Repairing and Meshing Imperfect Shapes with Delaunay Refinement

Oleksiy Busaryev
busaryev@cse.ohio-state.edu

Tamal K. Dey
tamaldey@cse.ohio-state.edu

Joshua A. Levine
levinej@cse.ohio-state.edu

Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210 USA

ABSTRACT

As a direct consequence of software quirks, designer errors, and representation flaws, often three-dimensional shapes are stored in formats that introduce inconsistencies such as small gaps and overlaps between surface patches. We present a new algorithm that simultaneously repairs imperfect geometry and topology while generating Delaunay meshes of these shapes. At the core of this approach is a meshing algorithm for input shapes that are piecewise smooth complexes (PSCs), a collection of smooth surface patches meeting at curves non-smoothly or in non-manifold configurations. Guided by a user tolerance parameter, we automatically merge nearby components while building a Delaunay mesh that has many of these errors fixed. Experimental evidence is provided to show the results of our algorithm on common computer-aided design (CAD) formats. Our algorithm may also be used to simplify shapes by removing small features which would require an excessive number of elements to preserve them in the output mesh.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*; J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

Keywords

Shape repair, Delaunay mesh generation, Topology, Piecewise-smooth complexes

1. INTRODUCTION

A typical computer-aided design (CAD) system allows a designer to build a shape by modeling the surface patches that comprise its boundary. Typically this boundary representation (B-Rep) defines a shape as a collection of surface patches (e.g. NURBS patches). Ideally the CAD software generates a watertight shape which is then stored in a format that maintains both a geometric description of each patch as well as topological connectivity between patches.

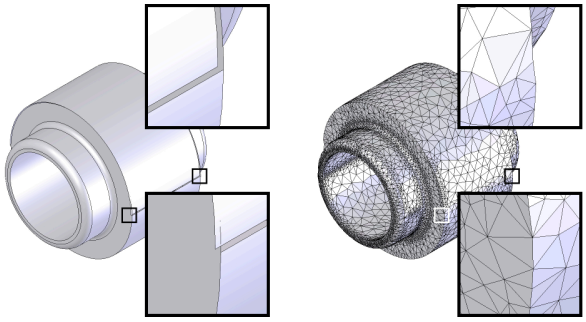

Figure 1: Repairing a CAD model. Left: this STATOR **model has gaps and overlaps between different faces. Right: these are merged by our algorithm.**

After the design process, the shape is then passed to an exterior application; many of which require again converting the shape to a suitable format. One common case is performing finite element analysis, but other applications include visualization or texture mapping [10]. At this stage generally a mesh is constructed using primitive elements such as tetrahedra or hexahedra.

Due to numerical problems, imprecise design, software idiosyncrasies, or data exchange issues, the surface patches produced at the CAD step may abut within unpredictable tolerances, resulting in gaps, overlaps, or intersections [19]. Moreover, shapes may be represented with duplicate geometric information that matches under any level of tolerance, but the duplication causes loss of topological information. For example, when CAD software produces two trimmed NURBS patches that meet, the shared trim curve may be stored independently at each patch. These errors are often visually imperceptible to the designer, but they usually impede any automatic mesh generation step as typical mesh generators require watertight, consistent representations as an input. Repairing inconsistencies in the object representation in order to achieve global continuity of the model boundary is a well-known challenge in the CAD community [5].

1.1 Contributions

We aim to automatically repair imperfect models while producing quality Delaunay meshes that approximate the

GDSPM09 October 2009 San Francisco, California USA

input shapes. With the exception of a recent work by Chong *et al.* [9], all past approaches treat this repair step as a decoupled component from the mesh generation step. We generate a mesh with repaired geometry directly from an imperfect CAD model without the need for preprocessing the model.

Our approach to the repair problem draws upon an idea used recently in a Delaunay meshing algorithm [12]. This algorithm models the input domain as a piecewise smooth complex (PSC), a collection of smooth surface patches where two or more meet at curves in non-smooth configurations provided they still satisfy the usual properties for being a complex. To ensure these curves are preserved in the output mesh, they are covered with a set of balls whose centers sample the curves. These protecting balls are later turned into weighted points and Delaunay refinement is run using a weighted version of the Delaunay triangulation.

We use this protection idea with certain modifications. First we generate a sufficiently small set of protecting balls to cover all curves in the imperfect input shape. We then merge pairs of protecting balls which violate criteria controlled by a user-specified distance tolerance γ, replacing each pair with the smallest ball containing the two. Redundant balls are then removed and the algorithm is repeated on the remaining set of balls.

