INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2006; **52**:823–841
Published online 21 March 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.1166

Automatic CAD model topology generation

Paresh S. Patel*,†, David L. Marcum‡ and Michael G. Remotigue§

Computational Simulation and Design Center, ERC, Mississippi State University, Mississippi State, MS 39762, U.S.A.

SUMMARY

Computer aided design (CAD) models often need to be processed due to the data translation issues and requirements of the downstream applications like computational field simulation, rapid prototyping, computer graphics, computational manufacturing, and real-time rendering before they can be used. Automatic CAD model processing tools can significantly reduce the amount of time and cost associated with the manual processing. The topology generation algorithm, commonly known as CAD repairing/healing, is presented to detect commonly found geometrical and topological issues like cracks, gaps, overlaps, intersections, T-connections, and no/invalid topology in the model, process them and build correct topological information. The present algorithm is based on the iterative vertex pair contraction and expansion operations called *stitching* and *filling*, respectively, to process the model accurately. Moreover, the topology generation algorithm can process manifold as well as non-manifold models, which makes the procedure more general and flexible. In addition, a spatial data structure is used for searching and neighbour finding to process large models efficiently. In this way, the combination of generality, accuracy, and efficiency of this algorithm seems to be a significant improvement over existing techniques. Results are presented showing the effectiveness of the algorithm to process two- and three-dimensional configurations. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: CAD; mesh; triangle/polygon; repair/heal/reconstruct; cleanup; stitching; filling

1. INTRODUCTION

Computer-aided design, analysis, optimization and manufacturing have become an integral part of the product development process in automotive, aerospace, electronics, and many other industries. Computer-based design process begins with creating a detailed geometry model in a computer aided design (CAD) system. This CAD model is the starting point for many downstream applications such as mesh generation, structural/fluid/thermal analysis,

*Correspondence to: P. S. Patel, Computational Simulation and Design Center, ERC, P.O. Box 9627, Mississippi State, MS 39762-9627, U.S.A.
†E-mail: patel@erc.msstate.edu
‡E-mail: marcum@erc.msstate.edu
§E-mail: remo@erc.msstate.edu

Received 10 November 2004
Revised 3 November 2005
Accepted 7 November 2005

rapid prototyping, numerical controlled machining, casting, computer graphics, and real-time rendering. Each of these downstream applications has specific requirements for the geometry definition and representation. Hence, success of the downstream application strongly depends on accuracy and consistency of the input geometry.

Computational fluid dynamics (CFD) simulation process has several stages including pre-processing, flow solution and post-processing of the results. Typically, pre-processing includes geometry cleanup and mesh generation to discretize the computational domain. In recent years many automatic structured and unstructured mesh generation methods emerged. Most of these methods require a suitable (i.e. a clean well connected water-tight) geometry to start the grid generation process. Unfortunately, CAD models of complex geometries translated through neutral file formats like Initial Graphics Exchange Specification (IGES) and StereoLithography Interface Specification (STL) have many geometrical and topological issues that prevent automatic creation of a water-tight geometry. They have many gaps, cracks, holes, overlaps, T-connections, invalid topology, and inconsistent orientation. As a result of these issues or errors, true automation of the grid generation process is still elusive. The analyst has to manually clean the geometry to make it suitable for grid generation. This cleanup (pre-meshing) process is very time consuming, expensive and tedious task for a design/analysis engineer. For realistic simulations, this is the single most labour-intensive task in the process, preventing true auto-meshing.

An algorithm is developed to detect the commonly found geometrical and topological issues and process them automatically to build topology information. The present algorithm is based on the iterative vertex pair contraction and expansion operations called *stitching* and *filling*, respectively. The edge-split operation does not only make these operations more reliable and accurate, but also allows to process the CAD models with non-uniform spacings. The algorithm closes small gaps/overlaps via the *stitching* operation and fills larger gaps by adding new faces through the *filling* operation to process the model accurately. The processed CAD models are guaranteed to be free of intersecting faces or surfaces, which is desirable for many applications. This algorithm is general and can process manifold as well as non-manifold geometry models. Moreover, the present algorithm uses a spatial data structure, octree, for searching and neighbour finding to process large CAD models efficiently. The current procedure is based on an assumption of the locality of geometrical and topological issues. It is assumed that the neighbouring surface patches have small gaps/overlaps due to the translation errors, numerical inaccuracies, and tolerance settings in different systems. In practice gaps/overlaps between the surfaces are generally very small, so the key assumption of locality is reasonable. However, there are many other issues that may cause large gaps/overlaps. For example untrimmed or missing surfaces may create large gaps and overlaps. These issues may violate the assumption of locality and can be processed up to some extent.

2. RELATED WORK

Butlin and Stops [1] addressed data exchange problems due to the transfer between different CAD/CAE software systems, which is usually required to get combinatorial benefit of different vendors' systems in the simulation process. Different geometric healing tools are developed to repair these geometric anomalies produced due to data transformation. Jones and Butlin [2] developed a toolset to repair the geometry and make the process of generating analysis models easier and more reliable. Mezentsev and Woehler [3] have also addressed the problem of geometry pre-processing for finite element analysis mesh generation application. Steinbrenner *et al.* [4] have developed an approach to create a fast surface mesh on imperfect CAD models with gaps (less than the user specified tolerance) without repairing it. Their ongoing work of automatic formation of a surface mesh spanning multiple surfaces is also described.

Adaptive Cartesian grid generation method [5] based approaches [6, 7] can also be used for CAD cleanup and surface triangulation. Hu *et al.* [6] have utilized an overlay grid, obtained through Cartesian grid generation, to cleanup and reconstruct the geometry. Intersecting points of the overlay grid and geometry (water-tight volume) is reconstructed using a point cloud. In their work, it has been reported that this approach does not work for a complex configuration. Wang and Srinivasan [7] have also demonstrated the use of an adaptive Cartesian grid generation method for 'dirty' geometry clean up to get a surface triangulation of a complex configuration. Geometry is used for getting the point intersections/projections to reconstruct the surface from these points. Further improvement of these approaches can be achieved by targeting only the bad areas with gaps and overlaps. Feature-based geometry refinement is also needed to generate the Cartesian grid in both methods. Cartesian mesh based approaches reconstruct the geometry approximately using the intersecting/projection points information hence output geometry is not accurate. Even if there is a small error (i.e. gaps or overlap) in some part of the geometry, these approaches rebuild the entire model approximately and accuracy depends on the refined Cartesian mesh cell (octant) size. Moreover, it may not be efficient to find many intersection/projection points if the Cartesian mesh is very fine, which is needed to achieve accuracy.

Many computer graphics and real time rendering applications also require an error free input geometry model. Baum *et al.* [8] developed a series of algorithms to preprocess the input geometry to meet the requirements of mesh based radiosity computation algorithms. Back-face culling, a technique to render complex models quickly [9], requires that the polygons in the model are oriented consistently. Murali and Funkhouser [10] described an algorithm based on space subdivision to construct a consistent solid and boundary representation from polygons. Gueziec *et al.* [11] developed greedy strategies to convert a set of non-manifold polygonal surface to a manifold. The aim of their work is to modify the topology of surfaces, not to correct geometrical errors.

Stereo Lithography (STL) is a widely used data exchange format in the Rapid Prototyping industry. In order to manufacture models correctly, input geometry must be geometrically and topologically correct. However, real world geometries translated through the STL files generally have many geometrical and topological errors like gaps, overlaps, intersections, inconsistent orientations, etc. Rock and Wozny [12] have used an AVL tree data structure to locate neighbour vertices efficiently to build model topology from a given set of unordered triangular facets. Bohn and Wozny [13] described a solution to achieve shell-closure of polyhedral CAD-models by extracting and triangulating the directed Jordan curves in three-dimensions to fill gaps. Makela and Dolenc [14] developed methods to handle overlapping and intersecting triangles efficiently. Sheng and Meier [15] have used a technique based on incremental matching and merging of boundaries of surface models to repair gaps. Morvan and Fadel [16] developed a virtual environment to correct the errors in a given model interactively. Barequet *et al.* [17, 18] used a computer vision technique called geometric hashing [19, 20] to repair geometrical and topological errors in the boundary representation (b-rep) of two-manifold geometry models.

