
Stitching and Filling: Creating Conformal Faceted Geometry

Paresh S. Patel¹, David L. Marcum², and Michael G. Remotigue³

¹ Computational Simulation and Design Center, ERC, Mississippi State University, Mississippi State, MS 39762, U.S.A. patel@erc.msstate.edu

² Computational Simulation and Design Center, ERC, Mississippi State University, Mississippi State, MS 39762, U.S.A. marcum@erc.msstate.edu

³ Computational Simulation and Design Center, ERC, Mississippi State University, Mississippi State, MS 39762, U.S.A. remo@erc.msstate.edu

Summary. Consistent and accurate representation of geometry is required by a number of applications such as mesh generation, rapid prototyping, manufacturing, and computer graphics. Unfortunately, faceted Computer Aided Design (CAD) models received by downstream applications have many issues that pose problems for their successful usability. Automatic or semi-automatic tools are needed to process the geometry to make it suitable for these downstream applications. An algorithm is presented to detect commonly found geometrical and topological issues in the faceted geometry and process them with minimum user interaction. The present algorithm is based on the iterative vertex pair contraction and expansion operations called *stitching* and *filling* respectively. 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.

1 Introduction

Computational design, analysis, optimization, and manufacturing have become an integral part of the product development process in automotive, aerospace, electronics, and many other industries. The simulation-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, rapid prototyping, numerical controlled machining, casting, computer graphics, 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.

The Computational Fluid Dynamics (CFD) simulation process has several stages including pre-processing, flow solution, and post-processing of the results. Typically, pre-processing involves 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 data translated through neutral file formats like IGES [IGE88] and STL [STL89] 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, inconsistent orientations. 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 labor-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 algorithm closes small gaps/overlaps via the *stitching* operation and fills big gaps by adding new faces through the *filling* operation to process the model accurately. 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 neighbor finding to process large models efficiently.

2 Related Work

Adaptive Cartesian grid generation method [Aft97] based approaches [HLBZ02, WS02] can be used for CAD cleanup and surface triangulation. Hu et al. [HLBZ02] has 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 et al. [WS02] 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 to get the intersection/projection points to reconstruct the surface from these points. Further improvement of these approaches [HLBZ02, WS02] can be achieved by targeting only the bad areas with gaps and overlaps. 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 in some part of the geometry, these approaches rebuild the entire model approximately, and accuracy depends on the refined Cartesian mesh cell 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. [BMSW91] developed a series of algorithms to preprocess the input geometry to meet the requirements of mesh based radiosity computation algorithms. Murali et al. [MF97] described an algorithm based on space subdivision to construct a consistent solid and boundary representation from polygons. This technique is simple and works well in the absence of degeneracies and narrow angles among neighboring polygons. It also requires significant

amount of time to process even small models with few hundred polygons. Gueziec et al. [GTLH01] developed greedy strategies to convert a set of non-manifold polygonal surfaces 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. Rock and Wozny [RW92] have used an AVL tree data structure to locate neighbor vertices efficiently to build model topology from a given set of unordered triangular facets. Bohn and Wozny [BW92] 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 [MD93] developed methods to handle overlapping and intersecting triangles efficiently. Sheng and Meier [SM95] have used a technique based on incremental matching and merging of boundaries of surface models to repair gaps. Their technique merges small as well as large gaps to process the models. Hence, it is not accurate to process large gaps due to missing geometry or polygons. Morvan and Fadel [MF96b] developed a virtual environment to correct the errors in a given model interactively, which is very time consuming and expensive to process large models. Barequet et al. [BK97, BS95] used a computer vision technique called geometric hashing [KSS86, SS87] to repair geometrical and topological errors in the boundary representation (b-rep) of two-manifold geometry models. Recently, Patel et al. [PMR05] developed a technique based on a modified iterative vertex pair contraction operation called *stitching* to build topology information for manifold and non-manifold models. This work is an extension of the topology generation algorithm [PMR05] to process geometry models more accurately.

