OVERLAY GRID BASED GEOMETRY CLEANUP

Jiangtao Hu, Y. K. Lee, Ted Blacker and Jin Zhu

FLUENT INC, 500 Davis St., Suite 600, Evanston, Illinois 60201

ABSTRACT

A newly developed system for defining watertight volume geometry on imported foreign CAD models is described. This system is based on an overlay grid mesher that is being developed in the GAMBIT preprocessing code. It is being explored as an alternative way of cleaning dirty geometries. This overlay meshing based geometry cleanup enables an automatic volume reconstruction from inconsistent B-Rep surface geometries. These inconsistencies include common edges defined twice, small gaps, holes and overlapping of faces. This new approach holds the promise of higher automation levels, speed and robustness when compared to more traditional interactive cleanup methodologies.

Keywords: geometry cleanup, dirty geometry, faceted models, overlay mesh

1. INTRODUCTION

Within one CAD system, it is generally easy for designers or analysts to create "clean" geometry with well-connected faces and watertight volumes. However, analysts often use different software tools to develop a finite element mesh for analysis.

Unfortunately all CAD systems are unique in their handling of model tolerance, topology connectivity and geometry smoothing. Thus interoperability between different CAD and preprocessing programs is always a big obstacle to designers and analysts. This problem is particularly acute for analysts who find it advantageous to explore combinatorial uses of different vendors' systems. The result of exporting and importing geometry using standard protocols such as STEP, IGES, native CAD formats and faceted representations is often a disconnected model with gaps or overlapping between surfaces. Hence importing a model with watertight volumes is difficult to achieve in practice.

This problem is compounded by the fact that most volume meshing algorithms rely on a watertight volume. Thus different geometry healing or cleanup tools have been developed [1-4].

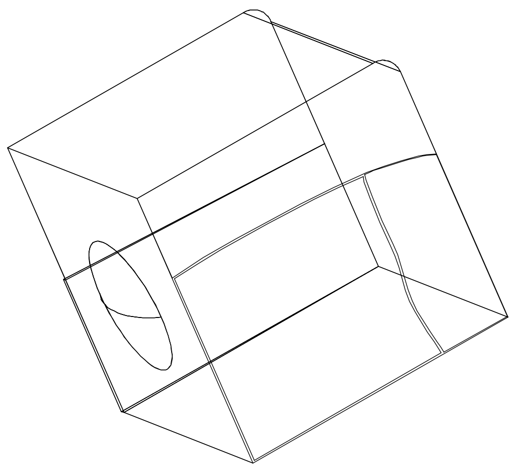


Figure 1 An example of dirty geometry imported from IGES.

One approach to cleanup is to use direct geometry changing tools. However, these are computationally expensive, complex and often produce undesirable side effects or unacceptable models. GAMBIT has augmented these healing tools with virtual tools to connect/disconnect or merge/divide edges and faces; then stitches them into watertight volumes. This method of topology adjustment independent of geometry changes (i.e. virtual topology) is becoming more common in commercial software. A certain level of automation can be achieved using virtual topology operators and a user specified tolerance. This tolerance can be based on the tolerance of the original CAD system, a percentage of the shortest edge length, etc. However, such automation usually falls short for more complicated geometries. The cleaning of the remaining disconnected geometry, as well as other refeature/defeature operations, usually requires manual intervention.

This paper explores a novel method that has the potential to provide a more automated tool for geometry cleanup. The input to the algorithm is the set of disconnected faces. The system automatically replaces the disconnected faces with newly constructed faces to create a watertight volume.

This cleanup relies on the overlay-meshing algorithm. Overlay meshing [5], of which the more traditional Cartesian meshing [6] is a subset, is also newly developed in GAMBIT. It has a significant advantage in that it does not require volumes to perform the meshing process. Instead, it only uses faces as input. This enables it to provide a continuous domain with hex cells based on a set of surfaces. This paper details our method of leveraging this technology to produce a set of connected faces, resulting in a watertight volume when successful. Although this paper represents a preliminary investigation only, results are very promising.

2. OVERVIEW OF CURRENT GEOMETRY SYSTEM

We are using GAMBIT as the framework for this technology and as such rely heavily on its geometry system. Therefore it's necessary to briefly introduce the basics of this system. GAMBIT provides a mixed pallet of geometrical representations including real (standard B-Rep), faceted, direct CAD and virtual as described below:

- **Real**. GAMBIT employs a solid modeling kernel, ACIS, to provide basic geometric creation and modification functionality. We term this standard B-Rep model as "real" geometry.
- **Faceted**. A faceted geometry uses a set of direct facets (triangles) to describe the shape. These facets can originate from prior meshes or from faceted representations typically employed for graphical display.
- **Direct CAD**. The direct CAD geometry relies on an external CAD system to supply the geometric functionality for the entities. These representations are then a set of appropriate pointers and associated data.
- **Virtual**. A virtual entity does not have a geometric definition of its own, but is based on other entities. GAMBIT also provides virtual topology tools, which are used to edit the topological entities only of the object. Usually such virtual topology editing includes merging of adjacent entities of similar dimension, splitting of entities into parts, construction of individual virtual entities and connection of multiple representations of a single entity into one [7][8].