3. THE PROBLEM DEFINITION

Processing CAD models for a downstream application like mesh generation is a major bottleneck in the entire CFD simulation process. The goal is to develop an algorithm to detect commonly found geometrical and topological issues, to process them and to build the topology information for the model automatically. Commonly found geometrical and topological issues include:

(i) Small gaps or cracks between the boundaries of surfaces;
(ii) Small overlap of surfaces;
(iii) T-joints;
(iv) No or invalid topology information.

CAD models are often represented as a set of triangulated surfaces in three-dimensional Euclidean space R^3 [21]. Let us define a geometry model $M=(V,F)$ as a set of vertices V and a set of triangular faces F. The vertex list $V=(v_1,v_2,\ldots,v_m)$ is an ordered sequence of vertices. Each vertex v_i is defined by three coordinates (x_i,y_i,z_i) and a unique index. The face list $F=(f_1,f_2,\ldots,f_n)$ is also an ordered sequence of faces. Every face or triangle f_i is defined by an ordered list of three vertex indices (j,k,l) and a unique index. A face made of vertices v_j, v_k and v_l can be denoted as $\triangle v_j v_k v_l$. An edge e_i is defined by an ordered sequence of two vertex indices (j,k). It can be denoted as $\overline{v_j v_k}$. An edge with one incident face is called a boundary edge and its end points are called boundary vertices. An edge with two and more than two incident faces is called a manifold edge and non-manifold edge, respectively. Note that the geometry model does not have topology or adjacency information. Topology information tells how geometric objects are connected. Many downstream applications need such information for further use of geometry models.

It is assumed that the gaps/overlaps due to the translation errors, numerical inaccuracies, and tolerance settings in different systems are small as compared to the model size. In practice gaps/overlaps due to these issues are generally very small and results demonstrate that the procedure works well for many realistic test cases. Moreover, the present algorithm can process models with non-uniform sizing functions and skinny triangles because the edge-split operation can take care of the non-matching neighbouring boundary edges with different point distributions.

4. ALGORITHM OVERVIEW

The present algorithm is based on the iterative vertex pair contraction and expansion operations (with and without edge-split) to process the geometrical and topological issues. The edge-split operation makes the vertex pair contraction [22, 23] and expansion operations more reliable and accurate, and allows to process the models with non-uniform spacing. It seems many previous efforts [13–18] assume that geometry models to be processed are two-manifold or use some special procedure to be able to handle non-manifold model like a two-manifold model. This assumption poses many restrictions not only the input model topology type but also on the processing algorithm design. The geometry model processing algorithm alters the topology of the input models. Hence, it is possible that even if the input geometry model is manifold the processed model can be non-manifold. The present algorithm can support manifold and non-manifold geometry models. Capability of handling manifold and non-manifold topology during the geometry processing makes the procedure more general and flexible. Mainly, it consists of boundary detection, boundary vertex pair generation, iterative vertex pair contraction and expansion with the following specific steps:

(i) Read and pre-process the input geometry model represented by vertices and indexed faces.
(ii) Detect and mark boundary edges and vertices.
(iii) Build the Octree data structure [24, 25] for efficient searching.
(iv) Generate a list of boundary vertex pairs. For each of the boundary vertices search for other boundary vertices or edges within a user specified resolution tolerance, ε_r, to pair with and insert into the list.
(v) Sort the list of boundary vertex pairs using a cost function that is dependent on the distance between the paired vertices.
(vi) Iteratively remove a boundary vertex pair from the sorted list with minimum cost. Perform the vertex pair contraction operation, if the cost of the vertex pair is less than the user specified glue tolerance, ε_g, otherwise perform the vertex pair expansion operation. Check and avoid adding self-intersecting triangles during the vertex pair expansion operation. Update the connectivity information during the vertex pair contraction and expansion operations.
(vii) Output the processed geometry model with adjacency information.

Detailed description of these steps is presented in the following subsections.

4.1. *Pre-processing*

First, build and pre-process the input geometry model represented by vertices and indexed faces. In this step, initialize the data structure and generate the list of vertices and faces and classify them. At this point the connectivity among the faces is not known. The goal is to find the matching boundary edges and merge them to build the topology information and correct the geometrical issues for the entire model. Now, detect the boundary edges and vertices by finding the number of faces attached to each edge. If an edge has one incident face then it is a boundary edge and incident vertices of a boundary edge are boundary vertices. Geometric entities are created, classified and marked with flags during this step for further processing.

4.2. *Efficient searching*

To build the list of the boundary vertex pairs, for a given boundary vertex, other boundary vertices or edges within the resolution tolerance, ε_r, need to be searched. Hence, efficiency of the algorithm strongly depends on the choice of the data structure used for answering such queries. There are many spatial data structures [24, 25] that can be used for this type of range queries. However, the octree, a simple yet powerful spatial data structure, is used in the present algorithm. A bounding box covering the entire geometry model is the root or parent cell of the octree. This root cell is recursively subdivided into eight children until each of the children contains few geometric objects. The aim is to search for the boundary objects in nearby region. Hence, only boundary objects are inserted into the tree to reduce the amount of data associated with the spatial search. Once the tree data structure is built, finding the geometric objects lying in a given search range is very fast. For a detailed description of the octree data structure classic references [24, 25] can be reviewed.

4.3. Boundary vertex pairs

At this point, all the boundary objects are marked and the octree data structure is built for efficient searching. For each boundary vertex, find other boundary vertices/edges within the resolution tolerance using the octree search. As shown in Figure 1(a), vertex v_i and v_j are paired without splitting the boundary edge for contraction, if $|v_i - v_j| \leqslant \varepsilon_r$. The boundary vertex pair generation procedure strongly depends on the relative position of the boundary vertices to be paired. For example, there is no clear correspondence between boundary vertices v_i and v_j in Figure 1(b). A large ε_r is required to pair them up. But, it is important to note that an appropriate choice of the ε_r is very important. Too small ε_r can leave many potential boundary vertex candidates un-paired. On the other hand, a large ε_r may pair inappropriate boundary vertices and makes the procedure less reliable. Moreover, the vertex pair contraction without edge-split cannot handle the T-joints (end point of one edge lies within another edge) situations. This usually occurs when a big surface is in the neighbourhood of two small surfaces and forms a T-like shape near the junction of surfaces or when two neighbouring curves are discretized using different point distribution functions.

To make the procedure more reliable and handle T-joints, boundary vertex pair contraction with edge-split is introduced. Consider the situation shown in Figure 1(b), there is no other boundary vertex within a smaller ε_r to pair vertex v_i. However, boundary edge $\overline{v_j v_k}$ split operation can create a vertex v_p and boundary vertices v_i and v_p can be paired without increasing ε_r. To find a nearby boundary edge, check if orthogonal projection of vertex v_i on

