Sealing Faceted Surfaces to Achieve Watertight CAD Models

B.M. Smith, T.J. Tautges, P.P.H. Wilson

October 2010

UWFDM-1389

Proceedings of the 19th International Meshing Roundtable, October 3-6, 2010, Chattanooga TN.

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

Sealing Faceted Surfaces to Achieve Watertight CAD Models

B.M. Smith, T.J. Tautges, P.P.H. Wilson

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

October 2010

UWFDM-1389

Proceedings of the 19th International Meshing Roundtable, October 3-6, 2010, Chattanooga TN.

Sealing Faceted Surfaces to Achieve Watertight CAD Models

Brandon M. Smith[1], Timothy J. Tautges[2], and Paul P.H. Wilson[3]

[1] University of Wisconsin-Madison bmsmith6@wisc.edu
[2] Argonne National Laboratory tautges@mcs.anl.gov
[3] University of Wisconsin-Madison wilsonp@engr.wisc.edu

Summary. Solid modeling engines are capable of faceting CAD models but may facet each face independent of adjacent faces. Regions of the resulting model have gaps between faces of their boundaries. An algorithm is described to seal faceted CAD models such that the boundary of neighboring faces has the same discretization along their shared edges. The algorithm works by sealing skin edges of geometric face faceting to geometric model edge facets, using vertex-vertex and vertex-edge contraction. Ten intricate CAD models of moderate to high complexity are tested with a range of facet tolerances. The algorithm succeeds in creating watertight models in most cases, with failures only at extreme values of facet tolerance and/or in the presence of geometric features which are outside the normal features encountered in most models.

Keywords: faceted model, watertight, seal, tessellation.

1 Introduction

Computational simulation relies increasingly on valid, accurate representations of the geometry of objects being simulated. Geometric models are most often constructed in Computer-Aided Design (CAD) systems, then transferred to a discretized representation for use in simulation. This discretized representation is often in the form of a Facet-Based Model (FBM), where geometric Vertices, Edges, and Faces are represented by collections of points, edges, and triangles, respectively. FBMs serve a variety of uses in computational simulation. First, they are often the means of transferring geometric models between applications, often in the form of STereo Lithography (STL) files [1]. FBMs are also sometimes used as the basis for generating the 3d discretization, or mesh, for a given geometric model [2, 3]. In other cases, the FBM is used directly, for example in monte carlo radiation transport (based on ray-tracing on FBMs)[4], and clash detection for robotic applications.

One of the reasons FBMs are used so widely is that they are provided by virtually all geometric modeling systems. Modeling engines such as ACIS [5] and Open.Cascade [6] all provide API functions for getting the triangular facets for each (geometric) Face and facet edges for each (geometric) Edge. Typically, facet edges and faces (triangles) are guaranteed to be within a facet tolerance of the geometric entities they resolve, and this facet tolerance is input to the modeling engine. However, most FBMs provided by geometric modeling systems suffer from a fundamental flaw which prevents their usage as-is for other applications: the FBMs are not watertight. That is, each Face and Edge in the geometric model is faceted independently, with neighboring Faces not sharing facet points with the Edge where they meet nor with each other. While there may be points in each of those facetings that are coincident along the shared Edge, this is not always the case, and for multi-material models, this is almost never true. For example, Figure 1 shows a model where facetings of two Faces bounding an Edge of a cylinder are far from coincident. This flaw must be fixed before FBMs can be used for the other applications mentioned above.

There has been a great deal of previous work on the subject of making faceted models watertight. These efforts fall roughly into two groups: one that views the faceted model similar to a point cloud that is the basis of deriving a closed faceted model, and another group that focuses on fixing localized problems in the model, e.g. by filling and closing holes in a faceted surface. None of the efforts reviewed in this work makes use of the topological information often available from geometric models, and few work on models having non-manifold topological features. Furthermore, these approaches vary greatly in their robustness, and few come with guarantees about what kind of FBMs can be made watertight. Implementations of these methods are also not available as open-source software, for use with other geometric modeling packages. A provably reliable solution for fixing FBMs, in an open-source implementation that could be applied in other modeling environments, is a key capability in developing other applications on FBMs.

The contribution of this paper is the demonstration of an automatic algorithm for sealing facet-based models. We demonstrate how using topological information that accompanies a certain class of geometric models simplifies this process, and prove that, under certain weakly-restrictive conditions, the method is guaranteed to succeed. Robustness of our method is demonstrated using non-manifold geometric models from a variety of applications, having thousands of Faces and Regions. The algorithm is freely available as open-source software under the LGPL license.

The construction of geometric models targeted in this work is a multi-step process. First, geometric Regions are constructed using primitive (sphere, cylinder, etc.) or free-form (spline surface) shapes and boolean operations between them, or more complex modeling operations like lofting or sweeping. Next, Regions are "imprinted" together; in this operation, any Faces and Edges that are partially coincident in space are made fully coincident, by

splitting the original Edges into coincident and non-coincident parts. This process can be viewed as the opposite of regularization, a common operation in most CAD systems. Following this, entities coincident in space and with like topology are merged, such that only one entity remains. This merging step modifies the topology of the geometric model, such that the remaining merged entity is adjacent to all entities bounded by the entities that were merged. Faces bounding multiple Regions have a sense (either forward or reverse) with respect to each of those Regions, with the convention that a forward sense implies the Face normal points out of the Region. The result of this process is a model with some Faces shared between neighboring Regions, and Vertices and Edges bounding multiple Regions.