It seems previous mesh-based efforts [BK97, BS95, BW92, MD93, MF96b, RW92, SM95] 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 explicitly supports manifold and non-manifold geometry models. Capability of handling manifold and non-manifold topologies during the geometry processing makes the procedure more general and flexible. Some of the previous approaches [BK97, BS95, BW92] collected the Jordan curves that are non-intersecting three-dimensional closed polygonals. These polygons are then triangulated to fill the gaps and holes. These approaches assume that boundary edges form a closed polygonal loop. However, it is possible that a set of boundary edges may not form Jordan curves and produce a valid triangulation of holes or require some user interaction in these situations. The *filling* process of the present algorithm uses a different approach to handle such situations. It does not require to form Jordan curves to fill gaps with new triangles. Moreover, the present algorithm stitches small gaps/overlaps and fills big gaps by adding new triangles to process the model accurately. In addition, it uses the octree data structure for searching and neighbor finding to process large models efficiently. In this way, the present procedure offers

combined benefits of generality, accuracy and efficiency for automatic processing of faceted geometry.

3 Geometry and Topology Representation

CAD models are often represented as a set of triangulated surfaces in three-dimensional Euclidean space R^3 [J. 98]. 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 $V = (v_1, v_2, \dots, v_m)$ 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 $F = (f_1, f_2, \dots, f_n)$ 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 $\Delta v_j v_k v_l$. 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. The goal is to develop an algorithm to process geometrical and topological issues and build the topology information (neighbor maps) with minimum user interaction.

4 The Topology Generation Algorithm

The present algorithm is based on the iterative vertex pair contraction and expansion operations to process the geometrical and topological issues. An edge-split operation is introduced to make vertex pair contraction [GH97, PH97] more reliable and accurate. 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 [H. 90a, H. 90b] 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. 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 sections.

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 Spatial Data Structure

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 [H. 90a, H. 90b] 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 a bit of old but classic references [H. 90a, H. 90b] can be reviewed.

4.3 Boundary Detection and 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 $|v_i - v_j| \leq \epsilon_r$. 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 can not handle the T-joints (end point of one edge lies within another edge) situations. This usually occurs when a big surface is in the neighborhood of two small surfaces and forms a T-like shape near the junction of surfaces or when two neighboring 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 a nearby boundary edge $\overline{v_j v_k}$ is possible. To perform this check, let's define $a = \vec{v_j v_i}$ and $b = \vec{v_j v_k}$. In theory, orthogonal projection is possible as long as the following inequality is satisfied.

$$0 \leq \frac{a \cdot b}{|b|} \leq |b| \quad (1)$$

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

$$\alpha \leq \frac{a \cdot b}{|b|} \leq |b| - \alpha \quad (2)$$

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,

$$v_p = \frac{b}{|b|} d \quad (3)$$

where $d = |v_j - v_p| = |a| \cos \theta = a \cdot b / |b| =$ 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 $\triangle v_p v_j v_k$. 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 $0 \leq \alpha \leq 0.5 |b|$. 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 $\alpha = \epsilon_g$. 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 $C(v_i, v_j) \leq$

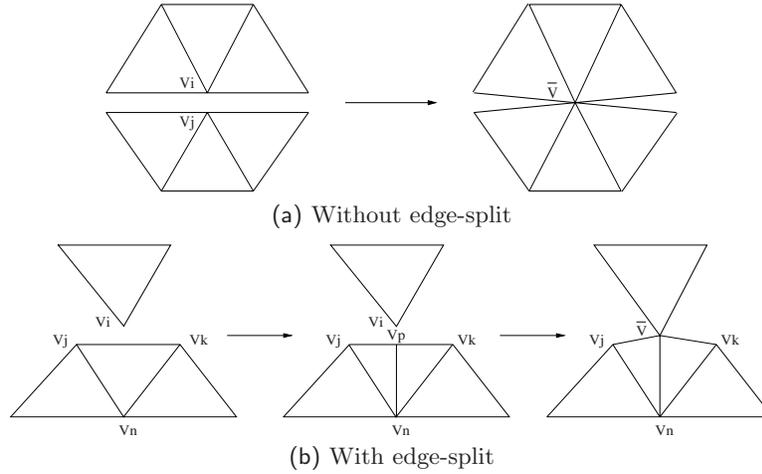


Fig. 1. Vertex pair contraction operation

ϵ_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. Figure 1(a) and 1(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 boundary vertex pairs near the junction of surfaces are penalized by setting their cost function to be zero. In this way, the boundary vertex pairs are forced to be contracted to join disjoint surfaces.