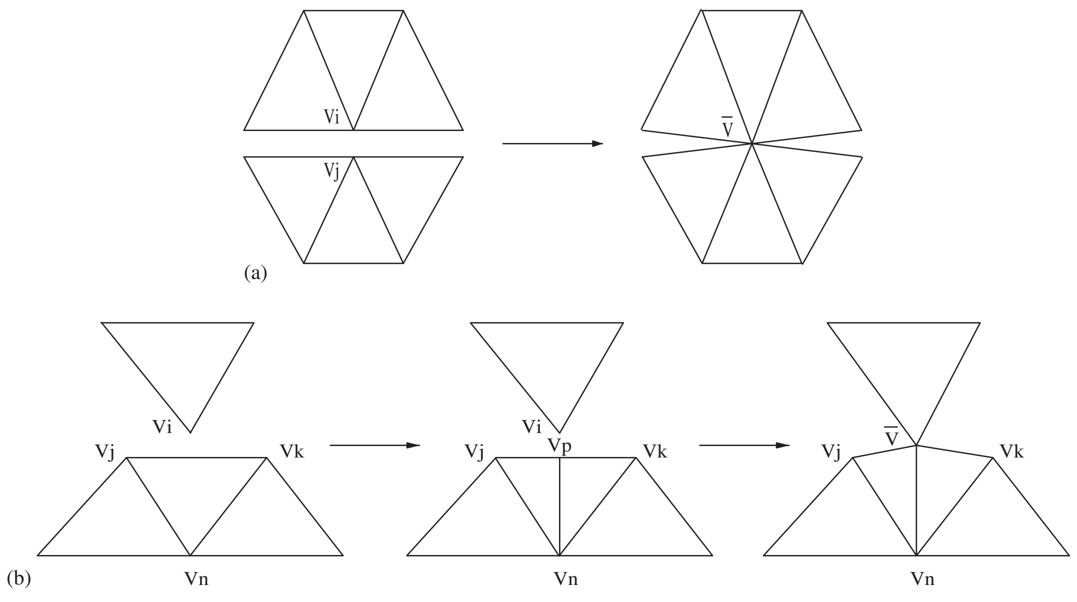

Figure 1. Vertex pair contraction operation: (a) without edge-split; and (b) with edge-split.

a nearby boundary edge $\overline{v_j v_k}$ is possible. To perform this check, let us define $a=\overrightarrow{v_j v_i}$ and $b=\overrightarrow{v_j v_k}$. In theory, orthogonal projection is possible as long as the following inequality is satisfied:

$$0 \leqslant \frac{a \cdot b}{|b|} \leqslant |b| \tag{1}$$

In practice, there may be a problem due to numerical inaccuracies to check the inequality given in Equation (1). To make it numerically more stable and robust, a threshold parameter α can be used and the resultant inequality relation is

$$\alpha \leqslant \frac{a \cdot b}{|b|} \leqslant |b| - \alpha \tag{2}$$

It can be checked using Equation (2) that orthogonal projection of the boundary vertex v_i on the boundary edge $\overline{v_j v_k}$ is possible. To compute the location of projection point v_p the following equation can be used:

$$v_\mathrm{p} = \frac{b}{|b|} d \tag{3}$$

where $d=|v_j - v_\mathrm{p}| = |a| \cos\theta = a \cdot b/|b| =$ projection of a on b, and $\theta = \angle v_i v_j v_k$. Now, split the edge $\overline{v_j v_k}$ with respect to projection point location v_p and pair the boundary vertices v_i and v_p. Edge split operation adds a new vertex v_p and face $\triangle v_\mathrm{p} v_j v_k$. The connectivity information for corresponding vertices and faces is also updated that modifies the topology. It is possible that the projection point v_p lies very close to vertex v_j when $\theta \approx 90^\circ$ and creates a new vertex and face due to edge-split operation. It can be avoided by checking the distance d and reject the edge-split operation, if d is small. The same check can be performed without computing d, the parameter α in Equation (2) can be chosen in such a way that the inequality is not satisfied if the projection vertex v_p and v_j happen to be very close.

In this way, there are two advantages on using the parameter α in Equation (2). First, it takes care of the instabilities due to the numerical inaccuracies and makes the check more stable. Second, it avoids the edge-split operation if the distance d is small without any extra computations. The value of the parameter α can be $0 \leqslant \alpha \leqslant 0.5|b|$. As mentioned earlier, $\alpha = 0$ makes the check numerically unstable and may create projection vertex v_p close to the boundary vertex v_j. While $\alpha = 0.5|b|$ does not allow the edge-split operation in most cases but it may make boundary vertex pair generation procedure less reliable. It is essentially the vertex pair generation without edge-split operation. The present algorithm uses $\alpha = \varepsilon_\mathrm{g}$. A boundary vertex pair of vertices v_i and v_j can be denoted as pair (v_i, v_j). All the boundary vertex pairs generated with and without the edge-split operation are inserted in a heap keyed on cost with the minimum cost pair at the top. The cost function of a boundary vertex pair (v_i, v_j) is $C(v_i, v_j) = |v_i - v_j|$.

4.4. Iterative vertex pair contraction

The topology generation algorithm iteratively removes a vertex pair (v_i, v_j) with minimum cost from the heap and performs the contraction operation, if $C(v_i, v_j) \leqslant \varepsilon_\mathrm{g}$. The boundary vertex pairs are merged to process the geometrical and topological issues. A boundary vertex pair (v_i, v_j) contraction moves boundary vertices v_i and v_j to a new location v_i, v_j or $\bar{v} = (v_i + v_j)/2$.

Merging boundary vertices actually modifies the geometry and processes the geometrical issues such as gaps, intersections, overlaps, etc. The vertex pair contraction operation also replaces faces and edges incident to the v_j with v_i. This step modifies the topology of the model to process the topological issues. The boundary vertex pair contraction operation generally does not collapse faces. However, in case of geometry with long skinny surfaces, the vertex pair contraction operation may produce degenerate faces that are removed from the processed model. Hence, a boundary vertex pair contraction operation merges two vertices and updates the connectivity information. It deletes a boundary vertex and may delete one or more faces. Figures 1(a) and (b) show boundary vertex pair contraction operations with and without edge-split, respectively. The processing algorithm iteratively removes the pair with minimum cost from the heap and performs the vertex pair contraction operation, if the cost of the vertex pair is less than the glue tolerance. This process is called *stitching*. Note that pair vertex contraction moves boundary vertices and modifies the geometry model. An error is introduced in the processed geometry model that is bounded by the resolution tolerance. Consider the effect of the edge-split operation on the error introduced in the processed model. As mentioned earlier, vertex pair contraction without edge-split requires to use a larger resolution tolerance than with edge-split to pair boundary vertices and produces more error. Hence, vertex pair contraction with edge-split operation is not only more reliable and robust but also more accurate.

4.5. Iterative vertex pair expansion

The vertex pair contraction operation with edge-split introduces less error in the processed geometry model than that of without edge-split. It moves the vertices physically in order to process the errors smaller than the glue tolerance. Hence, if the errors are large in the model to be processed, then the vertex pair contraction operation would require to choose a larger glue tolerance and produce more error in the processed model. A new operation, vertex pair expansion, is developed to fill the larger gaps with new triangles without moving the vertices as opposed to the vertex pair contraction operation. It fills the gaps larger than the glue tolerance and smaller than the resolution tolerance with new triangles without introducing any error in the processed model.

The topology generation algorithm removes a vertex pair (v_i, v_j) with minimum cost from the heap and performs the expansion operation, if $\varepsilon_g < C(v_i, v_j) \leqslant \varepsilon_r$. The boundary vertex pair (v_i, v_j) expansion adds one or more new triangles to fill the gap. Addition of new triangles actually modifies the geometry and processes the geometrical issues like gaps. The pair expansion also updates the list of incident faces to the vertices. This step modifies the topology of the model to process the topological issues. Hence, the boundary vertex pair expansion operation adds new faces and updates the topology information. Figure 2(a) shows the vertex pair (v_i, v_j) expansion operation without edge-split. It adds two new triangles $\triangle v_i v_j v_k$ and $\triangle v_i v_j v_n$. Figure 2(b) shows the vertex pair expansion with edge-split. It also adds two new triangles $\triangle v_i v_p v_k$ and $\triangle v_i v_p v_j$. The vertex pair expansion may also add one triangle near already stitched or junction of surfaces that will be discussed later in detail. This iterative vertex pair expansion process is called *filling*. Note that *filling* does not move the vertices like *stitching* operation to process the model. Therefore, it does not introduce any error in the processed model.