To provide a unifying framework for these diverse types, GAMBIT has separated the topology from the geometry of all these types. This then enables topology operators independent of geometry, i.e. virtual topology.

For efficiency, all four kinds of faces contain a triangular facet representation. GAMBIT uses this faceting as a fast approximated evaluation for graphics as well as for some geometry/mesh operations. Such a faceting is shown in Figure 2.

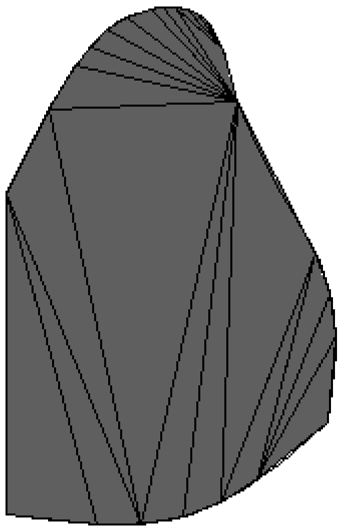


Figure 2 A Faceted representation for a general face.

3. OVERLAY GEOMETRY CLEANUP

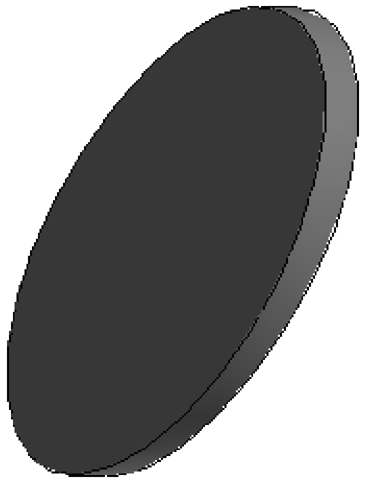
The proposed technique in this study is to cleanup a dirty geometry by taking advantage of the overlay meshing tool in GAMBIT. In this chapter, a concise summary of the method developed will be presented and details of key procedures and important features will be discussed.

The overlay cleanup technique can be divided into two phases, 1) edge intersections on a structured grid, and 2) geometry extraction from the intersected cells. These are described below.

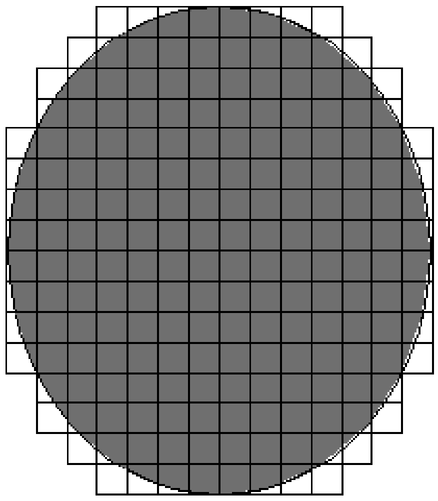
3.1 EDGE INTERSECTION COMPUTATIONS IN STRUCTURED GRID

To calculate intersections, a simple structured grid is first projected over the geometry. Cells in the grid are then recursively refined until certain accuracy is achieved. While refining the cells, the intersection information between cell edges and the faceted representation of the dirty geometry are computed and stored with the edge. The result is a set of intersecting points on edges of the computed overlay grid structure, as described in the sections below.

3.1.1 STRUCTURED BACKGROUND GRID GENERATION



(a)



(b)

Figure 3 (a) A cylinder shaped model before overlay mesh. (b) Intersecting hexes generated using overlay mesher.

The first step of the procedure is to overlay a structured grid over the dirty geometry domain. This procedure starts with a simple grid with a limited number of cells: typically one. Then cells in the grid are recursively refined as in a typical octree technique [9], until a desired accuracy is achieved. During this refinement, the cells having intersections with the dirty geometry are determined and refinement is carried out only for such cells. Unlike a typical octree mesher, any non-boundary cell can be deleted. This should reduce the required memory and computations significantly, and is being explored. Figure 3 shows a boundary grid for a simple geometry.

3.1.2 EDGE INTERSECTIONS

For speed and memory efficiency, currently the overlay-meshing tool only provides cutting information for the cell edges. Providing cutting information for cell faces, though not impossible, will significantly slow the performance of this algorithm. Thus the cleanup tool must deduce how the edge intersecting points would result in face cuts for each cell face. To assist in creation of correct face and cell cuts, we store the intersecting facet and the associated geometric face with each intersection edge.

3.1.3 CELL SIZE CONTROL

As briefly mentioned before, a simple grid is progressively refined into a final grid of uniform-sized cells. Basically, the finer the mesh, the more accurately the geometry will be reconstructed. The price is more memory and computation time. See Figure 4 (a) and (b) for a comparison of mesh size differences. Numerical experiments have shown that cell size control plays a crucial role in the following way.