The representation of non-manifold solid models, especially those with multiple 3d Regions sharing Faces between them, varies widely across modeling engines, whereas the representation and evaluation of manifold models is quite standardized. For this reason, most applications using this type of non-manifold geometry usually represent their own topology above the underlying solid modeling engine. This approach also simplifies implementation of "Virtual Topology" [7], facet-based geometry [8] and other specialized representations. This type of model is used to generate contiguous meshes for multi-material models by the Cubit [8] mesh generation code. The Common Geometry Module (CGM)[9] also uses this approach, maintaining its own topology graph for these and other purposes. CGM is used for all geometric modeling operations in this paper.

The remainder of this paper is structured as follows. The remainder of this Section discusses previous work in fixing FBMs, and the nomenclature and functions we use for accessing FBMs and geometric models. The algorithm for "sealing" FBMs, along with a proof of its reliability, is given in Section 2. Implementation of the algorithm is described in Section 3, with results given in Section 4. Conclusions and future work are described in Section 5.

1.1 Previous Work

Previous work in converting flawed FBMs into watertight faceted models can be grouped in two categories. Region-based approaches may guarantee watertightness of a solid bounded by triangles, but at the cost of reconstructing the entire region using an intermediate representation. Mesh-based approaches fix faceting errors with small perturbations in the vicinity of the error, but offer no guarantee of watertightness for the overall solid [10]. In a recent survey [11], Ju concludes that mesh-based methods have been most successful at repairing CAD models where errors are localized, while region-based methods are most successful at reconstructing polyhedrons from poor quality data, with loss of detail.

Region-based methods [12, 13, 14, 15] reconstruct the model based on point cloud data derived from the faceting. Because they discard the original facets, these methods do not preserve the details of the original geometric topology,

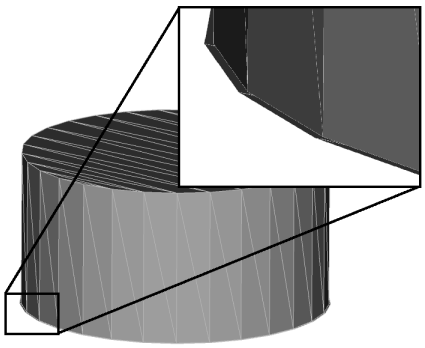


Fig. 1: A gap exists between faces of the cylinder because each face is faceted independent of bounding edges.

at least in cases where those details are not also geometrically distinct. Thus, Faces imprinted on another Face will not be preserved, since they are not geometrically distinct. Although region-based methods are the only option in the case of range-scanned data, they are not applicable to the problems described here, and are not considered further in this work.

Bohn and Wozny [16] create a watertight shell by using topology to identify local gaps in the faceting, which they close based in part on heuristics. The algorithm does not move or merge any vertices. Their method begins by identifying free edges, and organizing them into loops, where each loop bounds a region that must be filled in with facets. This filling is done using an "ear clipping" algorithm, where two consecutive edges in the loop are used to form a tringle, with the third edge replacing the two edges in the loop. Edge pairs with the smallest angles are favored in this approach. The algorithm concludes by using edge flips to improve geometric quality of the facets.

Barequet and Sharir also approach this problem [17] by idenitfying free edges, but split those edges such that the resulting edges are more consistent in edge length. A distance- and orientation-based heuristic is then used to determine matching groups of edges; gaps between these groups are filled by constructing new facets. In a subsequent effort [18], vertices within a certain distance tolerance are moved to eliminate gaps between triangulated faces. Edges are considered in pairs, rather than by groups forming arcs or loops. Free edges that remain after this process are identified and assembled into loops, and triangulated using [19]. In ambiguous situations, the user is prompted for guidance.

Sheng and Meier [20] use a series of increasing distance tolerances, up to a maximum tolerance, to merge vertices then edge pairs whose endpoints have been merged. This ensures that the best candidates will be paired with each other. One drawback of this approach is that it requires facet edges

of similar lengths to close gaps in the faceting; this is often not the case in typical facetings returned from modeling engines. Geuziec et. al [21] also note that Sheng and Meier's algorithm may create edges adjacent to more than two triangles, when the same end is merged more than once. They avoid this simply by preventing merges that will create edges adjacent to three triangles.

Borodin et al. improves on the merge-only approach by introducing a vertex-edge contraction, or t-joint operator [22, 23], where a free edge is split if a corresponding point is too far from either end but close enough to the edge. Vertex-edge contraction allows model repair to proceed using a smaller tolerance than if implemented with vertex merging alone. Kahlesz et al. [24] use a similar approach.

An approach by Busaryev et al. [25] represents face boundaries as strings of intersecting balls. Boundaries of adjacent faces are joined by combining balls using a tolerance. Although the watertight output is produced using Voronoi diagram and Delaunay triangulation techniques, the *repair* phase of combining balls resembles vertex-vertex contraction.

The advances in mesh-based watertightness are well summarized in an algorithm by Chong et al. [26]. First vertex-vertex contraction is performed by proximity. Next vertex-edge contraction is used, then large holes are triangulated. Finally skewed elements are removed by edge swapping or node merging. The last step is important because prior vertex-vertex and vertex-edge contraction likely create facets with poor aspect ratios.

These mesh-based algorithms attempt to restore missing information, in the form of where facets meet to close gaps, using three methods:

1. Perform vertex-vertex contraction by proximity.
2. Perform vertex-edge contraction by proximity.
3. Triangulate a patch across gaps that minimizes area or another heuristic quantity.