Some of the references [17, 26] reported that the resultant boundary may be self-intersecting as a limitation of their algorithm, if the downstream application requires the output to be free

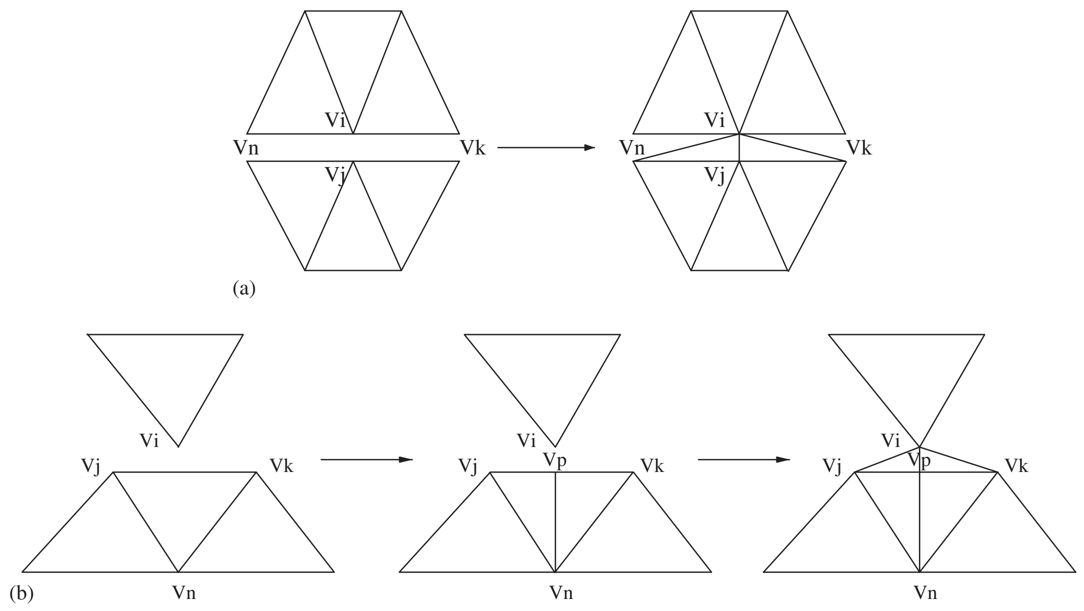

Figure 2. Vertex pair expansion operation: (a) without edge-split; and (b) with edge-split.

of this phenomenon. For example, rapid prototyping is another application where the CAD models are expected to be free of self-intersecting triangles. The present algorithm allows the user to decide whether the output model can have self-intersecting boundaries or not. The *filling* operation may produce self-intersecting faces in the boundary regions. It can be avoided by checking the triangle–triangle overlap test among the triangles incident to the vertices of a boundary vertex pair (v_i, v_j). Meaning that each triangle incident to vertex v_i is checked for intersection with the triangles incident to vertex v_j. If there exists an intersection between any of two triangles incident to vertices v_i and v_j, then perform the vertex pair contraction instead of the vertex pair expansion during the *filling* operation. It avoids adding new self-intersecting triangles in the boundary and close the intersection/overlap via *stitching* operation. There are many triangle–triangle intersection/overlap tests [27–29] developed by the real-time rendering community. The triangle–triangle overlap test using orientation predicates [27] is fast and robust, a significant improvement over other methods [28, 29], technique. The present algorithm uses the same method to test the triangle–triangle overlap and it is guaranteed that the output model is free of self-intersecting boundaries. The extra cost of checking triangle–triangle intersection test can degrade the performance of the algorithm, if the number of triangles need to be tested are large. The topology generation algorithm *stitches* the gaps/overlaps smaller than the glue tolerance and *fills* the gaps with new triangles. For most of the realistic cases, it is found that the boundary vertex pairs left in the heap after the *stitching* operation are very few as compared to the total number of pairs. The triangle–triangle overlap test is performed, only for the boundary vertex pairs left after *stitching*, during the *filling* operation. Hence, the extra computation of the triangle–triangle intersection test does not affect the performance of the algorithm much.

In summary, the output of the algorithm is a well-suited discrete CAD model along with the necessary topology information that can be an input for many downstream applications. In this mode the output mesh can be used as a background mesh to subsequently generate a high-quality unstructured mesh using Advancing Front Local Reconnection (AFLR) [30] algorithm.

5. RESULTS AND DISCUSSIONS

The topology generation algorithm following the outline given in the previous section is developed and implemented. Consider a simple plate as shown in Figure 3(a). There is a small gap and overlap between two triangulated surfaces that needs to be processed. Figure 3(b) shows the extracted boundary edges and boundary vertex pairs for further processing. Note that two neighbouring boundary curves are discretized with different point distributions. There is no clear correspondence among the boundary vertices for vertex pair generation. Hence, it is required to split some of the boundary edges to pair vertices more accurately and reliably. All these boundary vertex pairs are contracted iteratively using the vertex pair contraction operation that stitches the gaps and overlap. The resultant geometry after the *stitching* operation is shown in Figure 3(c). Note that the *stitching* operation moves vertices physically by changing their coordinates and introduces an error of the order of the resolution tolerance. It does not add any new triangles to repair the gap/overlap region. Finally, it can be checked whether the geometry model is processed by again extracting the boundary edges of the modified geometry. Figure 3(d) shows that there is no gap and overlap between two surfaces in the processed geometry. Moreover, the topology (connectivity map) information is available and can be used for downstream applications like mesh generation.

The *stitching* operation modifies the original model to process the geometrical and topological issues. The same plate model, as shown in Figure 4(a), can also be processed via *filling* operation. Figure 4(b) shows the boundary edges and boundary vertex pairs. Figure 4(c) shows the processed geometry model. Unlike the *stitching* operation, the *filling* does not move vertices and modify the original model. The original model can be processed without introducing any error but it adds many self-intersecting and skinny triangles in the boundary region of the processed model. Notice the difference between the processed models shown in Figures 3(c) and 4(c) using the *stitching* and *filling* operations, respectively.

The *stitching* operation merges boundary vertices to process the geometry and produces more error than the *filling* operation. However, if only *filling* operation is used then it may introduce many new triangles in the boundary regions for large models. Hence, to minimize the error in the processed model and reduce adding too many triangles, the *stitching* and *filling* operations are used together. Figures 5(a) and (b) show the same plate model and the boundary edges and vertex pairs, respectively. Figure 5(c) shows the processed model. Notice that small gaps/overlaps are merged together and large gaps are filled with new triangles. In case of large overlap, the *filling* operation either adds self-intersecting triangles in the boundary region or performs the vertex pair contraction to avoid adding self-intersecting triangles as specified by the user. In this way, combined use of the *stitching* and *filling* operations makes the geometry processing more accurate and practical.

Figure 6(a) shows two cubes sharing a common curve with cracks and overlaps between surfaces. The common curve is shared by four faces and it is required to provide an explicit