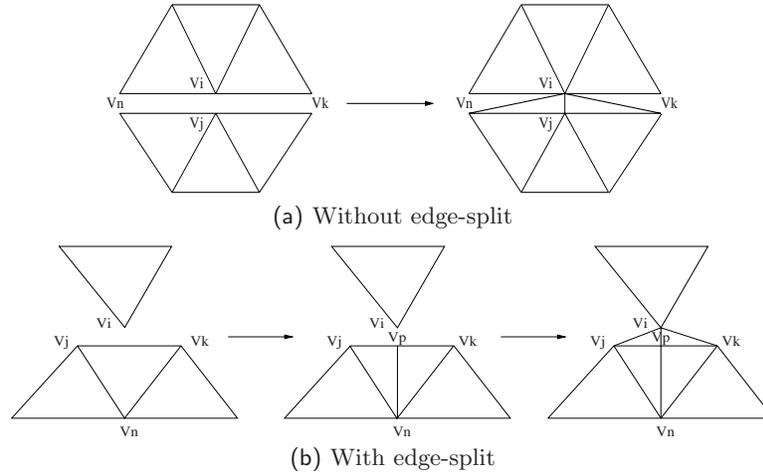


Fig. 2. Vertex pair expansion operation

The topology generation algorithm removes a vertex pair (v_i, v_j) with minimum cost from the heap and performs the expansion operation, if $\epsilon_g < C(v_i, v_j) \leq \epsilon_r$. 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 $\Delta v_i v_j v_k$ and $\Delta v_i v_j v_n$. Figure 2(b) shows the vertex pair expansion with edge-split. It also adds two new triangles $\Delta v_i v_p v_k$ and $\Delta v_i v_p v_j$. As mentioned earlier, if the value of the resolution tolerance is big-enough, then the vertex pair expansion would add a new triangle $\Delta v_i v_j v_k$ without splitting the edge $\overline{v_j v_k}$. The vertex pair expansion may also add one triangle near already stitched or junction of surfaces that will be discussed later in details. 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.

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) [MW95] algorithm. The current procedure is based on an assumption of the locality of geometrical and topological issues. It is assumed that the neighboring surface patches have small gaps/overlaps due to the translation errors, numerical inaccuracies, tolerance settings in different systems, etc. 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.

5 Results and Discussion

The topology generation algorithm following the outline given in the previous section is developed and implemented using object oriented technology in C++ on a unix workstation. CAD models obtained from various systems are processed to build correct topology information. 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. 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. 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.

The *stitching* operation modifies the original model to process the geometrical and topological issues. The same plate model can also be processed via *filling* operation. Figure 4 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.

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. To minimize the error in the processed model and reduce adding too many triangles, the *stitching* and *filling* operations are used together as shown in figure 5. Notice that small gaps/overlaps are merged together and large gaps are filled with new triangles. 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 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

