Converting Sets of Polygons to Manifold Surfaces by Cutting and Stitching

André Guéziec [1] Gabriel Taubin [1] Francis Lazarus [2] William Horn [1]

IBM T. J. Watson Research Center

Abstract

Many real-world polygonal surfaces contain topological singularies that represent a challenge for processes such as simplification, compression, smoothing, etc. We present an algorithm for removing such singularities, thus converting non-manifold sets of polygons to manifold polygonal surfaces (orientable if necessary).

We identify singular vertices and edges, *multiply* singular vertices, and *cut* through singular edges. In an optional *stitching* phase, we join surface boundary edges that were cut, or whose endpoints are sufficiently close, while guaranteeing that the surface is a manifold. We study two different stitching strategies called "edge pinching" and "edge snapping"; when snapping, special care is required to avoid re-creating singularities.

The algorithm manipulates the polygon vertex indices (surface *topology*) and essentially ignores vertex coordinates (surface *geometry*). Except for the optional stitching, the algorithm has a linear complexity in the number of vertices edges and faces, and require no floating point operation.

Key-words : Polygonal Surface, 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. Many such polygonal surfaces contain topological singularities, (e.g., edges shared by more than two triangles, several triangle fans incident to a single vertex) that represent a challenge for various algorithms that operate exclusively on a *manifold* surface. A manifold polygonal surface is such that the neighborhood of every vertex can be continuously deformed to a disk (to a half disk at the boundary: see Section 2). In fact, this corresponds to an intuitive definition of what a "surface" is, as opposed to an arbitrary collection of polygons.

In this paper, we essentially ignore the coordinates associated with the surface elements, and we look at the property of being a manifold as a purely topological one. Topological degeneracies can occur by design choice (e.g., vertex merging to avoid duplicating coordinates, or polygon reduction tools), or they can be produced by incorrect algorithms for building surfaces (e.g., iso-surfaces, triangulation of scattered points), or by correct algorithms containing software bugs, etc.

[1] IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, {taubin,gueziec,hornwp}@watson.ibm.com

[2] IRCOM-SIC (UMR CNRS 6615), SP2MI, Bvd. 3, Teleport 2, B.P. 179, 86960 Futuroscope Cedex, France, lazarus@sic.univ-poitiers.fr

Some concrete examples of algorithms that fail on input containing topological singularities are: algorithms for surface subdivision [1]; algorithms that simplify surfaces ([2, 3]); algorithms for surface compression[4]; algorithms for progressive transmission (Hoppe [5] relies on each triangle having no more than three neighbors); algorithms that (scan) convert a polygonal boundary representation of a potential solid for Rapid Prototyping [6]. Other algorithms yield undesired results when executed on non-manifold input, such as surface smoothing [7] (see Section 6).

Several approaches are possible:(1) modifying algorithms to handle non-manifold input; (2) trying to understand the source of errors in modeling or CAD packages, and lobbying (and hoping) for such errors to be corrected; (3) developing methods to correct the input. Following the first approach is application-dependent, and probably requires re-defining objectives (beyond just accepting non-manifold input). For instance, a number of surface simplification methods accept non-manifolds, but often they introduce many more degeneracies than those originally present. The second approach has little short term impact and may not be a complete solution in the long term as well (there will always be software bugs).

We have chosen the third approach. In this paper, we provide the complete description of a novel and efficient method for automatically converting a non-manifold surface to a 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. We assume that the topology of the surface is already built for the most part, and concentrate on removing the singularities. However, we have developed a stitching method (edge snapping) to help build the topology. In general, we assume that the corrections will involve a relatively small number of surface elements.

The algorithm is decomposed into two main parts: *cutting* and *stitching*. Cutting is a general method for disconnecting the surface topology along a set of marked edges or vertices. We mark *singular* edges and vertices; an edge is singular if more that two faces are incident to it; a singular vertex is defined in Section 2. 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 at once, by first breaking all connections between faces and later joining adjacent faces that share a unmarked edge. The local method operates only on marked vertices and endpoints of marked edges, by counting the number of (unmarked) 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. The global method is more appropriate when there is a large number of topological singularities to correct. The local method is more efficient when there are only few singular elements in a generally correct topology.