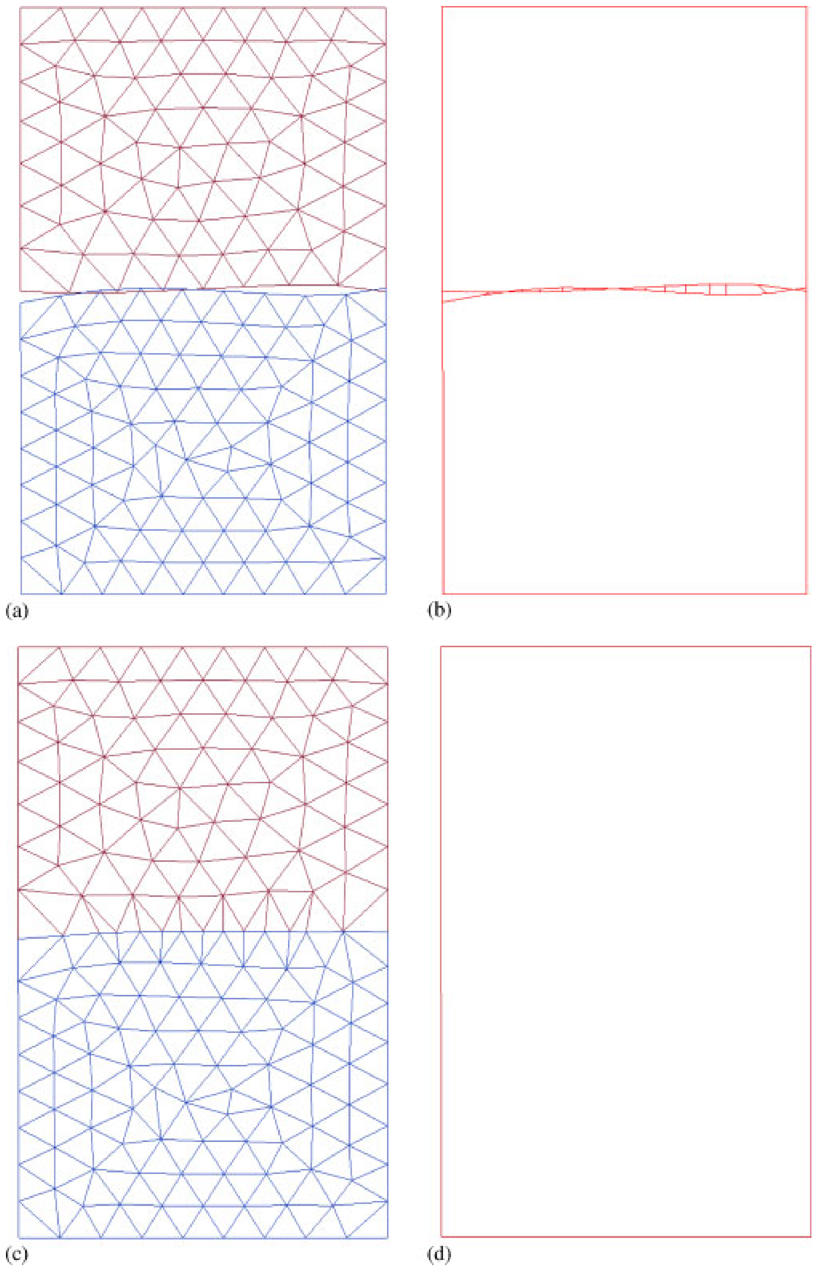

Figure 3. Plate showing the stitching and filling operation: (a) original geometry; (b) boundary edges and vertex pairs; (c) processed geometry; and (d) boundary edges after processing.

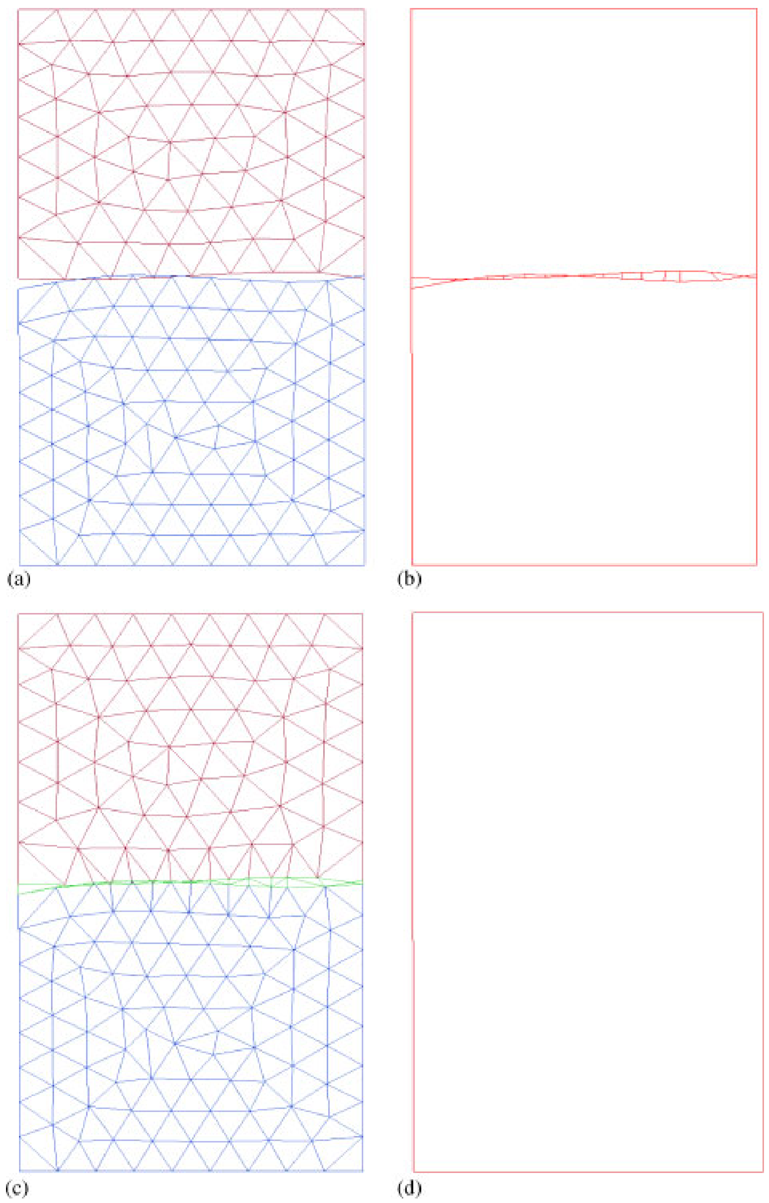

Figure 4. Plate showing the filling operation: (a) original geometry; (b) boundary edges and vertex pairs; (c) processed geometry; and (d) boundary edges after processing.

support for non-manifold topology to process this model. As mentioned earlier the topology generation algorithm can handle non-manifold situations that provides more flexibility and generality. Figure 6(b) shows the geometrically and topologically well-defined model obtained after processing. Note that the topology generation algorithm locally modifies the

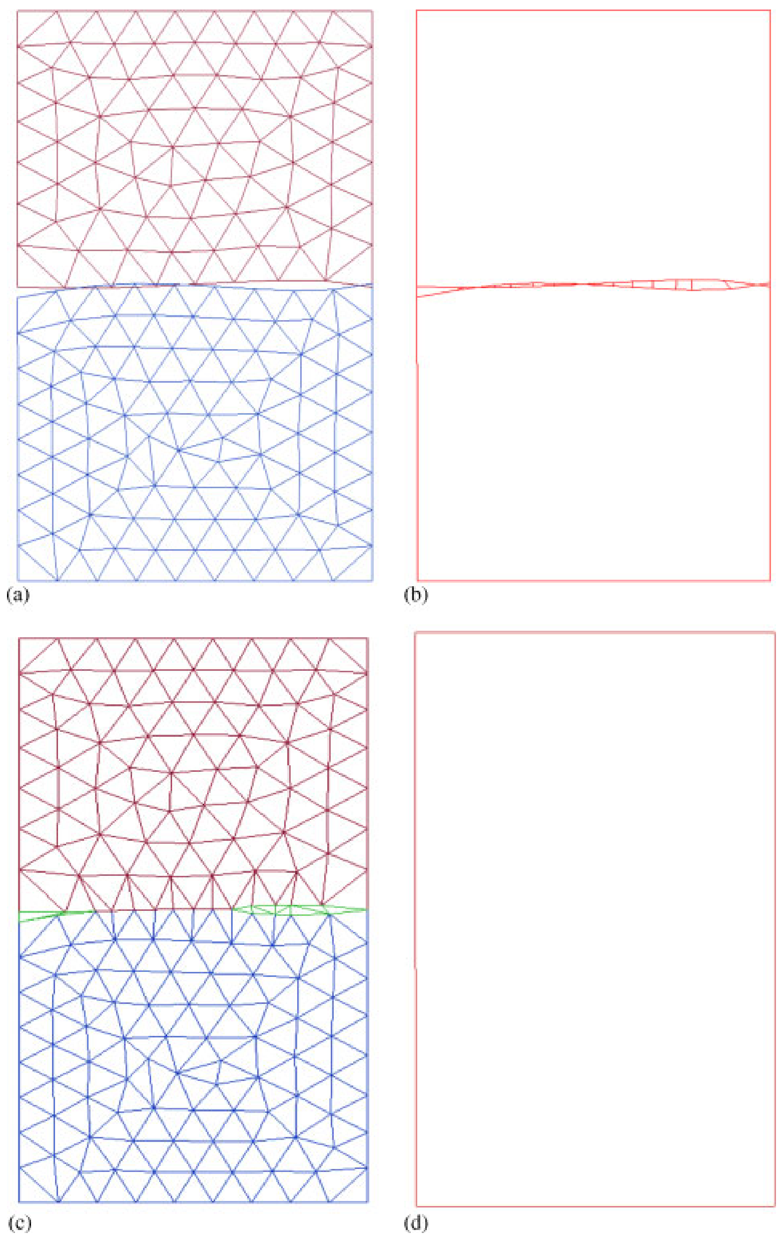

Figure 5. Plate showing the stitching and filling operations: (a) original geometry; (b) boundary edges and vertex pairs; (c) processed geometry; and (d) boundary edges after processing.

topology of the input model. Hence, it is possible that even if the input geometry model is manifold the output or intermediate model may be non-manifold. The data structure used in the implementation of present algorithm can support manifold and non-manifold topology.