However, we observe that much of the information they try to restore based on spatial proximity and searching was known at one point, but was lost or thrown away. For example, vertex-vertex and vertex-edge contractions are usually motivated by sealing neighboring geometric Faces at shared Edges. Knowing these topological relationships greatly narrows the search for matching free facet vertices and edges. In the case where the geometric model is still available, new points in the faceting can even be projected to the geometric entities the faceting is supposed to resolve. *We believe that methods for repair of FBMs are greatly improved by taking advantage of geometric model information that previously has not been used. Furthermore, we assert that this information is often available when the original faceted model is created, but has not been used because of limitations in file formats used to transfer FBMs to applications.* The sealing algorithm described in this paper uses geometric model information to improve the robustness of obtaining watertight FBMs.

1.2 Notation and Assumptions

In this work, we consider a geometric model described in the form of a Boundary Representation (BRep). This model consists of geometric Vertices, Edges, Faces, and Regions, indicated with script letters $\mathcal{V}/\mathcal{E}/\ \mathcal{F}/\mathcal{R}$, respectively. Note that non-manifold topology is allowed; in particular, many of the models we encounter have multiple Regions, with Faces shared between two of those Regions, and Edges and Vertices shared by multiple Regions. The collection of geometric model entities forms a cell complex; that is, the intersection of two entities of like dimension is either empty or corresponds to one or more entities of lower dimension that are also in the cell complex. Each Vertex, Edge, and Face has a corresponding faceting represented by a set, denoted with V, E, F, and R, respectively, with each of these sets containing points p, and possibly edges and faces e and f, depending on the topological dimension (R is empty, since Regions have no facets, but are still represented in the model, to get adjacency relations with other Faces). We assume that the faceting for each geometric Edge with a single Vertex has at least three facet edges. Edge facetings not satisfying this assumption would not be useful for graphics purposes, and in practice this assumption is satisfied in most observed facetings. Each faceting V, E, and F is itself also a $d-$dimensional cell complex; however, this cell complex does *not* share points with the facetings of other model entities, even those bounding the model entity in question. This is also typical of faceted models from most modeling engines. A function $d(.,.)$ returns the distance between the indicated entities, and $\Omega(.)$ represents the boundary of the indicated entity, which is a collection of (d-1)-dimensional facets.

There are two tolerances that are important to the algorithm described in this paper. First, the "merge tolerance" is the distance below which two entities are considered spatially coincident; we denote this as ϵ_g, recognizing that in effect this also serves as the geometric tolerance for those model entities. This tolerance is used in various places during the merging process described earlier. The facet tolerance ϵ_f is the maximum distance between a facet edge or face and the geometric Edge or face it resolves.

We assert certain things about the faceting, based on guarantees provided by the modeling engines constructing them. First, all facet points are within ϵ_g of the corresponding model entities. While this is typical of faceted models from most modeling engines, it could also be achieved by projecting facet points to the corresponding model entity, using a function provided by virtually all modeling engines. Second, all facet edges and triangles are within ϵ_f of the corrresponding model entities. Most modeling engines providing facetings take ϵ_f as an input parameter, though in practice this input is disregarded if it is much larger or smaller than the default value used by the modeling engine. We also assume that $\epsilon_f \gg \epsilon_g$; since both these parameters can be changed through user input, this does limit the variations allowed in one parameter after the other has been chosen. Finally, all points on the boundary of a given faceting, $\Omega(E)$ or $\Omega(F)$, are within ϵ_g of *some* model entity that bounds

the corresponding model entity, though *which* bounding model entity is not known on input. While not stated explicitly by the modeling engines, in practice this has been found to be the case for both ACIS and Open.Cascade. Each faceting E and F is non-degenerate (all points of $d-$dimensional facets, $d > 0$, are distinct), and is oriented and non-inverted (tangent or normal of facet is consistent with that of the underlying model entity in the neighborhood of the facet). Although we have not encountered cases where this assumption is invalid, invalidities of this type could be fixed using preprocessing similar to that discussed in Section 2.1.

The *local feature size* (LFS)[27] at point x is the smallest closed disk centered at x that intersects two Vertices, two non-adjacent Edges, or an Edge and a Vertex that is not adjacent to the Edge. We assume that LFS $\gg \epsilon_f$. We have observed cases where this assumption is not valid; preprocessing is used to fix these cases, as described in Section 2.1.

2 Sealing Algorithm

The requirements of the proposed algorithm are:

- Seal faceted Faces along Edges to create a watertight model.
- To preserve human efficiency, the algorithm must be automatic.
- New facets must be owned by exactly one Face.
- Support non-manifold Faces.
- Fast enough to use as a preprocessing module.
- Deformation of input model should be minimized.
- Creation of new triangles should be minimized.

The input to this algorithm is a set of Vertices, Edges, Faces, and Regions with appropriate topological relationships. Faces and Edges are represented by facets and edge elements that resolve their geometric entities to within the facet tolerance. The boundary of each faceted face is not the same as the face's adjacent faceted Edges. The endpoints of adjacent faceted edges are the same such that facet edges can be joined to form loops. The boundaries of Regions are represented by the facets of their bounding Faces. The facet tolerance is selected based on the feature size and required accuracy of the faceted representation. The input model is faceted, implying that the facet tolerance has previously been selected.