- **Cell size dictates accuracy:**

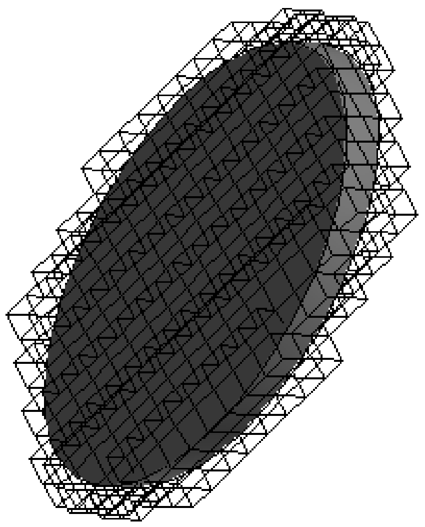
Cell size plays a very important role, in that it not only determines the accuracy of the final geometry, but also affects whether the cleanup will be successful or not.

This is because the overlay cleanup technology uses a linear interpolation to decide the cutting plane within each hex. Larger size meshes will lose geometry details more easily. If too many details are lost, it is impossible to recreate the volume.

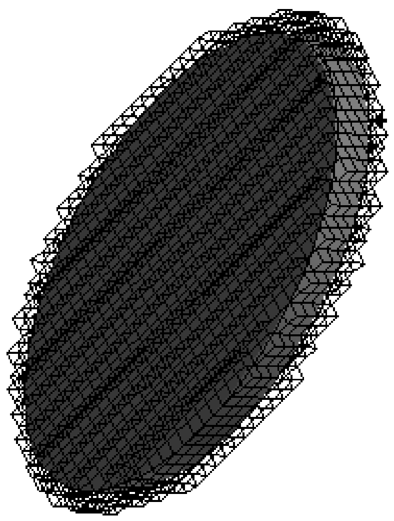
Since automated geometry cleanup is our goal, we have initially not utilized size functions [10] in that the process may become too complex for the user.

- **Memory and speed performance:**

The number of final crossing cells dominates the performance of the procedure. As a finer mesh is used, more memory usage and more computations for edge intersections are required.



(a)



(b)

Figure 4 (a) Intersecting hexes generated using overlay mesher, mesh size = 1. (b) Intersecting hexes generated using overlay mesher, mesh size = 0.5

- **Over refinement costs:**

Over-refinement also affects the robustness of the cleanup procedure. In general, a finer mesh gives more accurate results than a coarser mesh does. However, in some cases, excessively small cells may cause a failure in the cleanup procedure. Typical examples are cases when there are several layers of cells falling in the gap of the dirty geometry or several layers of cells covering the overlap region. In such cases, the expected intersections in those cells are either totally lost or become too complex for our initial heuristic algorithms, as will be described later. Ultimately this will cause a failure in the geometry reconstruction. Figure 5 schematically demonstrates the effect of over refinement on the robustness of the developed technique. In the first two figures, the current algorithm is able to match the two ends of the gap detected by taking advantage of neighbor information of the cells containing the gap ends. However, when there are one or more layers of cells between the gap ends, such as in the third figure, it is hard to match corresponding gap ends correctly. As a result, the current algorithm fails to reconstruct a volume when there are such cases due to the excessive refinement. It seems that the minimum meshing size should at least be larger than the largest gap size and larger than the largest overlap size of the model. Such a size would guarantee that no hex cell would sit in between the edges without any intersecting points (gap case), and guarantee that at most two layers of hex cells will cover the gap or the overlap. At present, our heuristics are not adequate to handle models with extensive overlapping areas.

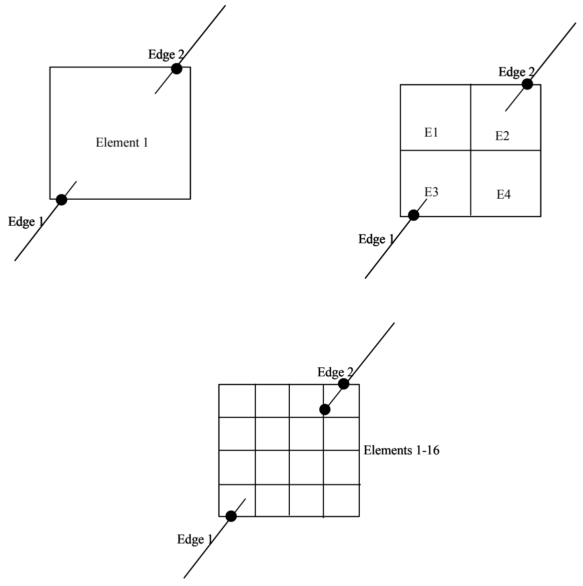


Figure 5 Over refinement cost for a gap case

3.2 GEOMETRY RECONSTRUCTION

During geometry reconstruction, the intersecting points are connected into facets. These facets are then organized into new vertices, edges, faces, and finally into a volume. These techniques are described below.

Following the creation of the overlay hex mesh, we get a list of intersecting cells along with intersecting points on each cell edge. Theoretically, a hex edge or a hex face can have any number of intersecting points. But, with a good meshing size, typically, a cell edge will have 0, 1 or 2 intersecting points. Likewise, a cell face will have 0, 1, 2, 3 or 4 intersecting points. If the geometry were clean (surfaces are connected), a cell face should never have only one intersecting point. However, if the cell edge is almost parallel to the intersecting facet, numerical difficulties prevent us from getting an accurate intersecting point. With additional complexity of dirty geometries, we will have cell faces that have only one intersecting point.