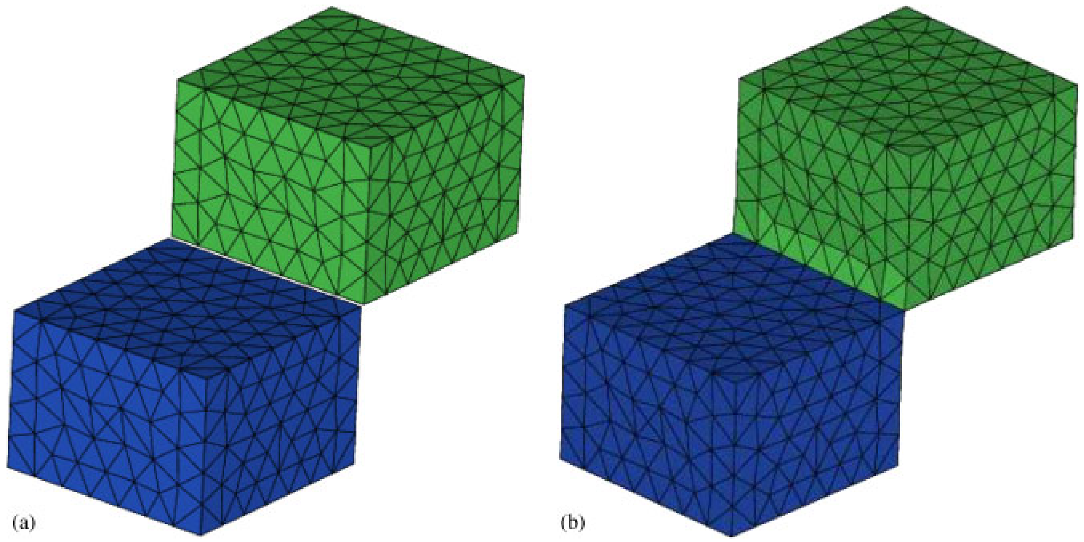

Figure 6. Two cubes sharing a common edge (non-manifold topology): (a) original model; and (b) processed model.

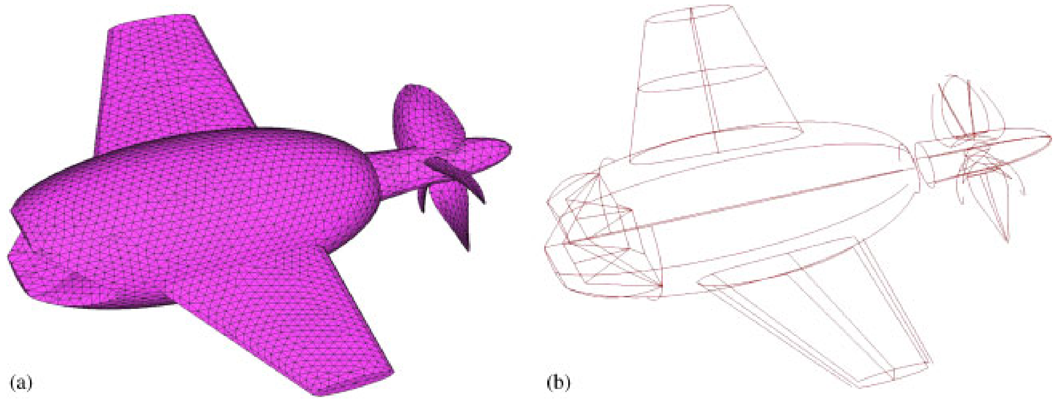

Figure 7. Flying minnow (original model): (a) original geometry; and (b) boundary edges before stitching.

Figure 7(a) shows the triangulated model of a flying minnow. Figure 7(b) shows the detected boundary edges of the same model. Figure 8(b) shows the remaining gaps after the *stitching* operation. These gaps are then triangulated via the *filling* operation. The flying minnow model consists of 5566 vertices and 9062 faces. The resolution tolerance, ε_r, and the glue tolerance, ε_g, used are 0.1 and 0.01, respectively. The boundary vertex pair generation process has formed 1162 vertex pairs, of which about 1127 pairs are contracted and only 35 pairs are expanded during the *stitching* and *filling* operations, respectively. The *filling* operation

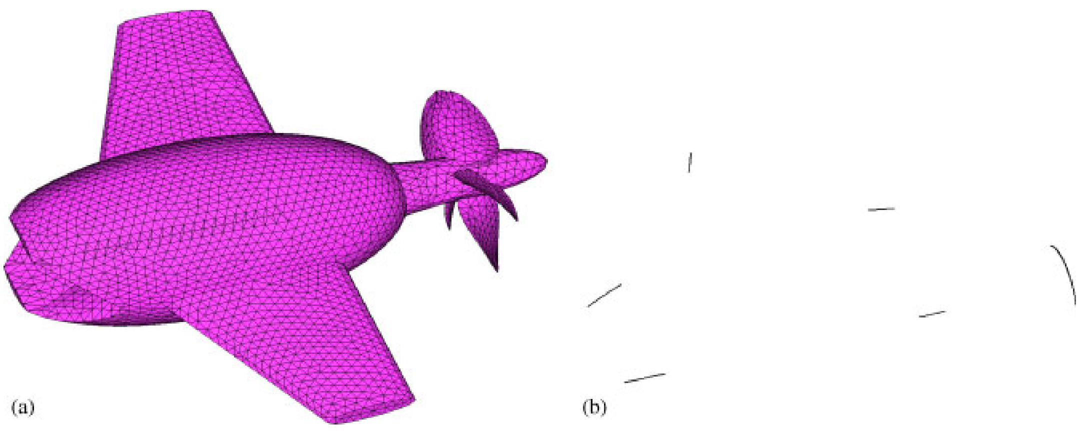

Figure 8. Flying minnow (processed model): (a) processed geometry; and (b) gaps filled with new triangles.

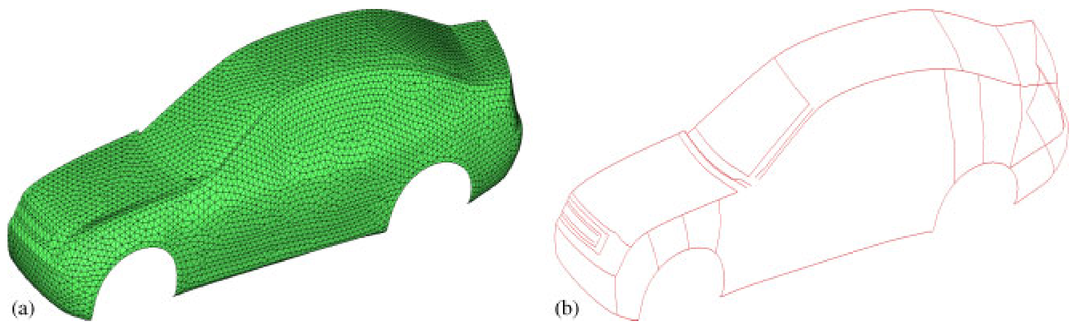

Figure 9. Body shell of Infiniti G35 (original model): (a) original geometry; and (b) boundary edges before stitching.

may introduce self-intersecting triangles in the boundary regions. Hence, if it specified by the user that output should be free of such phenomenon then the triangle–triangle overlap test is performed only for the triangles incident to the vertices that form 35 pairs. It is evident that boundary vertex pairs left in the heap after *stitching* are usually much less than the total number of pairs. Therefore, the extra cost of performing triangle–triangle overlap test would not degrade performance of the algorithm much. Boundary edge detection on the processed flying minnow model found no edge because it forms a closed volume and each of the edges in the model is two-manifold.