2.1 Preprocessing and Input to Sealing Algorithm

On input, the geometric model may not satisfy the local feature size assumption; that is, the model may contain features that are smaller than the facet tolerance input to the sealing algorithm. For example, the 40° section of the ITER model, discussed in Section 4, has multiple Edges and Faces of near-zero length and area, respectively. These features must be removed before the

sealing process can start. Edges with length less than the facet tolerance are removed by identifying their endpoints together; Edges that average less than the facet tolerance apart from each other are merged together. Faces can be removed if all of their bounding Edges occur in merged pairs. These faces are approximately one-dimensional and are the result of imprinting neighboring regions in the solid model. Regions are removed if all of their child faces have been removed, though in practice this does not occur for any of the models tested for this work.

One additional defect was identified for several faceted models used in this work, where skin vertices were distinct but within geometric tolerance (ϵ_g) of each other. This results in exactly two extra skin edges per defect. To remedy this, skin vertices within ϵ_g of each other are merged. The search for these defects is only within individual facets or facet edges, rather than all geometrically proximate facet points, and therefore is relatively inexpensive, scaling linearly with the number of facet edges.

After preprocessing is completed, we have the faceted geometric model $V/E/F/R$ (including toplogical relations between them), along with specification of the geometric and faceting tolerances ϵ_g and ϵ_f, respectively.

2.2 Sealing

Given the input described in Section 2.1, the goal is to seal facets in the model together, such that the facetings of the Faces, Edges, and Vertices satisfy the requirements of a cell complex. That is, the intersections of points, edges, and triangle facets should correspond to other points and edges in the faceting.

In this section, we present the sealing algorithm that is the primary subject of this paper. The algorithm is presented in its entirety here; proof of its success is discussed in the following Section.

The general structure of our algorithm is to work Face by Face, sealing the skin of the Face's faceting to the faceting of bounding Edges. We take advantage of the known topological relations between Faces and their bounding Edges and Vertices, and the correspondence between a Vertex, Edge or Face and its faceting. The only remaining assumption describes the geometric proximity of facet points, edges, and faces, with the geometric entities they are supposed to resolve.

We describe the sealing algorithm in two parts. First, the higher-level algorithm for sealing a Face faceting with that of bounding Edges is given in Algorithm 1. This part of the algorithm has three main parts: a) matching points on the skin of the Face faceting to the Vertex points; b) separating the edges on the skin of the Face faceting into arcs Ω^j, using the Vertex points as separators; and c) for each arc Ω^j, sealing the edges on that arc to those of a corresponding Edge that bounds the Face; this last step requires an algorithm *seal*, described in a following Algorithm. Part b) uses a function $d_{oriented}$ that computes the average distance between two ordered lists of facet edges. This distance function is computed as the summed distance between

parameterized points in both edge sequences normalized for edge length, *as the edges are traversed in their respective orders.* This traversal direction-aware measure will compute a significant distance between two sequences of edges that are exactly spatially coincident but ordered in opposite directions. This algorithm is necessary for distinguishing a number of pathological cases we have observed in the facetings of various real geometric models.

In the second part of the algorithm, shown in Algorithm 2, the task is to seal two sequences of facet edges. These sequences are ordered, such that they share the facet point(s) at their start point and, if distinct, their end point. Overall, this algorithm works by considering edges e_e and e_s from the Edge and skin sequences, respectively, that share a point, p_c; and sealing them together, either directly if the edges from either sequence are close enough in length, or by splitting an edge in either sequence then sealing to one of the resulting edges. A graphical depiction of the three cases is shown in Figure 2.

Algorithm 1 Face sealing algorithm.

I. $\forall F^i$
  a. $\forall \mathcal{V}^i \in_a \mathcal{F}^i$
    1. $find\ e \in \Omega F^i s.t.\ d(e, \mathcal{V}^i)\ min$
    2. $if\ d(p \in_{adj} e, \mathcal{V}^i)\ min,\ choose\ p$
    3. $else\ split\ e \rightarrow new\ p$
    4. $p\ \rightarrow V^i$
  b. $group\ \Omega F^i \rightarrow \Omega^j F^i\ using\ \ p \in \mathcal{V}^k \in_{adj} \mathcal{F}^i$
  c. $\forall \Omega^j F^i = e$
    1. $find\ E^k\ s.t.\ d_{oriented}(\Omega^j, E^k)\ min$
    2. $if\ E^k\ not\ sealed\ yet,\ E^k \rightarrow \Omega^j F^i$
    3. $else\ seal(\Omega^j F^i, E^k)$

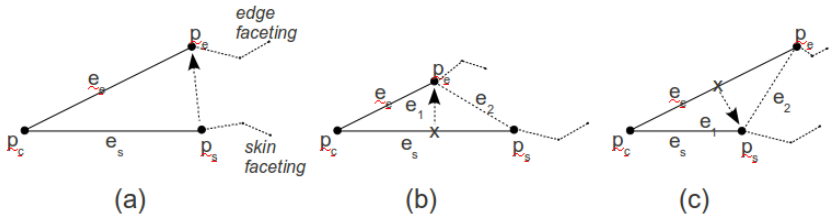


Fig. 2: Three cases for sealing edges from two sequences of facet edges. a) Facet edges are similar in length, and can be sealed directly; b) and c), split skin / edge facets, respectively.

Algorithm 2 Edge/skin sealing algorithm. $otherpoint(e, p)$ is the point adjacent to e not equal to p; for e_n, $p_n = next(\{e\}, e, p)$, e_n is the next edge along the ordered sequence of edges $\{e\}$ that shares p; $p_n = otherpoint(e_n, p)$.

