Generating Topological Structures for Surface Models

Rapid prototyping has imposed new requirements on the processing of CAD models. To manufacture an object directly, the model must correspond to a real object and thus represent a closed volume. Its boundary must be continuous and closed, and no cracks or improper intersections are allowed. CAD models cannot always meet these requirements, especially when they are constructed by a surface modeler. Unlike a solid model, a surface model consists of a collection of separate surfaces. The topological information is either absent or insufficiently represented. Topological structure is also lost in the geometric data exchange between different systems via standard formats like IGES (Initial Graphics Exchange Specification) and VDA-FS (Verband der Deutschen Automobilindustrie-Flächenklassifizierung, the surface-interface standard of the German automotive industry association). These formats are deficient in or incapable of transferring topological information. In general, the standard formats do not guarantee either closure or an unambiguous solid. Commercial CAD systems are seldom capable of generating the topological structures of transferred geometries.

The lack of topological information in surface models causes a number of difficulties in interfacing them to rapid prototyping (RP) processes. Without this information, verifying the completeness and correctness of models is difficult. The trimming curves of two neighboring surfaces or their boundaries do not exactly match, even when they result from the same intersection of surfaces. By transforming the surfaces into .STL file format, the de facto standard interface to RP processes, cracks and improper intersections may occur. Consequently, the layer boundaries of the model after slicing might not be contiguous or complete. This can result in cross-section filling errors and thus rejected parts.

Since solid models provide a complete and unambiguous representation of physical objects, converting a surface model to a solid representation offers a reliable way of interfacing surface models to RP processes. This would make it easy to generate valid .STL files. Although various researchers have studied the conversion between different solid representations, the problem of converting from surface to solid models has rarely been addressed. A method for analyzing adjacency in B-spline surface models has been reported. The method, however, only detects one common interval along one standard edge of each surface. It cannot process trimmed patches and surfaces, and its robustness in handling complex geometric situations is unknown.

Bohn and Wozny developed an algorithm to repair .STL files. The algorithm first identifies a set of directed Jordan curves representing lamina (unmatched) boundaries in an .STL model, then it triangulates the curves separately to close holes on the model boundary. However, basing the model correction directly on .STL files has several shortcomings from both an algorithmic and an applications point of view. Real-life .STL files are normally huge, implying considerable computation effort to derive the Jordan curves from a large number of triangles. Geometric and topological ambiguities are difficult to handle on the basis of triangles. For instance, it is not trivial to correctly treat a set of coincident triangles representing two duplicate surfaces coinciding in space. Improper intersections between adjacent surfaces in the original model are difficult to detect. Triangulating the unmatched boundaries of adjacent surfaces tends to create skinny, degenerate, or even wrong triangles that produce additional intersections.

This kind of problem is easy to detect and resolve at the higher level represented by surfaces. Also, from an aesthetic point of view, manipulating the model geometry on the basis of surface data is more likely to produce well-formed results. In this article, we present an algorithm that builds topological structures for surface models and generates a topological structure for them that meets rapid prototyping requirements for a closed volume with a continuous, closed boundary.
Rapid Prototyping Technology

1 Surface versus B-rep solid model.

2 Structure of the node Edge.

models consisting of trimmed and untrimmed parametric surfaces and containing no a priori information about surface adjacency. After the algorithm establishes a topological structure in a model, it is easy to generate an .STL file that satisfies the so-called vertex-to-vertex condition, which is identical to the manifold criteria defined in solid modeling (and presented in the next section). In fact, the task of building topological structures for surface models is equal to that of converting them into solid models.

The conversion problem

Figure 1 presents a schematic of the conversion problem, which can be formulated as follows: Given a surface model $M_{sf}$ consisting of a set of trimmed parametric surfaces $\{s_k\}$ and a tolerance $\varepsilon$, find a B-rep solid model $M$, such that the maximal deviation between $M_{sf}$ and $M$, does not exceed the tolerance $\varepsilon$.