With the completion of the meshing phase, the original geometry (faces) is discarded. A totally new set of faces is then created from the edge intersections (intersection points and facets) stored on the hex mesh. Thus, in the descriptions below, "edges" and "faces" shall refer to the new edges and faces being created rather than referring to the old faces of the original geometry.

3.2.1 Split cell faces

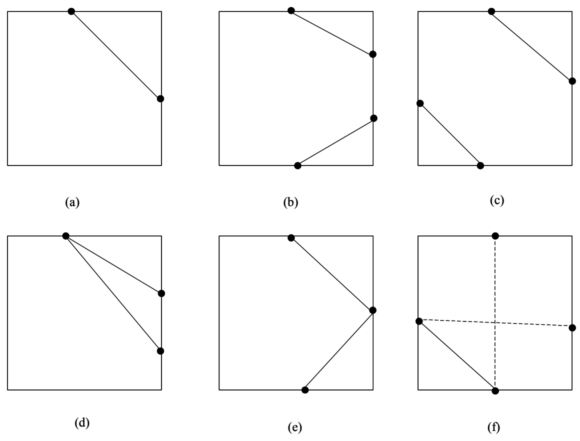
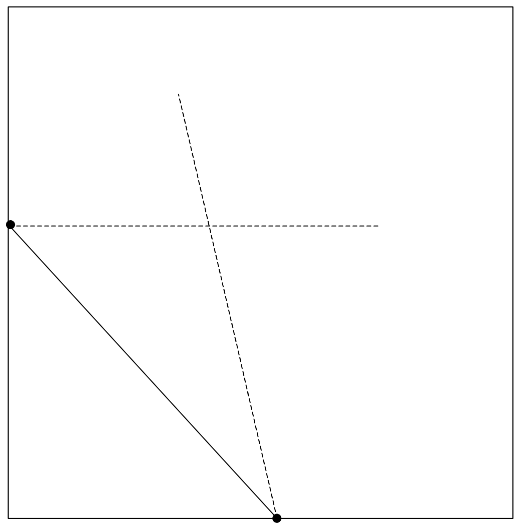


Figure 6 Different intersecting points' pattern on one cell face, and how the edges get connected.

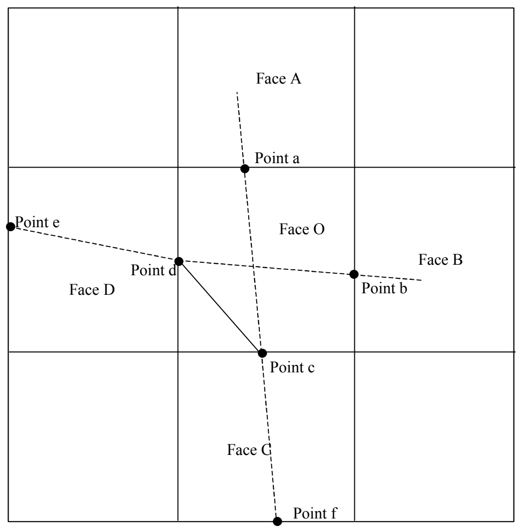
The first step in the reconstruction process is to transform the edge intersections into face intersections, or edges that split the cell face. Creation of the face splits naturally depends on the number of cell edge intersecting points. For example, if a cell face has only two intersection points, then one can directly connect them into a splitting edge (6a). However, if a cell face has more than two intersection points, then ambiguities arise. Figure 6 shows a number of connection possibilities. This ambiguity can often be resolved based on the information stored with each intersection. For example, 6c shows a single intersection on each edge. If only 2 of the intersections were generated from facets directly connected to each other, then it is natural that these should be connected. Of course the more complicated connections require extensive heuristics. Empirically, we have found the most common cases to be 1) two intersecting points on one cell face (6a) and 2) one face having four intersecting points and two of them on the same edge (6b). Figure 6 shows details of various edge creation methods.

One special case, shown in Figure 6f, is when the original faces are intersecting each other. If the intersection is confined to one cell (see Figure 7 (a)), the small overlapping area will be automatically removed. When the overlap goes through two layers of cells (see Figure 7 (b)), there will be more than two intersecting points on one cell face (Face O). To handle this case, the code checks through the neighboring cell faces (Face A, B, C and D). The objective is to find where the facet connectivity stops. In this case, they are Face A and B. Once the terminating ends are found, they are retracted back to the crossing face. In this example intersection, points a and b are removed. This leaves the remaining points, c and d, to be connected appropriately.



Dotted lines are the original facets' orientations;
Solid line is the final edge created.

(a)



Dotted lines are the original facets' connectivity;
Solid line is the final edge created.

(b)

Figure 7 (a) One cell covers the intersecting facets. (b) Details on how to create edge on overlap case if two layers of cells cover the overlap.