Begin: seal arc $\Omega^j = \{p\}, \{e\}$, $E^k = \{p\}, \{e\}$
I. Initialize:
  a. $p_c = \Omega_{start}(\Omega^j)$
  b. $e_s,\ p_s = next(\Omega^j, -, p_c)$
  c. $e_e,\ p_e = next(E^k, -, p_c)$
II. while Ω^j not sealed
  a. if $p_e = p_s$
    1. $e_e \leftarrow e_s$
    2. Ω^j *sealed.*
  b. else if $d(p_s, p_e) \leq \epsilon_f$
    1. $p_e \leftarrow p_s$
    2. $e_e \leftarrow e_s$
    3. $p_c = p_e$
    4. $e_s,\ p_s = next(\Omega^j, e_e, p_c)$
    5. $e_e,\ p_e = next(E^k, e_e, p_c)$
  c. else if $d(p_e, e_s) \leq \epsilon_f$
    1. split e_s with p_e, $e_s \rightarrow e_1(p_c, p_e), e_2(p_e, p_s)$
    2. $p_c = p_e$
    3. $e_e,\ p_e = next(E^k, e_e, p_c)$
    4. $e_s = e_2$
    (p_s unchanged)
  d. else if $d(p_s, e_e) \leq \epsilon_f$
    1. split e_e with p_s, $e_e \rightarrow e_1(p_c, p_s), e_2(p_s, p_e)$
    2. $p_c = p_s$
    3. $e_s,\ p_s = next(\Omega^j, e_s, p_c)$
    4. $e_e = e_2$
    (p_e unchanged)

2.3 Proof

Previous methods to obtain watertight faceting have in some cases relied on user guidance. This type of guidance is impractical for the large models encountered in our work. Therefore, automation is critical to practical use of this algorithm. Although space does not allow a complete derivation of the proof of reliability of our algorithm, we describe here the general structure of this proof.

We develop the proof of reliability of this algorithm in four steps:

Can seal Vertices to points on Face skin: By definition, the Vertex facet point p is located on the geometric model, and the facet edges in ΩF^i are within ϵ_f of the geometric Edges on $\Omega\mathcal{F}$. Because F^i is a cell complex and so is the geometric model itself, each Vertex point p will therefore be within ϵ_f of ΩF^i as well. p will be closest to only one point of ΩF^i

because LFS $\gg \epsilon_f$. p can therefore always be sealed to ΩF^i, possibly after splitting one of the edges in ΩF^i to insert a point closest to p.

Can separate ΩF^i ***into*** $\Omega^j F^i$***:*** ΩF^i is a cell complex, with points corresponding to $\mathcal{V}\epsilon_a\mathcal{F}^i$ sealed to ΩF^i. Because the geometric model is also a cell complex, these points will separate ΩF^i into sequences of facet edges $\Omega^j F^i$, with each sequence bounded by one or two $p\epsilon V, \mathcal{V}\epsilon_a\mathcal{F}$.

$\Omega^j F^i$ ***corresponds to*** $E^k, \mathcal{E}^k\epsilon_a\mathcal{F}^i$***:*** $\Omega\Omega^j = V^l$, and F^i and the geometric model are cell complexes, each $\Omega^j F^i$ will correspond to one E^k, and E^k are distinct because of the preprocessing done before execution of the sealing algorithm.

Can seal $\Omega^j F^i$ ***with*** E^k***:*** The facet points on $\Omega^j F^i$ and E^k are within ϵ_g of the geometric model; the facet edges on $\Omega^j F^i$ and E^k are within ϵ_f of the geometric model; and $\Omega^j F^i$ and E^k are ordered consistently with each other. Therefore, similar to sealing the Vertices to ΩF^i, the facetings $\Omega^j F^i$ and E^k can be sealed together, possibly requiring splitting some of the edges on $\Omega^j F^i$ or E^k.

When all Face facetings have been sealed to the facetings of their bounding Edges, the whole model has been sealed.

3 Implementation

This algorithm is implemented in C++. Geometric models are initialized by reading their faceting and geometric topology through CGM [9] into MOAB [28]. MOAB represents the geometric model entities as entity sets in the mesh, with the entity sets containing the facets defining the entity (Region sets are empty, since there is no 3d mesh), and topological relations between entities represented using parent/child relations between the sets.

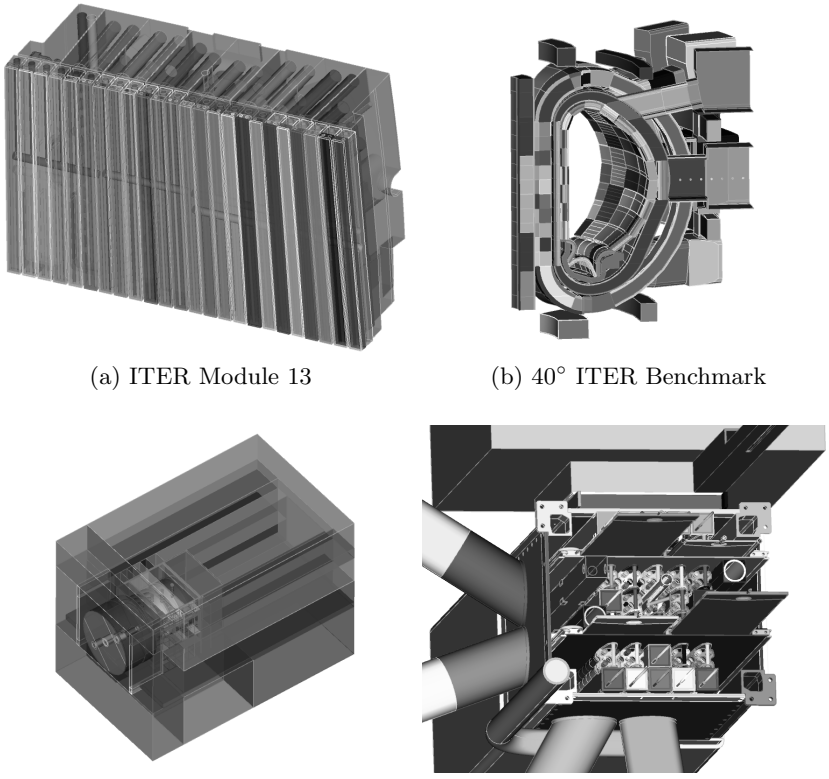
The sequences of facet edges used in the algorithm described earlier are represented only as ordered lists of facet points. Facet edges are not represented explicitly, since they are being modified frequently as sealing proceeds. Spatial searches are accelerated using MOAB's k-d tree decomposition.