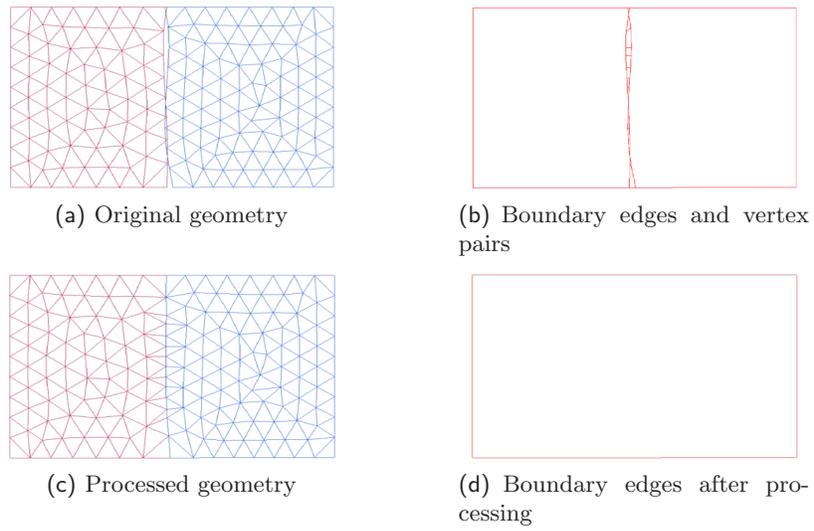


Fig. 3. Plate showing the stitching operation

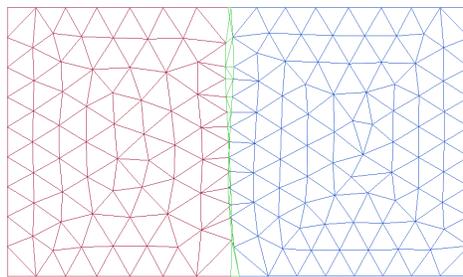


Fig. 4. Plate showing the filling operation

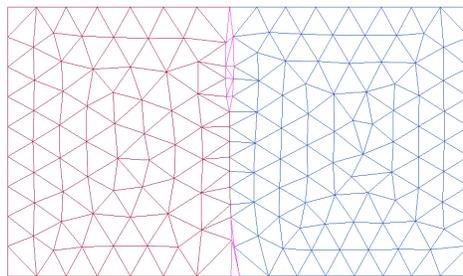


Fig. 5. Plate showing the stitching and filling operations

and topologically well-defined model obtained after processing. Note that the topology generation algorithm locally modifies the 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.

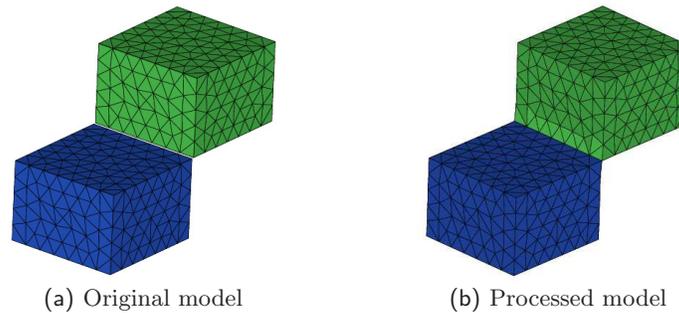
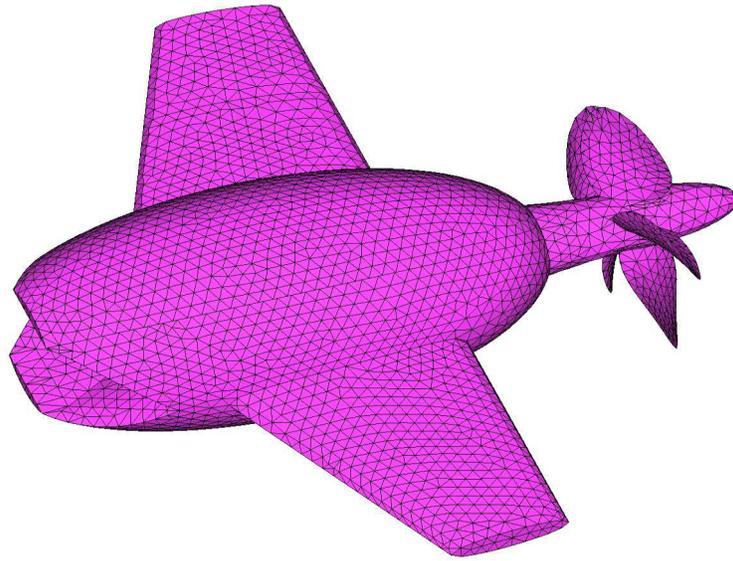


Fig. 6. Two cubes sharing a common edge (non-manifold topology)