3.2.2 Single intersection face reduction

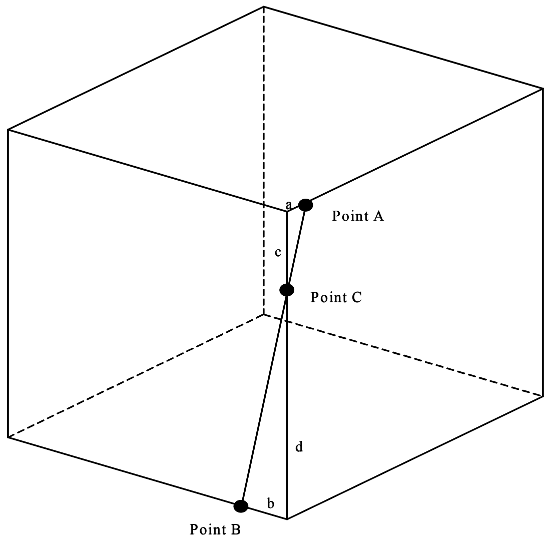


Figure 8 (a) A cell has intersection points A and B after mesh, find point C by satisfying $\frac{a}{b} = \frac{c}{d}$.

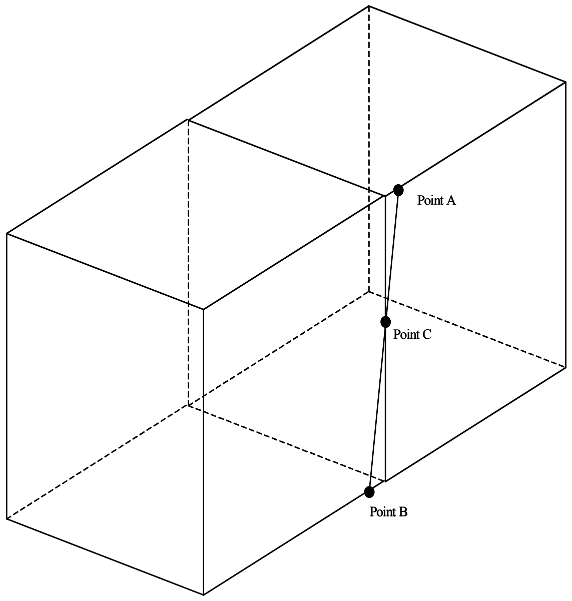


Figure 8 (b) Using the same method to determine Point C from neighboring cells' intersecting points A and B.

After all cell faces have been examined and connected with appropriate edges, we collect all cells that have faces owning only one intersecting point. Then we check if any such cell has two neighboring single intersection faces. If any are found, we create a new intersecting point on the shared edge by interpolating with respect to the other two points (see Figure 8 (a)). Finally, for those cells not having neighboring faces, we compare neighboring faces on neighboring cells. If they happen to be connectible, as shown in Figure 8(b), we create a new intersection on their shared edge. At the end of this procedure, single intersection faces may still exist and these remain for later processing.

3.2.3 Facet hex cuts

The next step in the process is to use the splitting edges of the faces to create splitting facets of the cells. From the edges, we can find either one or two closed patches in each hex. We then facet each patch for better data processing by creating a center node and connecting the center node to all edges. Figure 9 illustrates how the cutting facets are generated. Note that some loops will not close as shown in Figure 9d. For these loops, we calculate a center point and generate facets with the edges of the open loop.

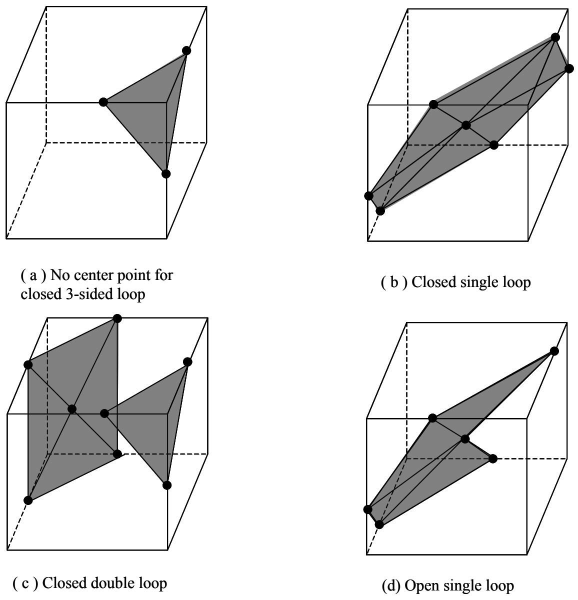


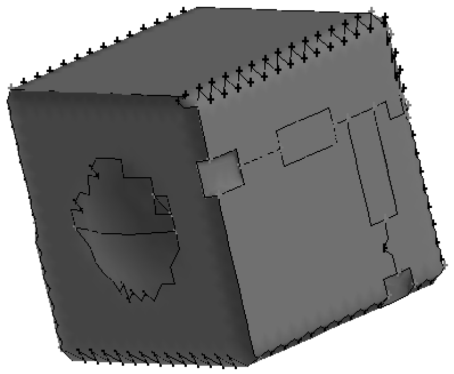
Figure 9 Creation of patches within each hex.