As illustrated in Fig. 1, several manifolds can be mapped to the original non-manifold by identifying vertices. To reduce the number of vertices, holes, or components, the cutting operation is followed by stitching. As defined in Section 5, stitching consists of

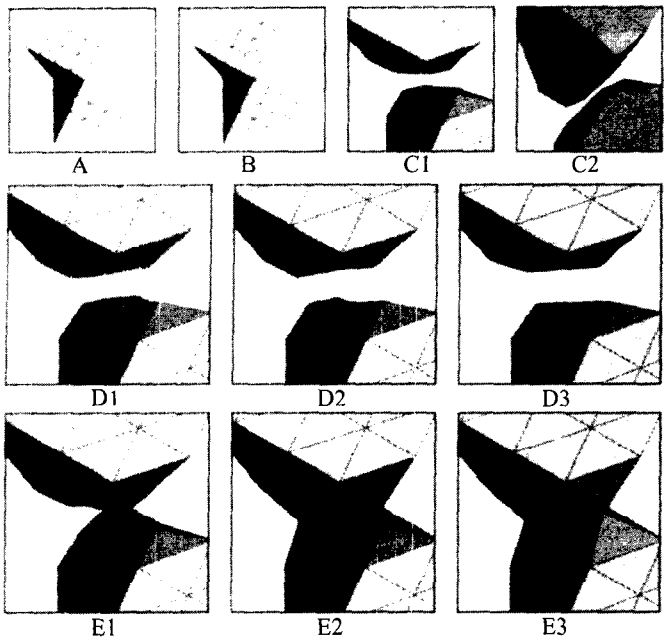

Figure 1: Converting to a manifold surface: A,B,C: cutting through singular edges; For illustrative purposes, topologically disconnected vertices are shown apart. We implement two stitching strategies: "pinching" edges along the same boundary (D) or "snapping" together edges belonging to different boundaries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

taking two boundary edges and identifying them, while guaranteeing that the surface is a manifold. We have observed that stitching is a delicate operation: for instance, when using the "zipping" method reported in [8], singular edges can be created when stitched multiple times. We show examples in Section 6 and provide sample timings.

Fig. 1 is a diagram illustrating our method. We consider two tetrahedra sharing an edge: we subdivide the surface of the tetrahedra into smaller triangles, resulting in the surface of Fig. 1A. We label singular edges and vertices and color them in red in Fig. 1B (regular edges are orange). After multiplying singular vertices, we have created two disconnected surface components in Fig. 1C, each of which has a boundary of length eight (in green); as explained in Section 4, 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 (or "pinching"), we have created in Fig. 1D two disconnected solids. Instead, when stiching along different boundaries (or "snapping"), we create Fig. 1E a single surface without boundaries. All three surfaces C, D, and E are manifolds with the same geometrical realization as A.

2 Polygonal Surfaces

For our purposes, a (polygonal) *surface* $S(\{\mathbf{v_i}\}, \{f_j\})$ 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. The face is said to be *incident* on such vertices. A pair (vertex, incident face) is called a *corner*. 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 for the vertices of a face modulo

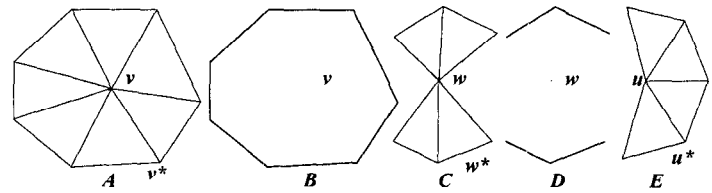

Figure 2: A: the star $\mathbf{v}^\star$ of a regular vertex $\mathbf{v}$ of valence seven. B: the link of $\mathbf{v}$. C: the star $\mathbf{w}^\star$ of a singular vertex $\mathbf{w}$. D: the link of $\mathbf{w}$, composed of two disconnected polygonal curves. E: the star $\mathbf{u}^\star$ of a boundary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

circular permutation, resulting in two orientations for that face. In this paper, we call *topology* of a surface, the set of ordered subsets of indices provided by the set of faces $\{f_j\}$, modulo circular permutation. We use the word *geometry* to mean the set of vertex coordinates $\{\mathbf{v_i}\}$. There are no particular constraints on the geometry for our methods to apply: polygons can be warped. Cutting and stitching operate on the topology only. Additionally, vertices or faces may have a number of continuous or discrete properties, such as colors, normals, and texture coordinates. Properties can also be associated with corners. The data structures we use are described in the Appendix.

We call the subset of faces of $\{f_j\}$ that share a vertex $\mathbf{v}$ the *star* of $\mathbf{v}$, noted $\mathbf{v}^\star$. The number of faces in $\mathbf{v}^\star$ is called the *valence* of the vertex $\mathbf{v}$. To form the *link* of a vertex, 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*. We call an edge incident on one single face a *boundary edge*. A regular vertex incident to a boundary edge is called a *boundary vertex*. These cases are illustrated in Fig. 2. A surface is a *manifold* if each vertex is a regular vertex; otherwise it is a *non-manifold* surface.

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.