The final results of sealing are represented in MOAB using the same geometric entity sets used for input. This representation can be output to various mesh formats, or can be left in MOAB for subsequent operations (e.g. facet coarsening and refinement to generate FE meshes).

4 Results

Test models displayed in Figure 3 and listed in Table 1 were chosen from previous radiation transport simulations. Models vary in complexity, with the most intricate having more than 29 million triangular facets. Before being sealed, five test models were already watertight by proximity. Although not

topologically the same, skin points of adjacent faceted surfaces were within the solid modeling engine's absolute tolerance of each other. These models were created using the ACIS solid modeling engine. Five test models did not originate in ACIS and were not watertight by proximity, due to file format translation, non-manifold surfaces, and imprecise modeling.



(a) ITER Module 13 (b) 40° ITER Benchmark

(c) FNG Benchmark (d) UW Nuclear Reactor

Fig. 3: Detailed CAD models were used to test the sealing algorithm.

Table 2 shows the number of Faces that failed to conform to their bounding Edges after sealing, for various models and facet tolerances. Note that for facet tolerances of 10^{-2} cm to 10^{-5} cm all but the ITER Benchmark model were watertight. This includes the default facet tolerance of 10^{-3} cm. The ITER Benchmark failed to completely seal because it contained many Faces with features smaller than the facet tolerance, created by imprinting. Failures occur if the facet tolerance becomes larger than the feature size, as in the 10^{-1} cm group.

Model	Geometric Entity			Facet Tolerance [cm]				
	Regions	Faces	Edges	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
UW Nuclear Reactor	2820	30237	65078	2.62	2.62	2.98	8.56	29.1
Advanced Test Reactor	2132	11827	22402	0.44	0.45	0.84	2.44	7.65
40° ITER Benchmark	902	9834	20485	0.32	0.78	2.07	8.76	16.3
ITER Test Blanket Module	71	4870	13625	0.07	0.08	0.12	0.38	1.57
ITER Module 4	155	4155	10255	0.29	0.29	0.34	1.07	2.89
ITER Module 13	146	2407	5553	0.28	0.29	0.50	2.54	8.65
FNG Fusion Benchmark	1162	4291	5134	0.11	0.11	0.14	0.46	1.14
ARIES First Wall	3	358	743	0.17	0.87	1.21	1.55	2.45
High Average Power Laser	15	139	272	0.15	0.47	0.53	0.61	0.88
Z-Pinch Fusion Reactor	24	95	143	0.05	0.29	0.99	1.17	1.53

Table 1: Geometric entity count and number of triangular facets [millions] as a function of facet tolerance [cm].

Model	Facet Tolerance [cm]				
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
UW Nuclear Reactor	1019	0	0	0	0
Advanced Test Reactor	88	0	0	0	0
40° ITER Benchmark	18	9	0	18	191
ITER Test Blanket Module	0	0	0	0	0
ITER Module 4	0	0	0	0	0
ITER Module 13	2	0	0	0	0
FNG Fusion Benchmark	63	0	0	0	0
ARIES First Wall	1	0	0	0	0
High Average Power Laser	0	0	0	0	0
Z-Pinch Fusion Reactor	3	0	0	0	0

Table 2: Number of face sealing failures as a function of facet tolerance [cm].

4.1 Triangle Count

The speed of computations performed on the FBM is affected by the number of triangles in the model. The number of triangles increases due to vertex-edge contraction (since triangle facets connected to the edge are split with the edge), but decreases due to vertex-vertex contraction of adjacent skin points of the same face. For the default facet tolerance of 10^{-3} cm, the change in the number of triangles ranged from 0% to 2%. Across the entire test suite of ten models and five facet tolerances, the change in the number of triangles ranged from -36% to 3%. A decrease in the number of triangles was common for the 10^{-1} cm facet tolerance, since at that tolerance many facet points on the same Face get merged.

During sealing, each Face imposes an additional constraint on adjacent Edges. If the faceted Edge itself were to be sealed, it too would impose an additional constraint on the sealing process. By replacing the faceted Edge with a corresponding skin arc as suggested in Algorithm 1, the number of

contraints on the edge is reduced by one. This decreases the number of additional triangles due to vertex-edge contractions.

4.2 Inverted Facets

When sealing, facets become inverted if a skin point is moved across an edge of the facet. This typically occurs with long, narrow facets. Each inverted facet is removed along with adjacent facets to create a polygon. The polygon is refaceted using the ear clipping algorithm as implemented in [29], with $O(n^2)$ time complexity, with n the number of points bounding the polygon. It is possible that the new facets are still inverted. If so, the polygon is iteratively expanded and refaceted until the new facets are no longer inverted. The polygon can only be expanded to the skin of the Face, so that the new facets are constrained to the skin. One can think of the Face skin as being owned by Edges instead of the face. Inverted facets that cannot be fixed by this method are a rare occurrence if the input model conforms to the input assumptions, as suggested by Table 3. Although the ear clipping algorithm is $O(n^2)$, n is typically small.