Let $P_i$ be a point on the boundary of $M$, and $P_{sf}$ be the point on $M_{sf}$ nearest to $P_i$. Then the conversion must satisfy

$$\sup_{P_i \in M} \|P_i - P_{sf}\| \leq \varepsilon$$

with $P_{sf} \in M_{sf}$ and $\|P_i - P_{sf}\| = \text{min}$

If such a solid model does not exist, problem areas in $M_{sf}$, such as missing surfaces and improper intersections, are noted by highlighting them.

Without loss of generality, let $M_{sf} = \{s_k\}$ be a set of trimmed piecewise polynomial surfaces. (Representations such as Bezier and B-spline can be converted into polynomial form.) A piecewise polynomial surface $s_k$ is a grid of parametric patches defined as the tensor product of two parametric variables $u$ and $v$:

$$s_k(u,v) = \sum_{j=0}^{m} \sum_{i=0}^{n} a_{ij} u^j v^i$$

$$u_1 \leq u \leq u_2, \quad v_1 \leq v \leq v_2$$

The surface is thus defined over the bounded rectangle of parameters $u$ and $v$, which is called parametric space. The trimmed region of the surface is defined as the common region of the interior of the outer boundary and the exterior of each of a set of inner boundaries. The boundary curves are referred to as trimming curves.

The trimming curves are defined in the parametric space

$$c_k(t) = (u(t), v(t)) \quad t_1 \leq t \leq t_2$$

where $u(t)$ and $v(t)$ are in turn polynomials of $t$. Note that the outer boundary curve of an untrimmed surface consists of the four line segments bounding its parametric space and that no inner boundary curves exist.

The goal representation of the conversion is the B-rep solid $M$, which is a directed graph $(V, E, F)$ with a finite number of vertices $V = \{v_i\}$, edges $E = \{e_i\}$, and faces $F = \{f_i\}$ bounded by edges and vertices. In this representation, $\{f_i\}$ defines a bounded and closed subset of the three-dimensional Euclidean space $E^3$. Each face of the graph has a certain orientation. Faces, edges, and vertices are labeled according to their position and orientation. If a collection of edges or vertices has the same label, the collection is identified and treated as a single edge or vertex, respectively. Note that the graph can represent both planar-faced and curved solids. For a curved solid, $F$ and $E$ are curved faces and curves, respectively, while $F$ and $E$ represent polygons and line segments, respectively, in a planar-faced model.

There are many ways to represent the graph $(V, E, F)$, such as Baumgart's winged edges, Mäntylä's half edges, Kalay's hybrid edges, and Weiler's edge uses. In our conversion procedure, we use a hierarchic data structure similar to the half-edge structure and consisting of five node types, as shown in Figure 1:

- The node Solid represents shells in the model and is the root node of the data structure.
- The node Face comprises a list of loops, one loop indicating the outer boundary of the face and the others representing holes in the face.
- The node Loop is a boundary curve of a face.
- The node Edge describes one line segment of a loop and contains pointers to its vertices and to its counterpart in another face. Thus, the node Edge embodies the adjacency relationships between faces and is the key for representing the topology information in a B-rep model. As depicted in Figure 2, the node contains an additional pointer, Mate, pointing to its counterpart. Observe that these two edges have the same vertices but different orientations.
- The node Vertex contains the $x$, $y$, and $z$ point coordinates.
To ensure the correctness of the conversion, the model converted from a surface model must satisfy the following manifold criteria:

- Every edge is shared by exactly two faces.
- Every vertex is surrounded by a single cycle of edges and faces.
- Faces do not intersect each other except at common edges and vertices.
- Shells have consistent orientations.