3 Conversion of Non-Manifold Surfaces

To remove topological singularities, our method begins by cutting, or disconnecting, the surface along singular edges and at singular vertices. After this operation, which is described in more detail in Section 4, by construction each vertex has a star formed of a single component of faces connected through regular edges, meaning that the surface is a manifold according to our definition. A number of pre-processing steps may be necessary in the presence of invalid faces, and to accomodate our two different methods of cutting (locally or globally). Also, as mentioned above, stitching may be useful to reduce the number of duplicated vertices, or to build the topology from an input consisting of disconnected polygons. The details of the conversion steps are as follows:

Step I: Processing Invalid Faces 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. 3: faces can be invalid because vertices are duplicated (Fig. 3A). In this case, the global cutting method automatically produces the polygon marked in green in Fig. 3A. Another possibility for an invalid face is to be incident on the same edge multiple times (Fig. 3B). 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

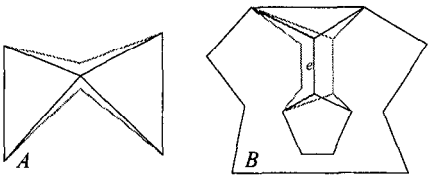

Figure 3: Invalid faces. A: duplicate vertex; the global method automatically produces the correction shown in green. B: face incident on the same edge e multiple times.

not the case, we mark an incident edge; marked edges are shown in red in Fig. 3B; the global method produces the solution marked in green in Fig. 3B.

These methods do not apply when using the local method: in practice in this case we have simply eliminated invalid faces.

Step II Singular edges are marked when building the edge data structure, as the edges that are shared by at least three different faces.

Step III: Pre-Processing for the Local Method We label standalone vertices when looping through the face list by counting the number of face incidences (valence) for each vertex. If the number of faces is less than the valence, then at least one additional connected component exists, so the vertex is an isolated singular 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.

Isolated singular vertices must be marked. 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^* by pivoting about v, starting with any "first" face f incident on v. 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 we encounter a boundary edge or the first face f. We then count the number of faces that were visited and compare this number to the valence of v. Note that this method also works if the faces are not consistently oriented.

Step IV We apply either the global cutting method on the marked edges or the local cutting method on the marked edges and vertices.

Step V (Optional): Building an Oriented Manifold We must verify that faces incident on each edge are consistently oriented. After cutting through singular edges, we propagate the orientation of faces of the resulting manifold using a spanning tree of faces. The number of required operations is proportional to the total number of faces. After propagating the orientation, some edges may have inconsistently oriented incident faces. In a second step, we mark these edges and cut along them (fewer edges are cut in this way as opposed to marking and cutting edges before propagating face orientations).

Step VI (Optional) We perform some stitching as described in Section 5.

4 Cutting

As illustrated in Figs. 4 and 6 cutting consists of disconnecting the surface along a collection of marked edges or vertices: multiple copies of vertices are created and assigned new indices; vertex indices in faces are modified to refer to the proper copy [1]. We describe a "local" and a "global" methods for cutting; we mark singular edges and vertices and cut through them to obtain a manifold.

4.1 Local Method for Cutting

We call this method *local* because it only operates on selected vertices and faces. Starting with a list of marked edges, we mark vertices that are endpoints of marked edges. Additional vertices may

[1] A rigourous definition of cutting (and stitching) can be found in Agoston [9]

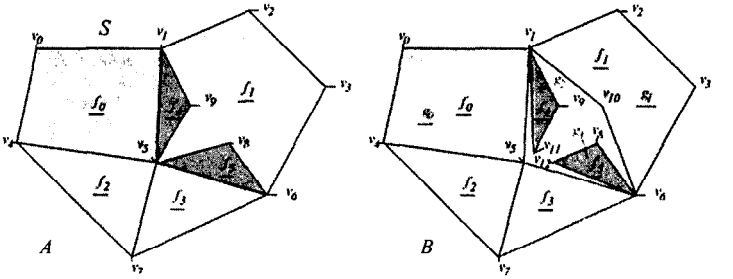

Figure 4: Local cutting. A: star of Vertex v_5 with marked edges in bold. B: multiplying Vertex v_5.

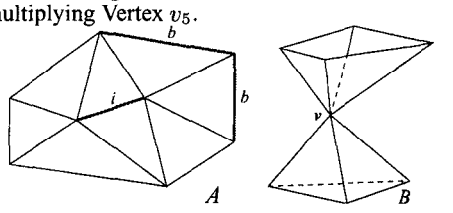

Figure 5: A: It is not possible to cut through any collection of marked edges: we need at least two adjacent edges if none of them is incident to the boundary. B: an isolated singular vertex **v**.

be marked as described below. We visit each marked vertex in turn; for each marked vertex v, we determine its star v^*, which can be obtained for each vertex by looping on the face array. We decompose v^* in subsets that are connected by unmarked edges. This can be performed using standard methods: we collect all unmarked edges incident on v, and maintain a partition on the faces of v^*, considering two faces adjacent if and only if they share a unmarked edge. Here we assume that all faces are valid. Invalid faces are treated in Section 3.

