, pp. 40-60, Jan. 1992.

[36] C.M. Hoffmann, J.E. Hopcroft, and M.S. Karasick, "Robust Set Operations on Polyhedral Solids," *IEEE Computer Graphics and Applications,* vol. 9, no. 6, pp. 50-59, Nov. 1989.

[37] J.A. Heisserman, "Generative Geometric Design and Boundary Solid Grammars," PhD thesis, Carnegie Mellon Univ., May 1991.

[38] G. Abram and L. Treinish, "An Extended Data-Flow Architecture for Data Analysis and Visualization," *Proc. Visualization '95,* pp. 263-270, Oct. 1995.

[39] *Open Visualization Data Explorer,* http://www.research.ibm.com/dx, 1999.

[40] B.G. Baumgart, "Geometric Modeling for Computer Vision," PhD thesis, Stanford Univ., 1974.

[41] A. Kalvin, "Segmentation and Surface-Based Modeling of Objects in Three-Dimensional Biomedical Images," PhD thesis, New York Univ., June 1991.

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

[43] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* McGraw-Hill, 1989.

André Guéziec holds a PhD degree in computer science from the University Paris 11 Orsay that was obtained with honors in 1993. He graduated from the Ecole Centrale Paris in 1989. He is a senior software engineer involved in data visualization, 3D graphics, and video processing, currently working at Sportvision and, recently, at MultiGen-Paradigm, where he worked on software architecture for information visualization and visual simulation. In 1992-1994, he was a postdoctoral fellow and adjunct faculty member at the Courant Institute of Mathematical Sciences of New York University, with a joint position at the NYU Medical Center. From 1994 to 1999, he was a research staff member at the IBM T.J Watson Research Center, where he worked on medical imaging and robotics, on geometric modeling, and on the mpeg-4 standard for 3D model coding. He holds seven patents and has published about 30 papers. He was the Keynote Speaker at the 1996 Leeds Statistics Symposium. He is a senior member of the IEEE. His research interests include several aspects of 3D graphics, computer vision, and medical imaging, with an emphasis on 3D shape modeling.

Gabriel Taubin earned a PhD degree in electrical engineering from Brown University and a Licenciado en Ciencias Matematicas degree from the University of Buenos Aires, Argentina. He is a visiting professor of electrical engineering at the California Institute of Technology and a research staff member at the IBM T.J. Watson Research Center, where, most recently, he was manager of the Visual and Geometric Computing Group. His main research interests fall into the following disciplines: applied computational geometry, computer graphics, geometric modeling, and computer vision For the last few years, his main line of research has been the development of efficient, simple, and mathematically sound algorithms to operate on 3D objects represented as polygonal meshes, with an emphasis on technologies to enable the use of 3D models for Web-based applications. He made significant contributions to 3D capturing and surface reconstruction, modeling, compression, progressive transmission, and display of polygonal meshes. The 3D geometry compression technology developed jointly with his group is now part of the MPEG-4 standard and part of the IBM HotMedia product. He is also known for the development of geometric signal processing on polygonal meshes. He is now also interested in the development of applications for smart assistants (gadgets) with sensory systems. He is a member of the IEEE.

Francis Lazarus received his doctorate from the University of Paris VII in 1995 with a thesis on morphing algorithms. He is a senior researcher at the CNRS (Centre National de la Recherche Scientifique) in Poitiers, France. He was a postdoctoral researcher at the IBM T.J. Watson Research Center from 1996 to 1997. His research interests include geometry compression, geometric modelling, computer animation, and 3D morphing.

Bill Horn holds a PhD degree from Cornell University and has worked on a variety of projects in mechanical computer aided design and 3D graphics. He currently manages the Advanced Visualization Systems Group at the IBM T.J. Watson Research Center.