The kernel of the conversion procedure is a so-called topology analyzer, which first finds the matching surfaces for each surface in the model. Two surfaces are said to be geometrically matching or partly matching if the shortest distance between their boundaries falls within the specified tolerance \( \varepsilon \). Then, the analyzer merges the matching boundaries according to the four manifold criteria. Using a curve subdivision theorem, the analyzer can linearly approximate the matching boundaries to obtain a set of points for each of them. Thus, the remaining work relates to the merging of the two point sets, which we describe in detail in the next section. The STL model finally results from triangulating the trimmed regions of surfaces.

**The topology analyzer**

The topology analyzer creates the topological structure for a surface model by merging surface boundaries that lie within the given tolerance \( \varepsilon \). Before describing its implementation in detail, we first give formal definitions for matching and merging: Suppose that surface boundary curves have been linearly approximated by line segments, which we now call edges. Two points \( p_1 \) and \( p_2 \) are said to match if \( |p_1 - p_2| \leq \varepsilon \). They are said to merge if they have the same Euclidean coordinates \( p_1(x, y, z) = p_2(x, y, z) \). Two edges \( e_1 \) and \( e_2 \) are matched if \( \min |e_1 - e_2| \leq \varepsilon \), and they are merged if their endpoints merge. Two curves \( c_1 \) and \( c_2 \) match if \( \min |c_1 - c_2| \leq \varepsilon \) and merge if at least one set of their matching edges merges.

Figure 3 shows the implementation of the topology analyzer for two boundary curves \( c_1 \) and \( c_2 \). The implementation consists of three main steps:

1. Linear approximation of the surface boundary curves.
2. Finding all matching boundaries for each surface boundary.
3. Merging the matching boundaries.

**Linear approximation of boundaries**

To linearly approximate boundary curves, recall a theorem that states:

\[
\sup_{t \in [a, b]} \|c(t) - l(t)\| \leq \frac{1}{8} (b-a) \sup_{t \in [a, b]} \|c''(t)\|
\]

where \( c(t) \) is a \( C^2 \) curve over \( t \in [a, b] \) and \( l(t) \) is the edge with \( l(a) = c(a) \) and \( l(b) = c(b) \). Given an approximation tolerance \( \delta \), it is easy to determine the number \( n \) of edges to be generated:

\[
n = \left| b - a \right| \sqrt{\frac{\sup_{t \in [a, b]} \|c''(t)\|}{8\delta}}
\]

Since the boundary-curve approximation occurs in Euclidean space, the approximation tolerance must be smaller than that used for the conversion, that is, \( \delta < \varepsilon \). The boundary curves are then segmented into the same pieces in parametric space.

**Matching boundaries**

Recall the definition that two boundary curves \( c_1 \) and \( c_2 \) are geometrically matching if they lie—in whole or in part—within \( \varepsilon \). We determine this by finding the minimum distance between them. Since two curves have been approximated by two sets of line segments, or edges \((e_1^1) \) and \((e_2^2) \), respectively, the problem reduces to finding the minimum distance \( d_0 \) between two edges \( e_1^1 \) and \( e_2^2 \). Thus, the matching condition can be reformulated as follows:

\[
\min (d_0) \leq \varepsilon
\]

For simplicity, we calculate only the shortest distances between the endpoints of \( e_1^1 \) to edge \( e_2^2 \) and vice versa. The minimum of these four distances is \( d_0 \).

Correspondingly, we reformulate the matching con-
Thus, the boundary curves geometrically match if at least one pair of such matching points exists. To find matching curves for a specific boundary curve requires checking that curve against all other boundary curves. This check requires the most computation time in the boundary-matching procedure. To speed up the process, practical implementations use the bounding box of each curve and thereby avoid unnecessary calculations of the minimum distances between curves.

**Merging boundaries**

This step generates the topological information for matching curves by merging the appropriate matching points (or vertices) into identical points in Euclidean space and considering their corresponding edges as identified. Note that the matching point coordinates in parametric space remain the same after merging. This step also corrects the orientation of boundary curves.