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 properties. Each instance of v is labeled from 0 to $n_c - 1$. In v^*, 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 Fig. 4. In Fig. 4A, marked edges are drawn bold. The unmarked edges incident on v_5 are (v_4, v_5) and (v_5, v_7). Four connected components of faces $g_0 \ldots g_3$ are identified in Fig. 4B. 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. The cut is completed once all marked vertices have been multiplied. Any two surface portions that share a collection of marked edges and no other edge or vertex will be disconnected by the cut. It is not possible to cut through any collection of edges: see Fig. 5A, we need at least two adjacent edges if none of the edges is incident to the boundary.

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 **v**, 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 proportional to the number of marked vertices multiplied by the largest valence of a vertex.

4.2 Global Method for Cutting

The global method for cutting requires the specification of a set of marked edges. We call it a global method because it operates on all

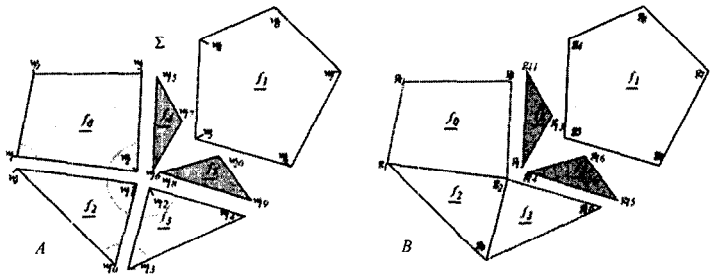

Figure 6: Global cutting. A: corner groups are shown using circular arcs. B: result of global cutting.

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. 6A, we show the corners $w_0 \ldots w_{20}$ obtained by completely disconnecting v_5^*.

We then define a partition of the face corners as follows: we visit the unmarked edges in S one after the other. For each unmarked 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 unmarked 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^* in Fig. 6B, where the corner groups $g_0 \ldots g_{16}$ are shown. The example configuration that we use is the same as in Fig. 4; however, as the method is global, all vertices in the configuration are affected, and not only v_5. The worst case complexity of the method is linear in the total number of corners (to disconnect the surface entirely) and in the number of unmarked edges.

Comparison of the two methods The labeling of vertices after the cut is different in both methods, but this does not affect the surface topology. Unlike the local method, the global method implicitly cuts through *isolated singular vertices*, which are singular vertices that are not endpoints of singular edges (see Fig. 5B). The global method eliminates "standalone" vertices, which are vertices without incident faces.

There are cases when one method will be preferred against the other: when the cut covers a large portion of the surface, the global method has a lower cost. The global method is more effective in the presence of invalid faces (as discussed below), standalone or isolated singular vertices. Alternatively, 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 is less costly because it visits only marked edges and vertices.

5 Stitching

Stitching corresponds to "identifying" boundary edges. The basic stitching operation is called an *edge stitch*. As explained in Section 4.2, 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 process. After each stitch, a new surface can be defined by identifying each corner group with a new vertex.

When applying our conversion algorithm to the surface in Fig 7A, we would obtain the surface in B after cutting using either method of Section 4, and unfortunate stitching choices (labeled 1, 2, 3 and 4) would create in C a non-manifold surface very similar to A (except that the red as well and green edges are not identified). We address this problem specifically in Sections 5.1 and 5.2. An edge stitch is called *valid* if it creates no singularity. Since the vertex identifications are induced by edge stitching, the link of every vertex cannot be disconnected. Accordingly, no isolated singular vertex can appear; we must only test for the creation of singular edges.