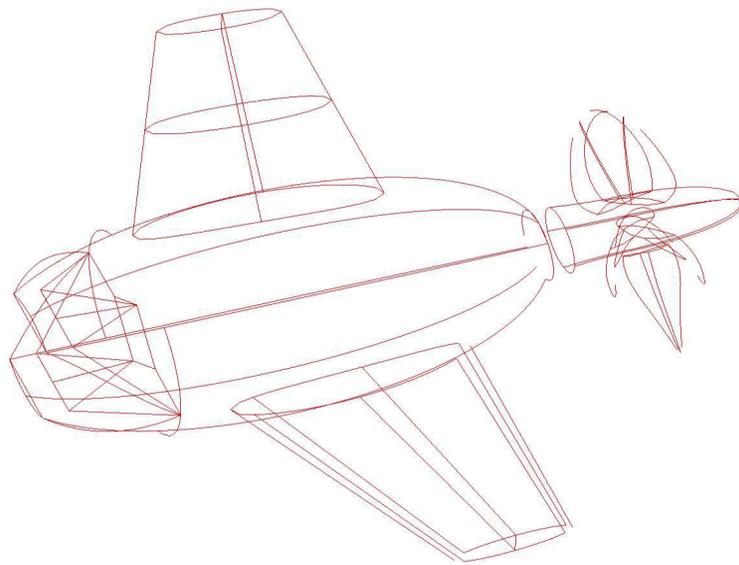
Figure 7(a) shows the triangulated model of a flying minnow. Figure 7(b) shows the detected boundary edges of the same model. Figure 8(b) shows the remaining gaps after the *stitching* operation. These gaps are then triangulated via the *filling* operation. The flying minnow model consists of 5566 vertices and 9062 faces. The resolution tolerance, ϵ_r , and the glue tolerance, ϵ_g , used are 0.1 and 0.01 mm 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 may introduce self-intersecting triangles in the boundary regions. 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 is also built during the processing. 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 mm respectively.

The discrete geometry of Infiniti G35 doors is shown in Figure 11(a). The model consists of 6443 vertices and 10814 faces. It has a large gap near the upper-right corner due to a missing surface. Figure 11(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. Figure 12(a) and figure 12(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 expansions in order to process the model with ϵ_r and ϵ_g of 2.25 and 0.1 mm respectively. This test

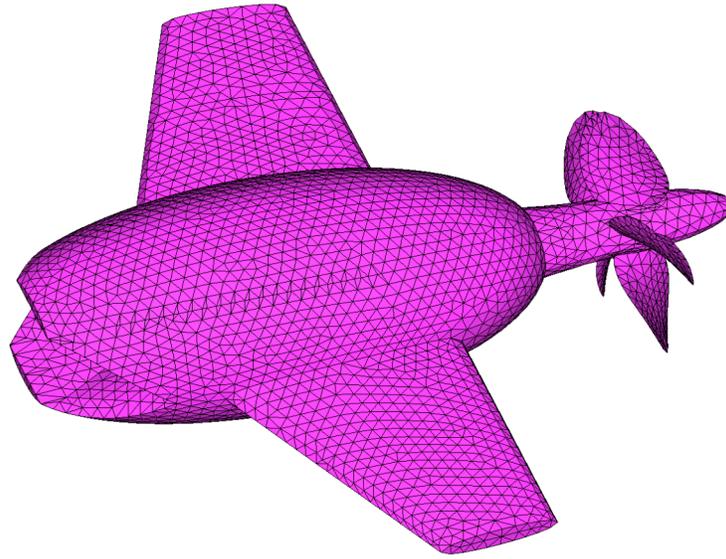


(a)



(b)

Fig. 7. Flying minnow (a) Original geometry. (b) Boundary edges before processing.