According to the manifold criteria, each edge on the boundaries can be merged only once, while the vertices can be merged many times. Therefore, the implementation of the merging procedure must be edge-based. This means that vertices can be merged with each other if their edges are merged. The result is a so-called *zipper merging procedure*: Starting with a pair of merged edges, the merging procedure expands in both directions until all edges are matched or the endpoints of the next pair of edges do not match (see Figure 5).

Let $c_1$ and $c_2$ be a pair of matching curves. The merging procedure can be formulated as follows:

1. Set the reference curve.
2. Search a pair of unique matching edges.
3. If no pair of such edges are found, go to step 6.
4. Zipper $c_1$ and $c_2$ bidirectionally from the matching edges until no matching edges are found in either direction.
5. Go to step 2.
6. Quit.

Since the vertices of two matching curves take the same Euclidean coordinates after merging, one curve is selected as the reference curve whose coordinates define the single vertex used. Our implementation selects the curve with more original points. After it selects the reference curve, the algorithm finds a pair of edges that uniquely match each other. Two edges are said to be uniquely matching if their two pairs of endpoints match each other and none of these endpoints matches with a third point (see Figure 6). (Note that we do not consider the orientation of two edges in this stage.) Practical applications show that such uniquely matching edges normally exist.

The two edges are then merged by assigning the same Euclidean coordinates to their endpoints and generating the Mate pointers.

Next, the merging edges expand in both directions. The expansion only requires the endpoints to be
matched; the matching need not be unique since the topology of the surface boundary guarantees the correct merging from this moment. The expansion continues until no more matching points are found. If the merging procedure has checked all endpoints on \( c_1 \) and \( c_2 \) for expansion, it is done. Otherwise, it looks for a new pair of unique matching edges on another part of \( c_1 \) and \( c_2 \), and repeats the merging procedure.

Figures 7 and 8 demonstrate some robustness of the merging procedure. Figure 7 shows a situation where three points, \( p_1, p_2, \) and \( p_3 \), match. Thus, \( e_5, e_6, \) and \( e_7 \) also match. The question is then whether \( e_5 \) or \( e_7 \) should be merged with \( e_6 \). Recall that the merging starts from a pair of unique matching edges. Since \( e_1 \) and \( e_2 \) uniquely match and are merged, it is easy to see that \( e_3 \) and \( e_4 \) must be merged. Therefore, \( e_6 \) and \( e_7 \) are a pair of merged edges and \( p_3 \) is merged with \( p_2 \).

Figure 8 depicts a similar situation, where \( e_1 \) and \( e_2 \) are already merged. It follows that \( e_1 \) and \( e_4 \) are a pair of merging edges and \( p_1 \) is merged with \( p_3 \). The merging edges cannot expand because \( p_2 \) does not match with \( p_4 \). Hence, the expansion restarts from another direction. The merging of \( e_5 \) and \( e_6 \) leads to the merging of \( e_8 \) with \( e_9 \). This results in a merging between \( p_2 \) and \( p_3 \). Since \( p_1 \) is already merged with \( p_4 \), the edge between \( p_1 \) and \( p_2 \) becomes a null edge (with zero edge length). This causes some modifications of curve boundaries. These modifications, however, are all within the given tolerance.

A B-rep solid model uniformly defines the normal vectors of all surfaces as pointing either away from or into the solid. Since the original surface model may contain surfaces whose normals are not uniformly defined, the orientations of their boundaries must be checked. We can do this easily by checking pairs of merged edges. A pair of merged edges always have opposite orientations. Thus, starting with a pair of merged edges, the correction procedure runs recursively until all edges have been processed.

**Incremental matching and merging**

If the matching tolerance is too large, it can cause multiple matching points and edges so that no unique matching edges can be found (see Figure 9). Therefore, the topology analyzer runs the matching and merging incrementally in the practical implementation. That is, given a tolerance \( \varepsilon \), each procedure starts with a smaller tolerance \( \varepsilon_0 < \varepsilon \), then repeats with an increased matching tolerance \( \varepsilon_0 = \varepsilon_0 + \Delta \varepsilon \) until all boundaries are...
To generate .STL models requires only separately triangulating these surfaces.