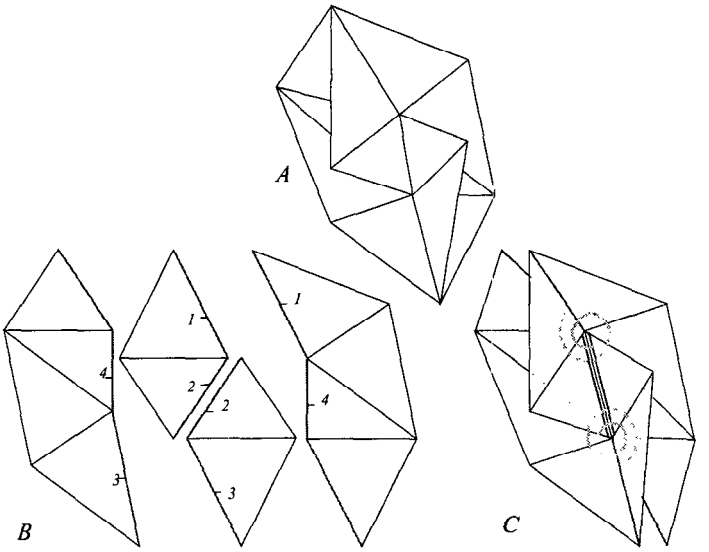

Figure 7: A: a non-manifold surface; after cutting through singular edges, we obtain the surfaces of B. B: an incompatible sequence of edge stitches (labeled 1,2 3 and 4), resulting in a non-manifold in C. C: spirals indicate which corners are identified (grouped) after stitching.

We propose two different greedy strategies for stitching. A Pinching Strategy consists of keeping the components that were created after cutting, and of stitching some of the boundary edges created when cutting. We prove that it is impossible to create a non-manifold using the Pinching Strategy. A Snapping Strategy consists of stitching along different boundaries, whereby a test needs to be developed to avoid creating singular edges, as illustrated in Figs. 7 and 10.

5.1 Pinching Strategy: Pinching Adjacent Boundary Edges Created by Cutting

Edges are determined to be "stitchable" if we did cut through such edges in the previous stage. We compute connected components of adjacent boundary edges (i.e, boundaries). For each boundary, we choose a pair of adjacent stitchable edges and pinch them along their common boundary vertex: 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 we always obtain a manifold, assuming that all singular edges and vertices were cut before. Firstly, as we stitch adjacent edges, we only identify one pair of vertices v_1 and v_2. Secondly, let us suppose that when trying to identify v_1 and v_2 we 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, 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 edge was stitched to it after cutting. Using the Pinching Strategy, when stitching an edge,

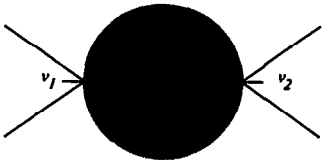

Figure 8: This configuration cannot be obtained by following Strategy I of pinching adjacent boundary edges.

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: this configuration is impossible using the Pinching Strategy (see Fig. 8).

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.

Having many components may be useful for the following situations: in some cases (e.g., the iso-surface of Fig. 13), most components except a few can be rejected because they correspond to noise or have no impact on a visualization. Also, some algorithms perform better with many components, e.g., the surface simplification of Rossignac and Borrel [10].

5.2 Snapping Strategy : Stitching Edges Belonging to Different Boundaries

It may be useful to allow stitching between edges that were not previously identified, but which are geometrically close to one another: for instance if the surface is specified by a set of disconnected faces and if the coordinates of vertices of such faces contain small, unintended discrepancies. Also, to provide suitable input for methods such as Taubin and Rossignac's [4], that have an overhead cost for every connected component, we wish to join disconnected surfaces and minimize the number of connected components. 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 332 VRML models before geometric compression: we report statistics pertinent to this application in Table 1.

Strategy We start 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 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

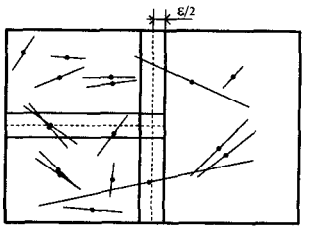

Figure 9: Stitchable edge pairs fall inside the same box.

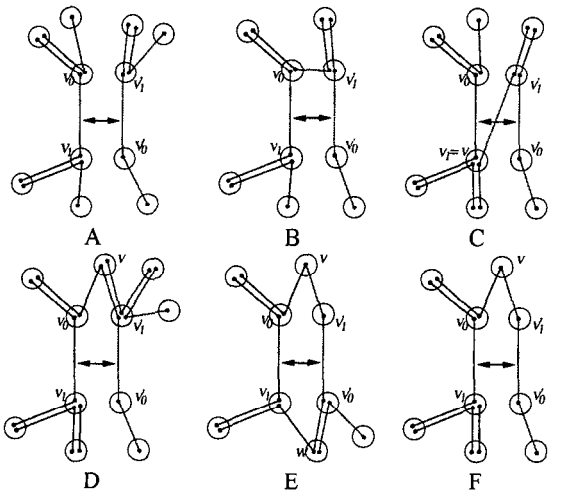

Figure 10: Different configurations for a proposed stitch between (v_0, v_1) and (v_1', v_0').

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.

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 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. 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. 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. 10A; 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. 10B shows this configuration. The stitch cannot be performed since it creates a self-loop edge which is prohibited in our surface model.