Model	Facet Tolerance [cm]				
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
UW Nuclear Reactor	272	0	1	0	13
Advanced Test Reactor	30	0	0	0	0
40° ITER Benchmark	7	7	4	0	10
ITER Test Blanket Module	0	0	0	0	0
ITER Module 4	0	0	0	0	0
ITER Module 13	2	0	0	0	0
FNG Fusion Benchmark	16	0	0	0	0
ARIES First Wall	0	0	0	0	0
High Average Power Laser	0	0	0	0	0
Z-Pinch Fusion Reactor	2	1	0	0	0

Table 3: The number of Faces containing inverted facets after sealing as a function of facet tolerance [cm].

4.3 Timing

The implementation was executed on an Intel Xeon 3 GHz processor with 64 bit Linux. The sealing algorithm is intended to be used as a preprocessing step for applications that use FBMs. Timing results are shown in Table 4. Results do not include file reading and writing, because as part of an existing application, the use of this algorithm will not incur additional runtime due to file reading and writing. In general, sealing required less than one minute for most models and facet tolerances.

Model	Facet Tolerance [cm]				
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
UW Nuclear Reactor	136	65	64	156	587
Advanced Test Reactor	93	16	27	76	235
40° ITER Benchmark	6	12	38	71	236
ITER Test Blanket Module	15	9	9	14	30
ITER Module 4	10	8	8	23	67
ITER Module 13	6	5	6	19	67
FNG Fusion Benchmark	7	4	4	9	29
ARIES First Wall	1	3	5	13	36
High Average Power Laser	1	1	2	5	25
Z-Pinch Fusion Reactor	1	1	2	4	12

Table 4: The time [seconds] to seal each model as a function of facet tolerance [cm].

4.4 Application Example: Particle Tracking

This algorithm was developed, in part, to seal faceted CAD models for Monte Carlo radiation transport. One of the causes of lost particles is leakage between Faces. Figure 4 shows the lost particle fraction for each model before and after sealing. The default facet tolerance of 10^{-3} cm was used for all models. The ITER Benchmark, Test Blanket Module, Module 4, and Module 13 models lost significantly fewer particles after sealing. Sealing did not significantly affect the UW Nuclear Reactor, Advanced Test Reactor, FNG Fusion Benchmark, ARIES First Wall, High Average Power Laser, and Z-Pinch Reactor models. This reflects the input models, of which half were already watertight by node proximity.

Sealing did not eliminate lost particles. The first three lost particles of each sealed model were investigated. In each case the particles became lost because of a specific defect in the particle tracking algorithm, unrelated to watertightness. Improvement of the particle tracking algorithm is an active research topic.

5 Conclusions

A tool was developed to make existing faceted models watertight without human intervention. Faces were sealed to their bounding edges using vertex-vertex and vertex-edge contraction. Because sealing progresses by Face, this algorithm naturally supports non-manifold geometry. Ten CAD models were tested over a 4 decade range of facet tolerances. Models were successfully sealed if the input assumptions of the algorithm were met. It is assumed that the facet tolerance is less than the feature size and greater than the merge tolerance. On average, sealed models contained more triangles than unsealed models as a result of vertex-edge contraction. Sealing can create inverted facets, with most occurring when input assumptions are not met. Timing

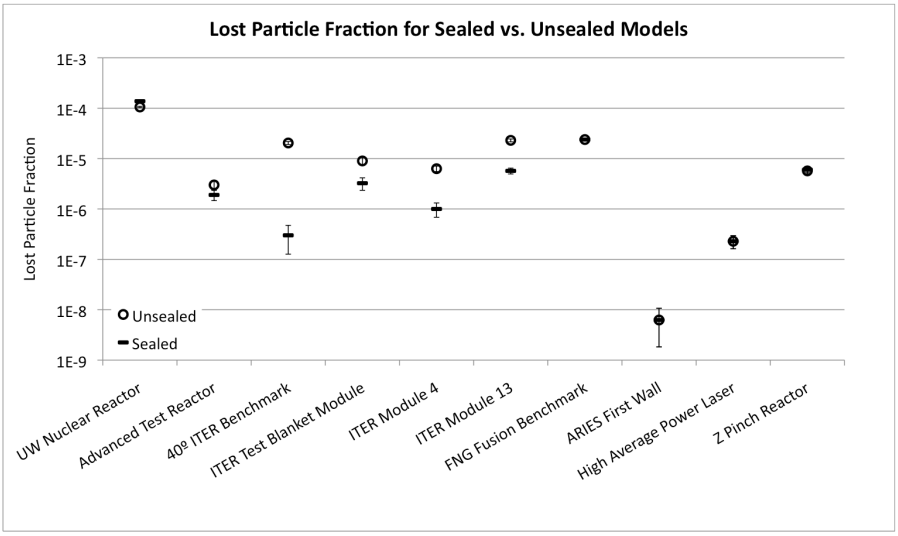


Fig. 4: Lost particle fraction before and after sealing each model. Error bars indicate one standard deviation.

results show that the algorithm is fast enough to be used in preprocessing for applications that use faceted geometry. A particle tracking application was used to test models before and after sealing. One cause of lost particles, leakage between unsealed Faces, was eliminated through application of the sealing algorithm. The cause of remaining lost particles was determined to be roundoff and precision issues in the ray-triangle intersect computation; this is the topic of further research.