3.2.4 Creation of faceted geometric faces

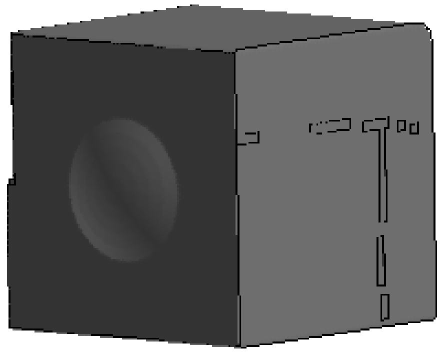
These newly generated cell facets can be combined to form a geometric face. These are passed into our geometry converting tools just as if they were imported meshes. This converting tool will create faceted geometric faces based on either feature angles or facet groups that are initially passed in. We attempted to designate groups for the converting tool based on which original face it was closest to. We calculated the proximity by comparing the distance from the facet's center to the face. However, this tended to create zigzag shaped boundaries as shown in Figure 10 (a). Neighboring triangular facets tended to oscillate between different faces because the center points vary slightly in location. Also, geometry import often splits of a larger surface into an inappropriate set of smaller ones. Merging these back together is impossible using closest face approach. One advantage of the closest face method is that it tended to roughly maintain the existing geometric faces. Notice the edges bounding the bumping circular area in

Figure 10 (a). Although the boundaries are jagged, there are still 2 faces as in the original input geometry. In consideration of problems mentioned above, we finally decided to use a feature angle for the converting. This allows the conversion to create faces as smooth as it will. However, this is also problematic in that we will lose the original face boundaries. This then may cause problems for the analyst when adding boundary conditions on faces. Since GAMBIT provides mechanisms to correct the conversion interactively, this solution was deemed adequate.

Figure 10 (b) shows the result of using a feature angle based conversion method. Notice the face boundaries are much smoother even though the circular faces have been merged away. The smaller rectangular faces occur as part of hole filling as described below.



(a)



(b)

Figure 10 Constructed geometry (a) by facet groupings based on proximity to original geometric faces (b) reconstructed with a feature angle (30 degree)

3.2.5 Filling holes

In order to create a watertight volume, each face edge should share two faces. As described above, there are cases where the edge intersecting points (Point L in Figure 11) are missing or left dangling. This leads to the open loops described earlier. These open loops then will form missing patches in the cell splitting when combined with neighbor cells. Figure 11 is an example of one edge missing an intersecting point and the resulting missing patch. Thus a hole in the faceted faces is formed. When there are more missing points in one hex as well as in neighboring hexes, the holes can grow larger with an irregular folded shape as the original geometry dictates. The code will collect all the connected edges that bound a hole, group them into loops, and try to create virtual faces on each of the loops. Here, points A, B, C, D, E, F, G, and H are intersecting points, point L is the missing intersecting point, and points I, G and K are points the code created when faceting the cutting plane in each hexes. The shaded area is the previously created facets' face, while the area bounded by points A, D, G, F, K, C and I is the void space on which the code will attempt to create a virtual face.

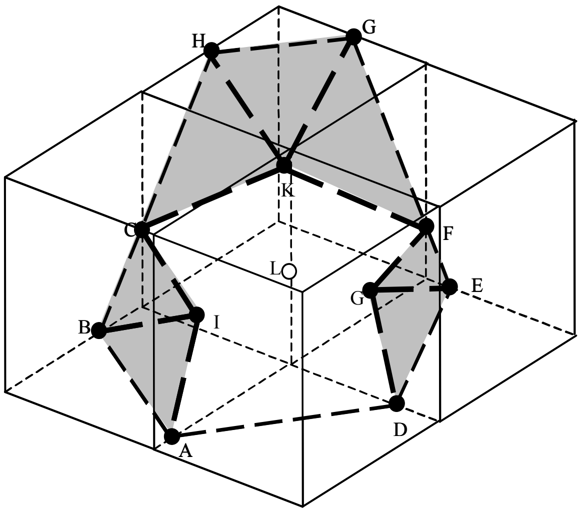


Figure 11 Filling the hole represented by points A, D, G, F, K, C, I, A with a virtual face.

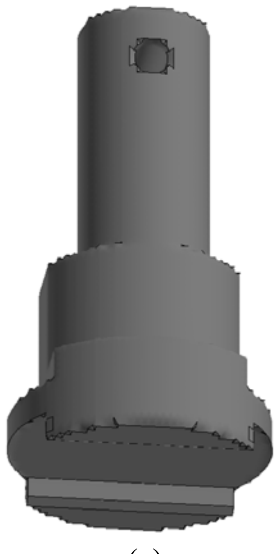
3.2.6 Stitch volume

If all above procedures are successful, we get a set of new geometric faces that are connected to each other. There will be no holes and no overlaps in between. They then form a watertight volume using a standard volume stitch operator.