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. 10C,D,E and F. 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 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. 10C and 10E. A procedure called ClassifyMerging(v, v') evaluates the effect of merging two corner groups, v and v' and classifies the 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 ClassifyMerging(v_0, v_1') and perform the merging if there is an implicit stitch (3).

Step II We evaluate ClassifyMerging(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 first two steps sequentially. In the case of Fig. 10C, the above procedure will merge v_0 with v_1' in Step I. However, ClassifyMerging(v_1, v_0') will prevent the second merging in Step II. Figure 10D shows another case where order is important. ClassifyMerging() works by maintaining a list of current edges incident to the corners of a group. ClassifyMerging() 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 4. 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.

6 Examples

Conversion of Non-Manifold Surfaces 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 the Pinching or Snapping Strategies 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

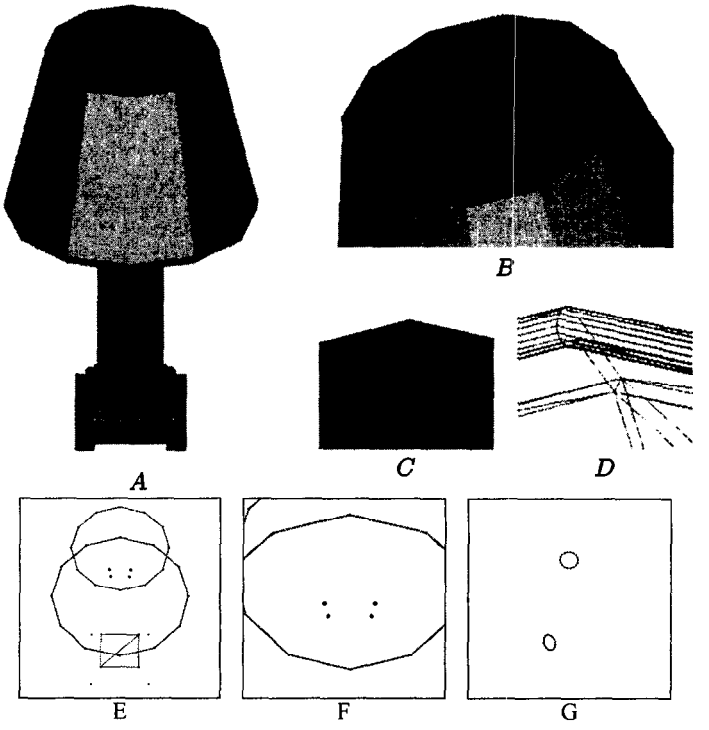

Figure 11: Lamp model. A: general view. B, C, D: successive details showing edges shared by more than two faces. E,F,G: Singular edges are shown in red and singular vertices in black in increasingly detailed views.

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. 11A. The original model had 5054 triangles and 2810 vertices; we discovered 125 singular edges and 128 singular vertices. After conversion by cutting through singular edges and vertices, there were 5052 triangles and 3058 vertices. Fig. 11A shows the various connected components after conversion using different colors. The conversion took less than one second with an IBM RS6000 580.

The second example is a polygonal model of the space ship Enterprise with 12539 triangles and 15011 vertices. Fig. 13A (on the color page) 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 the Snapping Strategy, there were 12552 triangles and 7429 vertices. The conversion took 21 seconds using an IBM Power PC 42T. This example is a good advocate for automated correction methods: asking a user to decide on how to locally connect the surface 1800 times seems impractical.

The third example is a polygonal approximation of an iso-surface extracted from a CT-scan of a fossil monkey jaw, illustrated in Fig. 13A. The original model had 75842 triangles and 37624 vertices. We discovered 462 singular edges and 563 singular vertices; singular edges are shown in Fig. 13B and singular vertices in Fig. 13C. Although non-manifold iso-surfaces are justifiable in the general case, in this case singularities came from an incorrect algorithm. 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 singu-

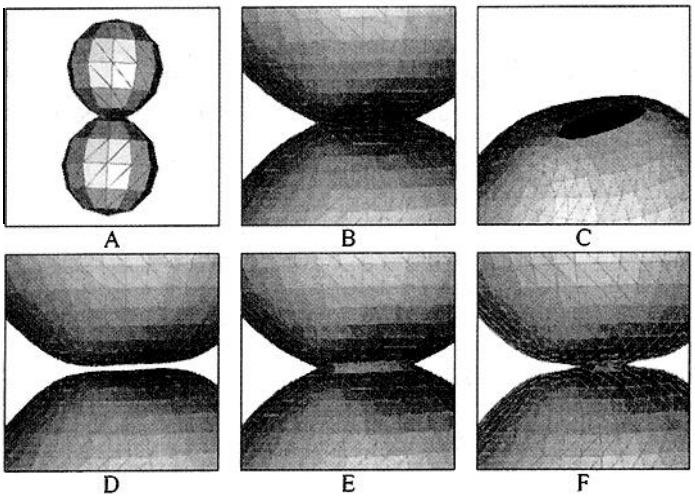

Figure 12: Surface smoothing and singularities A. non-manifold model of two spheres sharing two edges. B: after subdividing and smoothing. C: after cutting, subdividing and smoothing. D: after pinching boundary edges, subdividing and smoothing. E: after snapping boundary edges. F: another result of edge snapping.