We use a triangulation algorithm that we developed for trimmed surfaces. It performs the triangulation completely in parametric space. The trimmed region is first mapped into the parametric space and approximated by 2D polygonal regions. These regions are then pretriangulated by the algorithm—a restricted Delaunay algorithm that extends the well-known Delaunay triangulation for handling polygons with holes. The generated triangles are further subdivided until each edge of the triangles is smaller than the length determined by the surface definition and specified tolerance. All triangles are finally mapped back into Euclidean space to calculate the coordinate triples for the triangle vertices.

This triangulation algorithm also includes a strategy we developed to prevent cracks and improper intersections between patches and surfaces during the generation of triangles. Interested readers can refer to our prior publication for more details. Figure 12 shows the .STL model of the schnapps bottle in Figures 10 and 11.

**Conclusions**

The algorithm presented here correctly interfaces surface models to RP processes. Its topology analyzer creates the topological structure for surface models by merging the surface boundaries according to the manifold criteria, if the boundaries are within a given tolerance.

The algorithm requires no a priori information about surface adjacency. The models are finally triangulated based on the topological structure, and the results satisfy the vertex-to-vertex condition set by the .STL format.

We have implemented the algorithm in C and integrated it in a geometric interface between neutral CAD surface data and StereoLithography Apparatus (SLA) machines from 3D Systems (Valencia, California). The robustness of the algorithm in correctly handling a variety of complex geometric situations has been demonstrated successfully in the manufacture of numerous SLA parts containing anywhere from tens to hundreds of parametric surfaces (trimmed and untrimmed) in VD-AFS formats.

Practical experience shows that most problems encountered in real-life surface models occur in boundary areas between neighboring surfaces. For example,
design and numerical inaccuracies caused by representation conversions during data exchange result in inexp- act matches between two neighboring boundaries. This creates small gaps and improper intersections between surfaces if the surfaces are triangulated independently. .STL is not likely to correctly handle areas containing improper intersections. As our description of the topology analyzer shows, this kind of problem can be solved using surface boundary data. Since automatic correction of CAD models might significantly violate the original design intention, our program runs interactively. If problems such as missing surfaces cannot be corrected within a given tolerance, the program graphically highlights them for construction by the CAD system. In this sense, the program can also be used as a tool for designers to verify the completeness of CAD models.

Although the algorithm was developed for interfacing CAD surface models to RP processes, it can also serve many other engineering purposes, such as generating finite-element modeling meshes or reconstructing object models from multi-viewed scan data. Currently, we are exploring these kinds of application possibilities with the algorithm.

Acknowledgments
This work has been partly supported by the BRITE (Basic Research of Industrial Technology for Europe) program (project no. 3527) and by the German Science Foundation DFG–Deutsche Forschungsgemeinschaft (project no. HI 424/4-1). We would like to thank our colleague Marcus Joppke who has successfully implemented the triangulation algorithm and produced the triangulation model used in this article. Thanks are also due to Fran Wijgorra for her help in preparing the article.

References

Xuejun Sheng is a research engineer at the Bremen Institute of Industrial Technology and Applied Work Science at the University of Bremen (BIBA). His research interests include geometric modeling, computer graphics, CAD/CAM, and robotics. He received a BS in electrical engineering from the North China Institute of Electrical Power in 1982, the equivalent of Dipl.-Ing. in electrical engineering from the University of Saarland, Germany, in 1986, and a PhD in electrical engineering at the University of Bremen, Germany, in 1992.

Ingo R. Meier is a research engineer at the Bremen Institute of Industrial Technology and Applied Work Science at the University of Bremen (BIBA). His scientific interests include geometric modeling, neural networks, and robotics. He studied computer science at the University of Bremen.

Readers may contact Sheng at BIBA, Hochschulring 20, 28359 Bremen, Germany, e-mail stan@biba.uni-bremen.de.