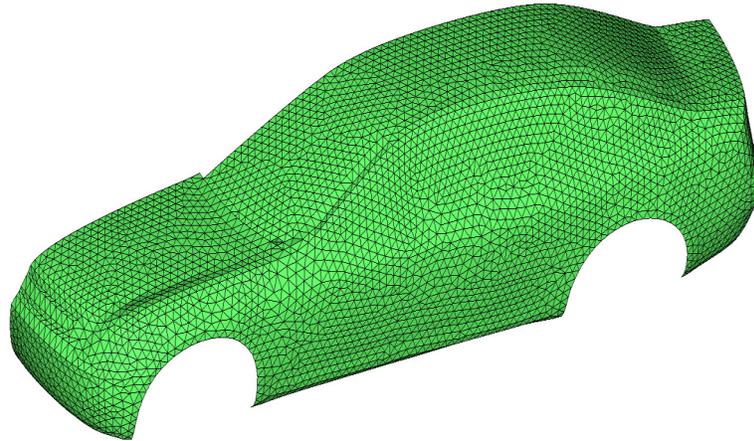


(a)

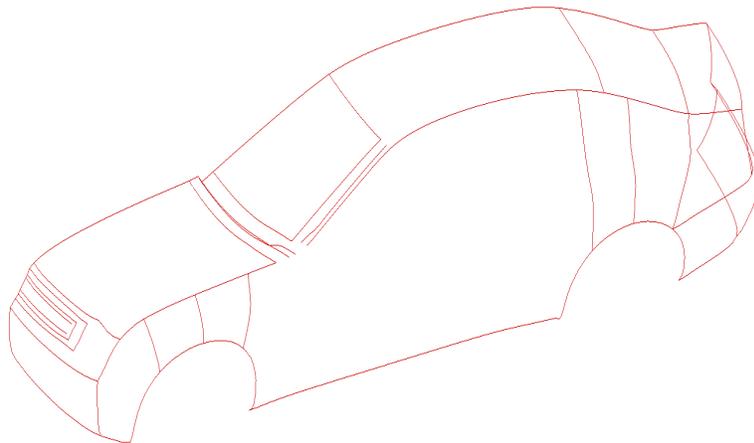


(b)

Fig. 8. Flying minnow (a) Processed geometry. (b) Gaps filled with new triangles.



(a)



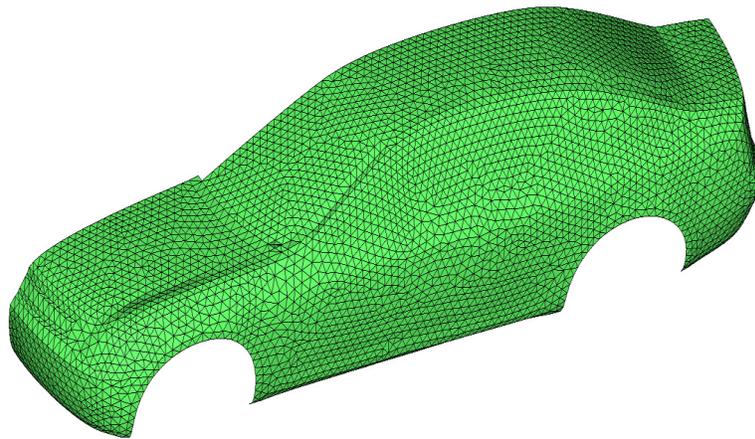
(b)

Fig. 9. Body shell of Infiniti G35 (a) Original geometry. (b) Boundary edges before processing.

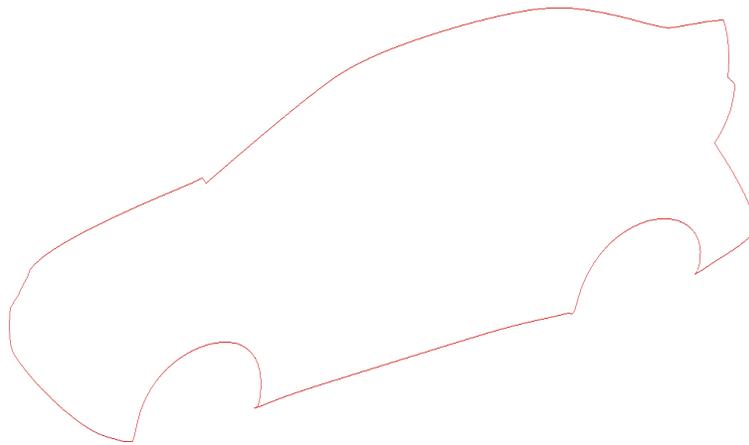
case shows that the topology generation algorithm can even create small missing geometry entities up to some extent. For example, it did add new faces in place of a small missing surface near the upper-right corner of the door.