lar. After removing invalid triangles, we discovered 2 remaining non adjacent singular edges and 10 singular vertices. After cutting through singular edges and vertices, we obtained 75371 triangles and 37636 vertices. The conversion took 6 seconds with an IBM RS6000 580 workstation, including the removal of invalid faces. 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.

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 [7]: 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 edges and vertices. We use the example of two spheres sharing two edges, providing a non-manifold model in Fig. 12G. In Fig. 12H we attempt to subdivide and smooth G, and notice the non-smooth behavior in the vicinity of the singular edges. In Fig. 121, we first cut through the singular edges using the methods of Section 4 and then subdivide and smooth. In Fig. 12J, we smooth after cutting and stitching using the Pinching strategy. In Fig. 12K, we smooth after cutting and stitching the Snapping Strategy. Fig. 12L illustrates another outcome when stitching using the Snapping strategy as well.

7 Related Work

Our method is very different from most of the previous work as it operates solely on the surface topology. The first category of prior art methods operate both on the geometry and topology to modify surfaces so that they can represent the boundary of solids [11]. This is an important issue for Rapid Prototyping of models represented using the .STL format consisting of topologically disconnected triangles (see [12, 13, 6, 14]). 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) with a "reasonable" polygonal surface without creating intersections [15]. Relatedly, Butlin et *al.* [16] attempt to "repair" CAD data in order to use it for engineering analyses or to simplify data exchange. Barequet and Kumar [17] operate on STL files; as with the global cutting method of Section 4.2, they first stitch through regular edges, but they can subsequently create a non-manifold after stitching additional edges. Murali and Funkhouser [18] start from polygon faces to partition the volume in cells, and determine if each cell is solid. From the solid cells, they produce a manifold boundary representation.

The second category consists of tools to create and manipulate surface models. The technique of Szeliski et *al.* [19] 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. The technique of Welch *et al.* [20] 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 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 indices. Veron and Leon [21] 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 and tools for building boundary representations of solids. It is related because conversions between manifold and non manifold representations are discussed, for instance in Hoffman [22] (see also [23, 24, 25]). Heisserman [26] 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.

8 Conclusion

We have used cutting and stitching for the automatic conversion of a set of polygons to a manifold polygonal surface, that can potentially exhibit self-intersections (which are not treated). All properties are passed on to the output surface. Face and corner properties are unchanged, as our method preserves corners. When a vertex is multiplied, the same properties are assigned to all copies.

We have successfully applied this conversion to extend algorithms for surface simplification [3] and compression [4], to enable processing of non-manifold surfaces. This method was successfully applied to pre-process non-manifold polygonal surfaces before simplification in IBM Data Explorer [27]. This method may not be suitable if the original surface was intended to be a non-manifold, i.e, 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 user 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. Other strategies can be developed for stitching, depending upon the application.

A 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.

By looping through the face array, we build a structure of n_e surface edges, recording the number of incident faces. For effi-

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

Table 1: Statistics on conversion and stitching using the Snapping Strategy applied to VRML 2.0 data available on the World Wide Web. 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.

ciency, 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.

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 [28]; 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 5.2.

REFERENCES

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

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

[3] 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.

[4] G. Taubin and J. Rossignac. Geometry Compression through Topological Surgery. Technical Report RC-20340, IBM Research Division, January 1996.

[5] H. Hoppe. Efficient implementation of progressive meshes. Technical Report MSR-TR-98-02, Microsoft Research, Redmond, Washington, January 1998.

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

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

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

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

[10] 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.

[11] M. Segal and C.H. Sequin. Partitioning polyhedral objects into non-intersecting parts. *IEEE Computer Graphics & Applications*, 8(1):53–67, January 1988.

[12] I. Makela and A. Dolenc. Some efficient procedures for correcting triangulated models. In *Proc. Symp. on Solid Freeform Fabrication*, pages 126–34, July 1993. Dept. of Mech Eng., Univ of Texas at Austin.