Figure 9(a) shows the body shell of Infiniti G35 car model. Figure 9(b) shows the boundary edges before processing. This model has 4283 vertices and 7448 faces. Figure 10(a) is the processed vehicle model after the *stitching* and *filling* process. To verify the repairing process, boundary edges of the modified geometry are extracted as shown in Figure 10(b). It shows the boundary curves that are shared by only one surface. Moreover, the topology information

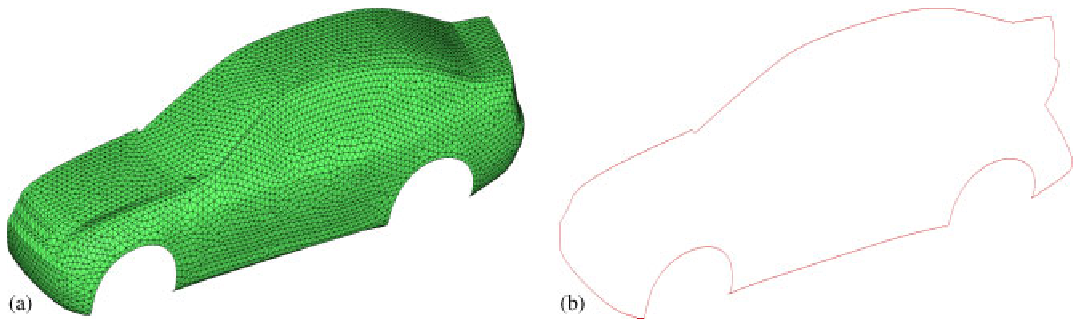

Figure 10. Body shell of Infiniti G35 (processed model): (a) processed geometry; and (b) boundary edges after processing.

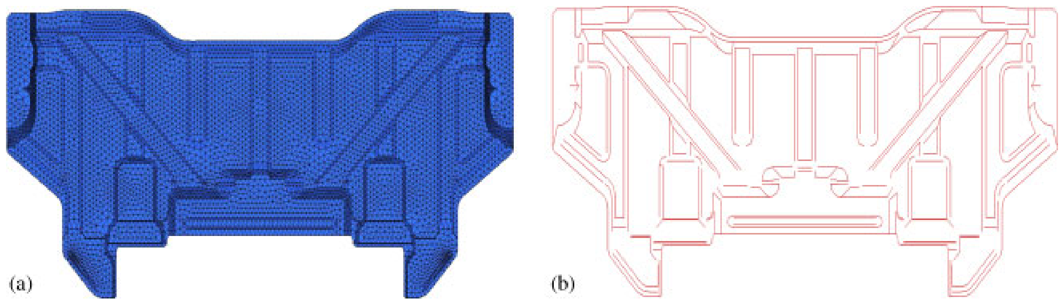

Figure 11. Front part of under-body floor of Infiniti G35 (original model): (a) original model; and (b) boundary edges and vertex pairs.

is also built during the stitching process. The total number of boundary vertex pairs generated was 420 for the value of ε_r and ε_g to be 1.0 and 0.01, respectively.

Figure 11(a) shows the front part of Infiniti G35 under-body floor. It has 7784 vertices and 11 633 faces. Boundary edges and vertex pairs of original model are shown in Figure 11(b). The number of boundary vertex pairs formed is 1844. Model topology is built using the topology generation algorithm. Figure 12(a) shows the processed model with topology information. The resolution tolerance and glue tolerance used for the model processing are 0.07. Figure 12(b) shows the boundary edges of the processed model. Note that only the true boundary of the model is not connected to other surfaces. All the interior surfaces have correct connectivity information.

The discrete geometry of Infiniti G35 doors is shown in Figure 13(a). The model consists of 6443 vertices and 10 814 faces. It has a large gap near the upper-right corner due to a missing surface. Figure 13(b) shows the enlarged view of the gap that has to be processed. It is important to note that the size of the gaps and some of the surfaces in the model are of the same order. Figures 14(a) and (b) show the processed model and enlarged view of the rectangular region. The large gap is filled with new triangles via the *filling* operation. The topology generation algorithm performed about 827 vertex pair contractions and 92 vertex pair

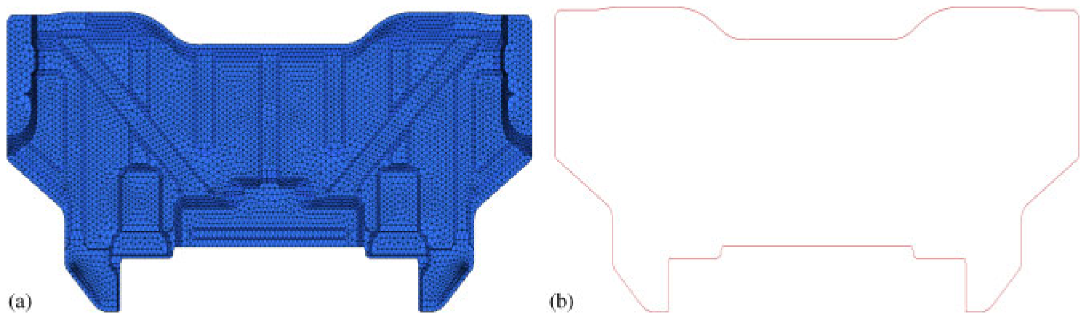
Figure 12. Front part of under-body floor of Infiniti G35 (processed model): (a) processed model; and (b) boundary edges and vertex pairs.

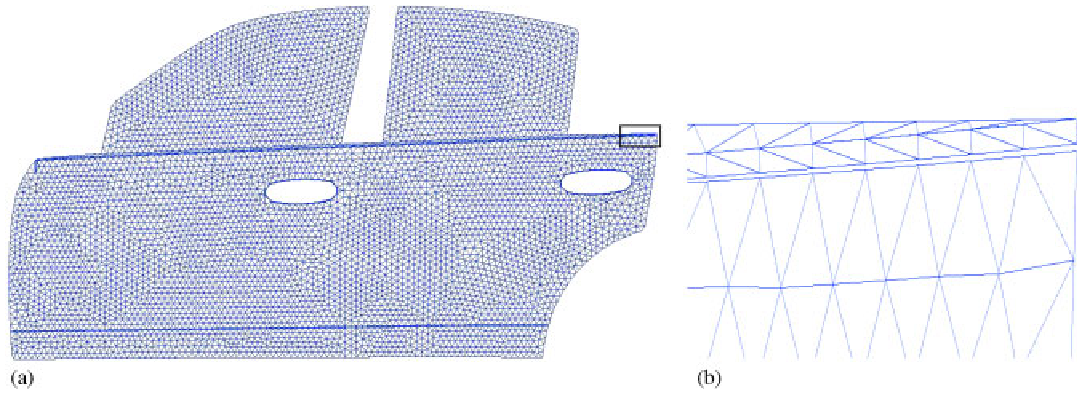
Figure 13. Doors and windows of Infiniti G35: (a) original model; and (b) enlarged view of the rectangular region showing the gap due to a missing surface.