6 Conclusion

Automatic detection and processing of commonly found geometrical and topological issues such as gaps, overlaps, intersections, T-connections, invalid or no topology,



(a)



(b)

Fig. 10. Body shell of Infiniti G35 (a) Processed geometry. (b) Boundary edges after processing.

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. Results demonstrate that the generality, accuracy and efficiency of the topology generation algorithm appears to be a significant improvement over existing methodologies. In addition, the processed models are guaranteed to be free of self-intersecting boundary edges. This work is a step towards automatic geometry processing for mesh generation applications. There are many issues that need to be addressed to automate the pre-processing of geometry for the same application. For example, the user has to provide an appropriate value of distance threshold. It may

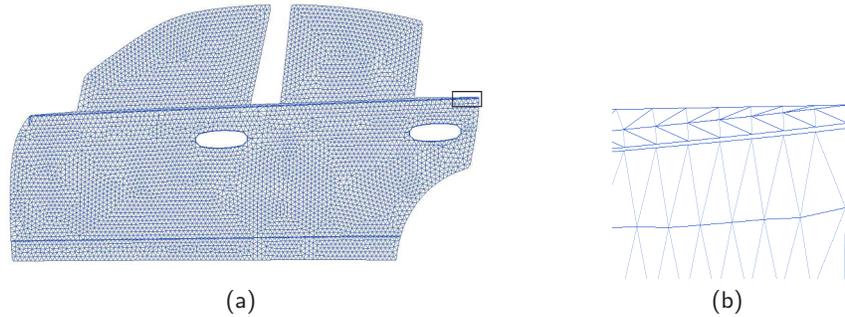


Fig. 11. Doors and windows of Infiniti G35 (a) Original model. (b) Enlarged view of the rectangular region showing the gap due to a missing surface.

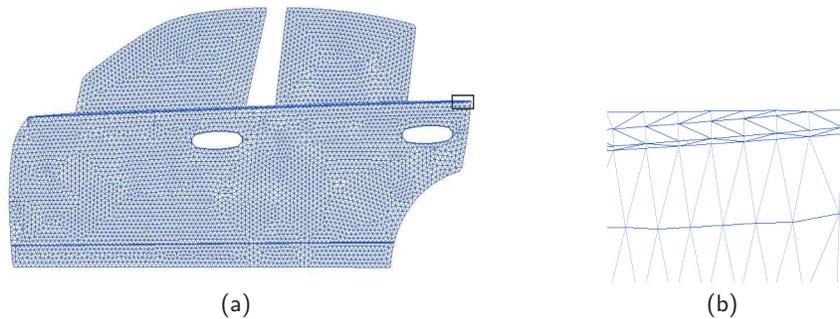


Fig. 12. Doors and windows of Infiniti G35 (a) Processed model. (b) Enlarged view of the rectangular region shows that the gap is filled with triangles in the processed model.

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.

[Aft97] M. J. Aftosmis. Solution adaptive cartesian grid methods for aerodynamics flows with complex geometries. von Karman Institute of Fluid Dynamics, Lecture series 1997-02, 1997.

[BK97] G. Barequet and S. Kumar. Repairing cad models. In *Proceedings: IEEE Visualization*, pages 363–370, Phoenix, AZ, 1997.

[BMSW91] D. R. Baum, S. Mann, K. P. Smith, and J. M. Widget. Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. In *Proceedings: SIGGRAPH*, volume 25, pages 51–60, Las Vegas, Nevada, 1991. ACM.

[BS95] G. Barequet and M. Sharir. Filling gaps in the boundary of a polyhedron. *Computer Aided Design*, 12(2):207–229, 1995.