3.2.7 Connected faces instead of a volume

For cases where the input is not a closed domain, e.g. just two disconnected faces, procedures 1-4 still can be used. However, if the geometry does not represent a volume in some gross way, then one would expect disconnected edges. This translates to "holes" that should not be filled. Thus, when filling holes in procedure 5, we check the loop to see if it covers a void space. The check is accomplished by using a bounding box. We check if all newly generated facets' points are within the bounding box. If so, the loop doesn't define a void space. As a result, no virtual faces will be generated and no volume will be created. A warning message to alert the user to this fact is provided.

4. EXAMPLES


(a)

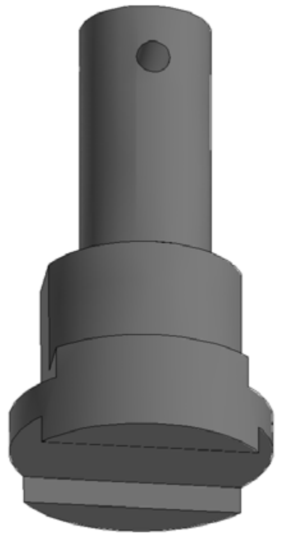
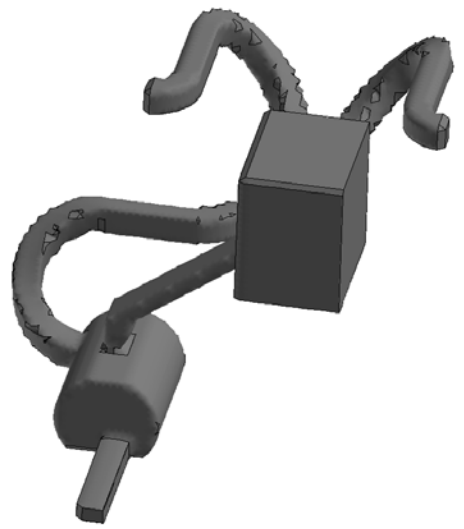

(b)

Figure 12 Mechanical pin (a) Volume created using mesh size = 2. (b) Desired volume in GAMBIT

Empirically we found that by using a proper meshing size, most of the dirty geometries can be converted into volumes, Figure 12 (a) shows a volume created for a column shaped geometry, and Figure 12 (b) shows the desired volume. Notice that some edges have been rounded and small virtual faces introduced to fill holes. Also, the edges have a limited zigzag effect due mainly to the overlay mesh position relative to the geometry. Figure 13 (a) shows the volume created for a mixer and Figure 13 (b) is the desired mixer in GAMBIT. In this example, there are numerous virtual faces created to cover holes in the faceted representation. Although both examples create watertight volumes, there are a lot of small virtual faces that would need to be merged to simplify the geometry.


(a)

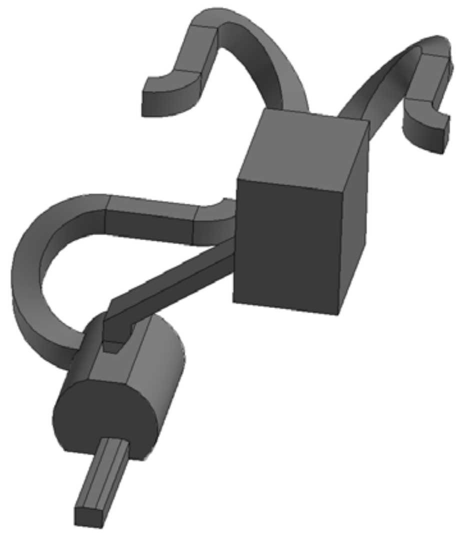
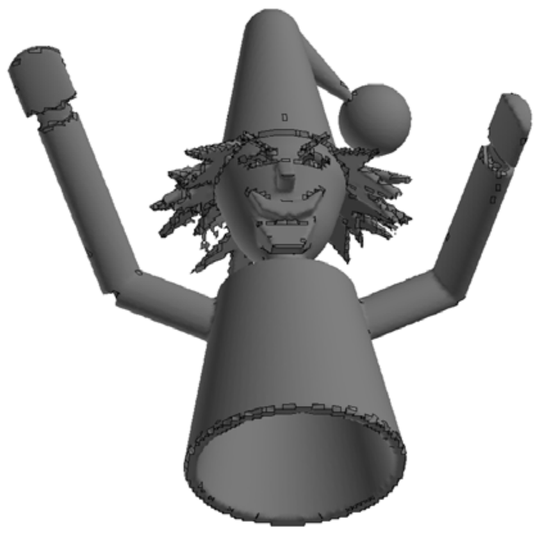
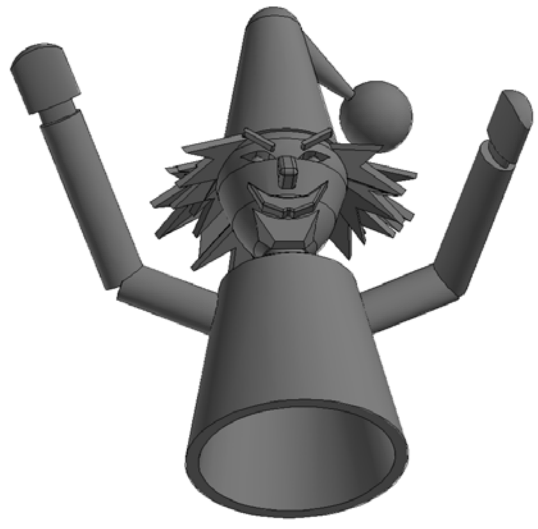

(b)

Figure 13 Mixer example (a) Volume created using mesh size = 4, overall dimension ≈ 150. (b) Desired volume in GAMBIT.