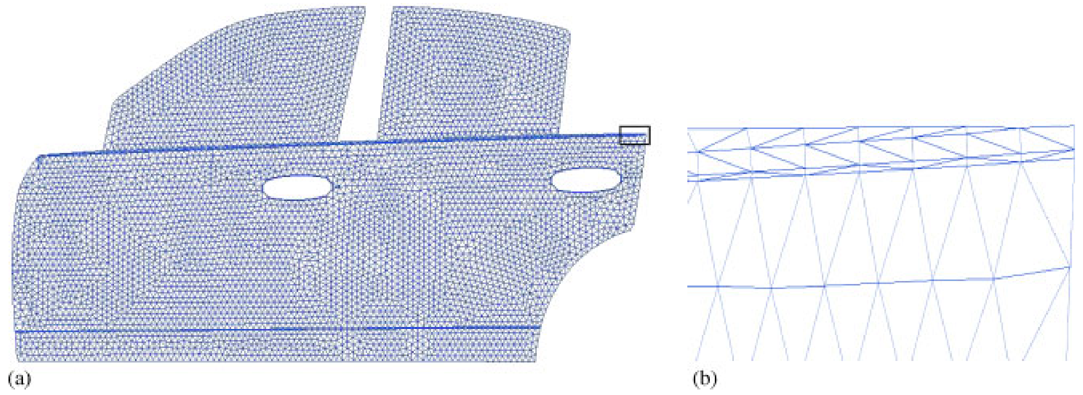
Figure 14. Doors and windows of Infiniti G35: (a) processed model; and (b) enlarged view of the rectangular region shows that the gap is filled with triangles in the processed model.

expansions in order to process the model with ε_r and ε_g of 2.25 and 0.1 mm, respectively. This test case shows that the topology generation algorithm can even create small missing geometry entities up to some extent. For example, it did add new faces in place of a small missing surface near the upper-right corner of the door.

6. CONCLUDING REMARKS

Automatic detection and processing of commonly found geometrical and topological issues such as gaps, overlaps, intersections, T-connections, invalid or no topology, etc. is achieved for two- and three-dimensional configurations. Unlike a CAD model repair procedure that requires significant user interaction, the proposed methodology is highly automated. The procedure can successfully process manifold and non-manifold geometry models according to the needs of many downstream applications. Preliminary results show that the algorithm can process the geometrical and topological flaws automatically and efficiently. This work is a step towards automatic geometry processing for mesh generation applications. There are many other issues that need to be addressed to automate the pre-processing of geometry for the same application. The algorithm should handle large gaps/overlaps among the surfaces in CAD models. Moreover, the present repair procedure introduces some error, which is bounded by a user specified distance threshold, in the processed geometry model that must be as small as possible. Hence, relatively large gaps should be filled with new triangles to avoid significant modification of the original model. Moreover, the user has to provide an appropriate value of distance threshold. It may require the user to try different threshold values until a reasonable value is found. It is found that small variation in distance threshold does not affect the results much in most cases. However, selection of the distance threshold should be automatic and adaptive. Currently, work is in progress to address these issues.

REFERENCES

1. Butlin G, Stops C. CAD data repair. *Proceedings of 5th International Meshing Roundtable*, Sandia National Laboratories, 1996; 7–12.
2. Jones MR, Butlin PG. Geometry management support for auto-meshing. *Proceedings of 4th International Meshing Roundtable*, Sandia National Laboratories, 1995; 153–164.
3. Mezentsev AA, Woehler T. Methods and algorithms of automated CAD repair for incremental surface meshing. *Proceedings of 8th International Meshing Roundtable*, Sandia National Laboratories, 1999; 299–309.
4. Steinbrenner JP, Wyman NJ, Chawner JR. Fast surface meshing on imperfect CAD models. *Proceedings of 9th International Meshing Roundtable*, Sandia National Laboratories, 2000; 33–41.
5. Aftosmis MJ. Solution adaptive Cartesian grid methods for aerodynamics flows with complex geometries. *28th Computational Fluid Dynamics*, von Korman Institute of Fluid Dynamics, Lecture Series 1997–2002, 1997.
6. Hu J, Lee YK, Blacker T, Zhu J. Overlay grid based geometry cleanup. *Proceedings of 11th International Meshing Roundtable*, Sandia National Laboratories, 2002; 313–324.
7. Wang ZJ, Srinivasan K. An adaptive Cartesian grid generation method for 'Dirty' geometries. *International Journal for Numerical Methods in Fluids* 2002; **39**:703–717.
8. Baum DR, Mann S, Smith KP, Winget JM. Making radiosity usable: automatic preprocessing and mesh techniques for the generation of accurate radiosity solutions. *Proceedings of ACM SIGGRAPH*, *Volume 25 of Computer Graphics*, New York, 1991; 51–60.
9. Foley JD, Dam A, Feiner SK, Hudges JF. *Computer Graphics*: *Principles and Practice* (2nd edn).
10. Murali TM, Funkhouser TA. Consistent solid and boundary representation from arbitrary polygonal data. *Proceedings of the Symposium On Interactive 3D Graphics*, Providence, RI, 1997; 155–162.
11. Gueziec A, Taubin G, Lazarus F, Horn B. Cutting and stitching: converting sets of polygons to manifold surfaces. *IEEE Transactions on Computer Graphics* 2001; **7**(2):136–151.

12. Rock SJ, Wozny MJ. Generating topological information from a Bucket of Facets. In *Proceedings of the Solid Freeform Fabrication Symposium*, Marcus HL *et al.* (eds). The University of Texas at Austin: TX, 1992; 251–259.
13. Bohn JH, Wozny MJ. Automatic CAD-model repair: shell-closure. In *Proceedings of the Solid Freeform Fabrication Symposium*, Marcus HL *et al.* (eds). The University of Texas at Austin: TX, 1992; 86–94.
14. Makela I, Dolenc A. Some efficient procedures for correcting triangulated models. In *Proceedings of the Solid Freeform Fabrication Symposium*, Marcus HL *et al.* (eds). The University of Texas at Austin: TX, 1993; 126–134.
15. Sheng X, Meier IR. Generating topological structures for surface models. *IEEE Computer Graphics and Applications* 1995; **15**(6):35–41.
16. Morvan SM, Fadel GM. IVECS, interactively correcting STL files in a virtual environment. In *Proceedings of the Solid Freeform Fabrication Symposium*, Marcus HL *et al.* (eds). The University of Texas at Austin: TX, 1996.
17. Barequet G, Sharir M. Filling gaps in the boundary of a polyhedron. *Computer Aided Design* 1995; **12**(2): 207–229.
18. Barequet G, Kumar S. Repairing CAD models. *Proceedings of the IEEE Visualization*, Phoenix, AZ, 1997; 363–370.
19. Kalvin A, Schonberg E, Schwartz JT, Sharir M. Two-dimensional model-based, boundary matching using footprints. *International Journal of Robotics Research* 1986; **5**(4):38–55.
20. Schwartz JT, Sharir M. Identification of partially obscured objects in two and three dimensions by matching noisy characteristics curves. *International Journal of Robotics Research* 1987; **6**(2):29–44.
21. Thompson JF, Soni BK, Weatherill NP. *Handbook of Grid Generation*. CRC Press: Boca Raton, FL, 1998.
22. Garland M, Heckbert PS. Surface simplification using quadratic error metrics. *ACM SIGGRAPH Computer Graphics Proceedings*, 1997; 209–216.
23. Popovic J, Hoppe H. Progressive simplicial complexes. *ACM SIGGRAPH Computer Graphics Proceedings*, 1997; 217–224.
24. Samet H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley: Reading, MA, ISBN 0-201-50255-0, 1990.
25. Samet H. *Applications of Spatial Data Structures*: *Computer Graphics*, *Image Processing*, *and GIS*. Addison-Wesley: Reading, MA, ISBN 0-201-50300, 1990.
26. Patel PS, Marcum DL, Remotigue MG. Stitching and filling: creating conformal faceted geometry. *Proceedings of 14th International Meshing Roundtable*, Sandia National Laboratories, 2005.
27. Guigue P, Devillers O. Fast and robust triangle–triangle overlap test using orientation predicates. *Journal of Graphics Tools* 2003; **8**(1):39–52.
28. Moller T. A fast triangle–triangle intersection test. *Journal of Graphics Tools* 1997; **2**(2):25–30.
29. Held M. ERIT—a collection of efficient and reliable intersection tests. *Journal of Graphics Tools* 1996; **2**(4): 25–44.
30. Marcum DL. Efficient generation of high quality unstructured surface and volume grids. *Proceedings of 9th International Meshing Roundtable*, Sandia National Laboratories, 2000.