- [BW92] J. H. Bohn and M. J. Wozny. Automatic cad-model repair: Shell-closure. In *In H. L. Marcus et al., eds., Proc. Solid Freeform Fabrication Symp.*, pages 86–94, Austin, TX, 1992. The Univ. of Texas.
- [GH97] M. Garland and P. S. Heckbert. Surface simplification using quadratic error metrics. In *Proceedings: SIGGRAPH Computer Graphics*, pages 209–216. ACM, 1997.
- [GTLH01] A. Guezic, G. Taubin, F. Lazarus, and B. Horn. Cutting and stitching: converting sets of polygons to manifold surfaces. *IEEE Transaction on Computer Graphics*, 7(2):136–151, 2001.
- [H. 90a] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990. ISBN 0-201-50300.
- [H. 90b] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990. ISBN 0-201-50255-0.
- [HLBZ02] J. Hu, Y. K. Lee, T. Blacker, and J. Zhu. Overlay grid based geometry cleanup. In *Proceedings: 11th International Meshing Roundtable*, pages 313–324. Sandia National Laboratories, 2002.
- [IGE88] Initial graphics exchange specification (iges) version 4.0. NBSIR 88-3813, 1988.
- [J. 98] J. F. Thompson and B. K. Soni and N. P. Weatherill. *Handbook of Grid Generation*. CRC press, Boca Raton, Florida, 1998.
- [KSS86] A. Kalvin, E. Schonberg, J. T. Schwartz, and M. Sharir. Two-dimensional model-based, boundary matching using footprints. *Int J. of Robotics Research*, 5(4):38–55, 1986.
- [MD93] I. Makela and A. Dolenc. Some efficient procedures for correcting triangulated models. In *In H. L. Marcus et al., eds., Proc. Solid Freeform Fabrication Symp.*, pages 126–134, Austin, TX, 1993. The Univ. of Texas.
- [MF96a] S. M. Morvan and G. M. Fadel. Ivecs: An interactive virtual environment for the correction of .stl files. In Conference on Virtual Design, Univ. of California, Irvine, CA, 1996.
- [MF96b] S. M. Morvan and G. M. Fadel. Ivecs, interactively correcting .stl files in a virtual environment. In *In H. L. Marcus et al., eds., Proc. Solid Freeform Fabrication Symp.*, Austin, TX, 1996. The Univ. of Texas.
- [MF96c] S. M. Morvan and G. M. Fadel. Virtual prototyping using .stl files. International Body Engineering Conference (IBEC), Detroit, MI, 1996.
- [MF97] T. M. Murali and T. A. Funkhouser. Consistent solid and boundary representation from arbitrary polygonal data. In *In Proc. Symp. On Interactive 3D Graphics*, pages 155–162, Providence, RI, 1997.
- [MW95] D. L. Marcum and N. P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA Journal*, 33(9):1619–1625, 1995.
- [PH97] J. Popovic and H. Hoppe. Progressive simplicial complexes. In *Proceedings: SIGGRAPH Computer Graphics*, pages 217–224. ACM, 1997.
- [PMR] P. S. Patel, D. L. Marcum, and M. G. Remotigue. Automatic cad model topology generation. *International Journal of Numerical Method in Fluids*, under review.
- [PMR05] P. S. Patel, D. L. Marcum, and M. G. Remotigue. Building topological information for triangulated models. SIAM Computational Science and Engineering Conference, February 12-15, Orlando, Florida, 2005.

- [RW92] S. J. Rock and M. J. Wozny. Generating topological information from a bucket of facets. In *In H. L. Marcus et al., eds., Proc. Solid Freeform Fabrication Symp.*, pages 251–259, Austin, TX, 1992. The Univ. of Texas.
- [SM95] X. Sheng and I. R. Meier. Generating topological structures for surface models. *IEEE Computer Graphics and Applications*, 15(6):35–41, 1995.
- [SS87] J. T. Schwartz and M. Sharir. Identification of partially obscured objects in two and three dimensions by matching noisy characteristics curves. *Int J. of Robotics Research*, 6(2):29–44, 1987.
- [STL89] Stereolithography interface specification (stl). valencia, ca. 3D Systems Publications, 1989.
- [WS02] Z. J. Wang and K. Srinivasan. An adaptive cartesian grid generation method for “dirty” geometries. *International Journal of Numerical Method in Fluids*, 39:703–717, 2002.