Acknowledgement

The authors thank Jason Kraftcheck for his advice and assistance using MOAB. This work was supported, in part, by Sandia National Laboratories and the US ITER Project through Sandia contracts #579323 and #866756. This work was also supported by the US Department of Energy Scientific Discovery through Advanced Computing program under Contract DE-AC02-06CH11357. Argonne National Laboratory (ANL) is managed by UChicago Argonne LLC under Contract DE-AC02-06CH11357.

The submitted manuscript has been created in part by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in

References

1. 3D Systems Inc. Stereolithography Interface Specification, 1988.
2. E. Bchet, J. C. Cuilliere, and F. Trochu. Generation of a Finite Element MESH from Stereolithography (STL) Files. *Computer-Aided Design*, 34(1):1 – 17, 2002.
3. D. Rypl and Z. Bittnar. Generation of Computational Surface Meshes of STL Models. *Journal of Computational and Applied Mathematics*, 192(1):148 – 151, 2006. Special Issue on Computational and Mathematical Methods in Science and Engineering (CMMSE-2004).
4. Timothy J. Tautges, Paul P.H. Wilson, Jason A. Kraftcheck, Brandon M. Smith, and Douglass L. Henderson. Acceleration Techniques for Direct Use of CAD-Based Geometries in Monte Carlo Radiation Transport. In *Proc. International Conference on Mathematics, Computational Methods, and Reactor Physics*, 2009.
5. Spatial Corporation. Spatial Product Documentation. http://doc.spatial.com/index.php/Refinements#Surface_Tolerance, 2009.
6. Open CASCADE technology, 3D modeling & numerical simulation. http://www.opencascade.org/.
7. A. Sheffer, T. D. Blacker, and M. Bercovier. Virtual Topology Operators for Meshing. *International Journal of Computational Geometry and Applications*, 10(2), 2000.
8. Sandia National Laboratories. CUBIT Geometry and Mesh Generation Toolkit. http://cubit.sandia.gov, 2010.
9. Timothy J. Tautges. The Common Geometry Module (CGM): a Generic, Extensible Geometry Interface. *Engineering with Computers*, 17(3):299–314, 2001.
10. Stephan Bischoff and Leif Kobbelt. Structure Preserving CAD Model Repair. *Eurographics*, 24(3):527–536, 2005.
11. Tao Ju. Fixing Geometric Errors on Polygonal Models: A Survey. *Journal of Computer Science and Technology*, 24(1):19–29, 2009.
12. Herbert Edelsbrunner and Ernst Mücke. Three-Dimensional Alpha Shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.
13. Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A New Voronoi-Based Surface Reconstruction Algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 415–421, New York, NY, USA, 1998. ACM.
14. T. M. Murali and Thomas A. Funkhouser. Consistent Solid and Boundary Representations from Arbitrary Polygonal Data. In *I3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 155–ff., New York, NY, USA, 1997. ACM.
15. Tao Ju. Robust Repair of Polygonal Models. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 888–895, New York, NY, USA, 2004. ACM.
16. Jan H. Bohn and Michael J. Wozny. A Topology-Based Approach for Shell Closure. *Geometric Modeling for Product Realization*, pages 297–319, 1993.

17. Gill Barequet and Micha Sharir. Filling Gaps in the Boundary of a Polyhedron. *Computer Aided Geometric Design*, 12:207–229, 1995.
18. Gill Barequet and Subodh Kumar. Repairing CAD Models. In *VIS '97: Proceedings of the 8th Conference on Visualization '97*, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.
19. G. T. Klincsek. Minimal Triangulations of Polygonal Domains. *Combinatorica*, 79, 1980.
20. X. Sheng and I.R. Meier. Generating Topological Structures for Surface Models. *IEEE Computer Graphics and Applications*, 15(6):35–41, 1995.
21. André Guéziec, Gabriel Taubin, Francis Lazarus, and William Horn. Converting Sets of Polygons to Manifold Surfaces by Cutting and Stitching. In *VIS '98: Proceedings of the Conference on Visualization '98*, pages 383–390, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.
22. P. Borodin, M. Novotni, and R. Klein. Progressive Gap Closing for Mesh Repairing. *Advances in Modelling, Animation and Rendering*, pages 201–21, 2002.
23. International Meshing Roundtable. *Stitching and Filling: Creating Conformal Faceted Geometry*, 2005.
24. Ferenc Kahlesz, Ákos Balázs, and Reinhard Klein. Multiresolution Rendering by Sewing Trimmed NURBS surfaces. In *SMA '02: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, pages 281–288, New York, NY, USA, 2002. ACM.
25. Oleksiy Busaryev, Tamal K. Dey, and Joshua A. Levine. Repairing and Meshing Imperfect Shapes with Delaunay Refinement. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 25–33, New York, NY, USA, 2009. ACM.
26. C. S. Chong, A. S. Kumar, and H. P. Lee. Automatic Mesh-Healing Technique for Model Repair and Finite Element Model Generation. *Finite Elements in Analysis and Design*, 43:1109–1119, 2007.
27. Jim Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18(3):548–594, 1995.
28. Timothy J. Tautges, Ray Meyers, Karl Merkley, Clint Stimpson, and Corey Ernst. MOAB: A Mesh-Oriented Database. Technical report, Sandia National Laboratories, 2004. SAND2004-1592.
29. Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 1998.