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

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

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

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

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

[18] T.M. Murali and T.A. Funkhouser. Consistent solid and boundary representations from arbitrary polygonal data. In *Symposium on Interactive 3D Graphics*, pages 155–161, Providence, RI., apr 1997. ACM.

[19] 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.

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

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

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

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

[24] 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.

[25] 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.

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

[27] 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.

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

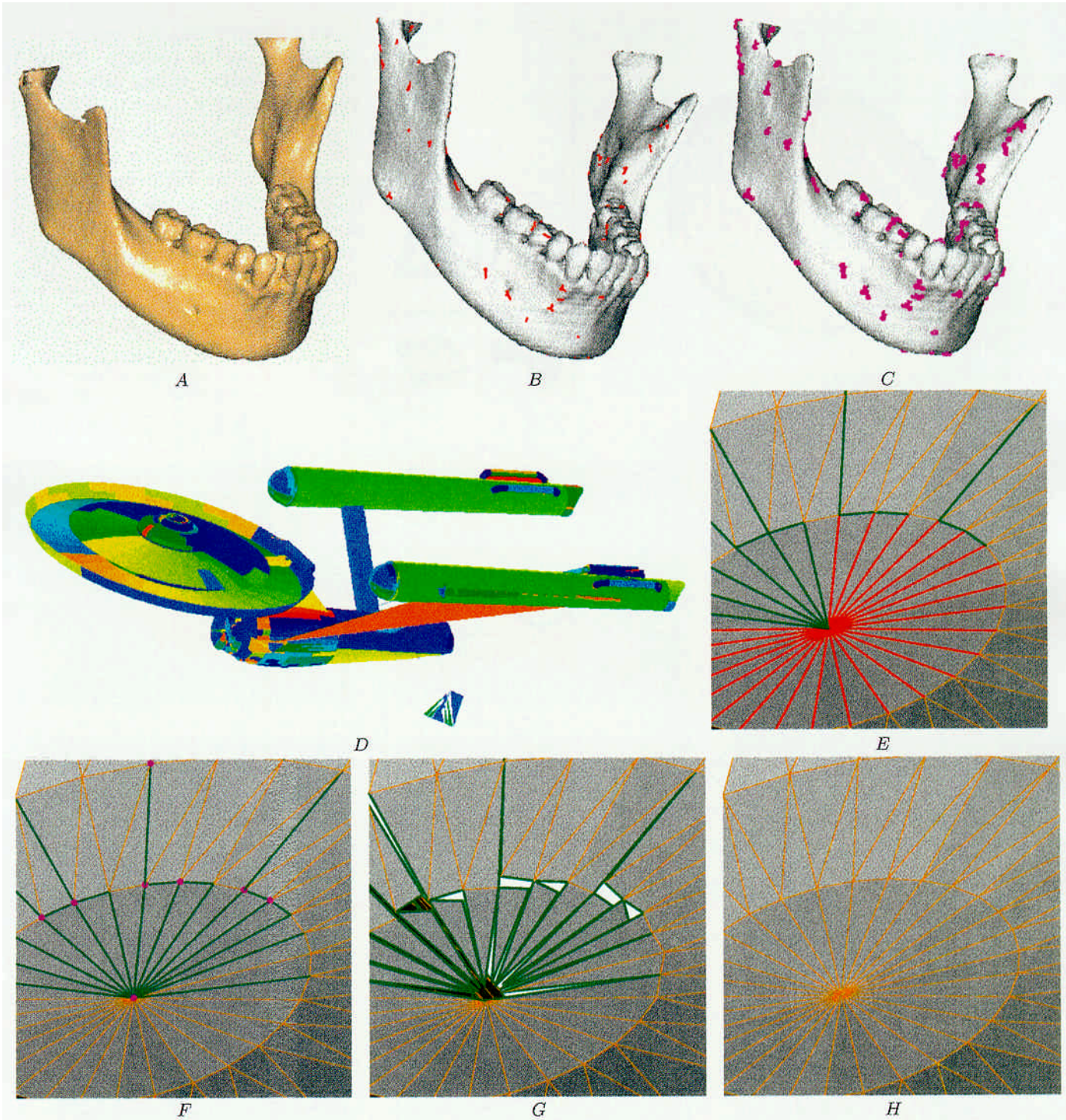

Figure 13: Fossil monkey jaw (non-manifold iso-surface). A: general view. B: singular edges are shown in red. C: singular vertices are shown in magenta.
Enterprise model. D: general view: disconnected surface components are painted using different colors. E: detail: singular edges are shown in red; boundary edges are shown in green; regular edges are shown in orange. F: same as E, after removing invalid triangles; singular vertices are shown in magenta; no more singular edges are visible in this area. G: after cutting through singular edges and vertices. H: after stitching using the Snapping Strategy.