Figure 14 is an example where the current code fails to create a volume out of the dirty geometry, but we can see even in this case, the faces are created and connected to each other. This geometry has a lot of folded structures in the hair part so there are many cell edges intersecting the geometry more than once. The current implementation cannot handle such complex cases. As a result, many edges were missed and many virtual faces were employed to fill the void spaces. In this particular example, some loops are too degenerate to be handled in GAMBIT. Thus a final volume was not generated.



(a)



(b)

Figure 14 Jack example (a) Geometry created using mesh size = 0.5. Overall dimension ≈ 30. Notice that there are very detailed geometries around the hair; it failed to create volume because some of the loops failed to create virtual faces. (b) Desired volume in GAMBIT.

5. DISCUSSIONS AND FUTURE WORK

This paper explores a geometry cleanup technique by using an overlay meshing tool. The whole procedure includes two phases: intersection computation with a structured grid and geometry reconstruction. By using some reasoning techniques, it is possible to recreate vertices, edges and faces, finally stitching them into volumes.

While this work is only in the preliminary phase, it shows promise as an alternative to more manual cleanup methods. Advances are required to resolve the weaknesses discovered as follows:

- A feature-based refinement is needed on the generated overlay grid. This would eliminate the dependence on mesh size and further reduce user input. It would also decrease the complexity of holes to be filled.
- A method of extracting crisper edges is being explored. Since the edge intersections record the facet of the original geometry, it is thought that we can employ the facet normal information. Figure 15 shows how these normals could be used to improve the splitting edges of the cell faces, causing a similar improvement in the splitting facets. This is being explored.
- A reduction in the number of virtual faces is needed. We believe this will occur as our heuristics for adding missing intersections matures.
- It may be profitable to employ existing facets where available instead of reconstructing all of the facets. Then the method could focus on only those areas where gaps and overlaps exist.

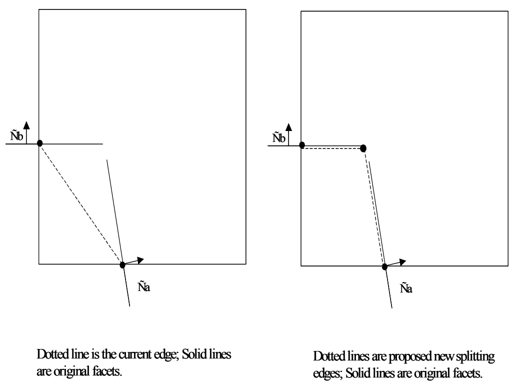

Ñb
Ña
Ñb
Ña
Dotted line is the current edge; Solid lines are original facets.
Dotted lines are proposed new splitting edges; Solid lines are original facets.

Figure 15 Proposed improvements in cell face splits

REFERENCES

[1] G. Butlin and C. Stops, "CAD Data Repair," Proceeding 5th International Meshing Roundtable, SAND96-2301, pp7-12, 1996.

[2] NAFEMS, How to – Integrate CAD and Analysis, CAD/FE Integration Working Group,

NAFEMS Finite Element Methods & Standards, 1996.

[3] T. J. Tautges, "Automatic Detail Reduction For Mesh Generation Applications," Proceeding 10th International Meshing Roundtable, pp. 407-418, 2001.

[4] S. J. Owen and D.R. White, "Mesh-Based Geometry: A Systematic Approach to Constructing Geometry From A Finite Element Mesh", Proceeding 10th International Meshing Roundtable, pp.83-96, 2001

[5] Y. K. Lee, J. Zhu and T. Blacker, "A three-dimensional Cartesian grid mesh generation for staircase meshes," Internal report, Fluent Inc., 2002.

[6] H. Samet, "The Design and Analysis of Spatial Data Structures. Addison-Wesley Series on Computer Science and Information Processing." Addison-Wesley Publishing Company, 1990.

[7] T. Blacker, A. Sheffer, J. Clements and M. Bercovier, "Using virtual topology to simplify the mesh generation process". ASME, Applied Mechanics Division. P45-50, 1997.

[8] A. Sheffer, T. D. Blacker and M. Bercovier, "Clustering: Automated Detail Suppression Using Virtual Topology," Joint ASME/ASCE/SES Summer Meeting. June 1997.

[9] M. A. Yerry and M. S. Shephard, "Automatic three-dimensional mesh generation by the modified-octree technique," IJNME, 20, pp. 1965-1990, 1984.

[10] GAMBIT 2.0 User's Manual, Fluent Inc. (2002)