Upon termination we have a set of protecting balls covering each input curve as well as all gaps, overlaps, and intersections with diameter smaller than the parameter γ. The centers of these balls sample the curves in the repaired PSC, and each ball's radius is large enough to cover nearby imperfections of the input. At this point, we insert each ball as a weighted point in a Delaunay triangulation and Delaunay refinement follows to mesh the repaired PSC input.

Using this approach, we can successfully generate meshes despite many of the types of inconsistencies commonly produced with CAD software. When the user provides the tolerance γ in a correct range, our algorithm produces a mesh with desired qualities (e.g. watertight) that can be used for later applications. In addition, since our meshing algorithm handles boundaries and non-manifold features, the algorithm *always* terminates for every $\gamma \geq 0$ when the input has no intersecting patches.

2. RELATED WORK

2.1 Shape Repair

Many techniques for repairing models have been produced by the CAD, meshing, and computer graphics communities. The solid modeling community offers the notion of a regular set, or more recently ϵ-regular set [24], to provide a foundation for some of the techniques. Broadly, the approaches to repair can be categorized into two groups. The first group repairs the input shape by performing local modifications that merge and fix incorrect surface patches. The second uses volumetric techniques to globally compute a new shape without errors. Both approaches have been used for input shapes represented in typical B-Rep formats (e.g. STEP and IGES) as well as polygonal approximations of B-Reps.

The local approaches are similar to our algorithm in that they often have some scheme to modify elements that are co-located within a user-specified tolerance. For example, surface boundary curves may be merged using various rules. Some approaches score and merge curve segments [1] or first detect [23] and merge boundary curves [15, 19, 28]. Other approaches stitch the gaps between boundary curves of surface elements [2, 14]. Some researchers move or merge vertices of polylines approximating the boundary curves [4, 9, 26]. Others suggest combinations of the three approaches [22]. Local volumetric techniques have been used to repair the imperfect regions near curves without globally recomputing surface patches [3]. Shape repair by editing a virtual topological structure for a model has also been investigated [25].

In contrast to the local approaches, volumetric techniques offer global methods to shape repair. These approaches build a partition of the volume containing the shape and use the subdivided volume to build a new B-Rep. Murali and Funkhouser use the input surface patches to build a BSP-tree representing the shape [20]. Other approaches usually build either a uniform or adaptively sized Cartesian grid containing the volume [16, 17, 18, 21, 29]. After building the grid, a new shape is extracted by using either isosurfacing techniques or projecting a subset of grid vertices back to the input. These approaches can handle some types of inconsistencies that often cause the local approaches to fail, such as two deeply overlapping surface patches. However, they can have trouble preserving sharp features in the input as well as handling inputs with non-manifold features.

2.2 PSC Meshing

Our input shape is a representation of a PSC $\mathcal{D}$, possibly with errors. A PSC is a collection of k-dimensional *faces* called vertices (0-faces), curves (1-faces), and patches (2-faces). Each k-face in the PSC is defined by its unique geometry (i.e. for surface patches a plane, quadric, or b-spline surface; for curves a line or b-spline; and for vertices their coordinates in Euclidean space). In addition, each k-face has connectivity information for its boundary—a set of curves for the boundary of each patch and a set of vertices for the boundary of a curve.

We use $\mathcal{D}_i$ to denote the set of i-faces in a PSC and $\mathcal{D}_{\leq i}$ to represent the union $\mathcal{D}_0 \cup \mathcal{D}_1 \cup \ldots \cup \mathcal{D}_i$. If $\mathcal{D}$ is a PSC, it satisfies the usual conditions for being a complex: if any two $(i+1)$-faces $\sigma, \sigma' \in \mathcal{D}_{i+1}$ intersect, their intersection is a union of (lower dimensional) faces in $\mathcal{D}_{\leq i}$ on the boundary of both σ and σ'.

Delaunay refinement relies on the idea of Chew's furthest point strategy [8] to incrementally build a Delaunay mesh for an input domain. The basic approach is simple: to construct a Delaunay mesh for some domain, repeatedly insert points from the domain until the mesh satisfies a set of conditions. For different types of domains, one must prove that the set of conditions checked by the algorithm ensures the desired properties in the output mesh. The most significant burden is to prove that the algorithm terminates. If inserted points are chosen carefully so that a lower bound on their pairwise distances is maintained, a packing argument guarantees termination since only a finite number of points can be inserted by the algorithm.

Recently, Cheng, Dey, and Ramos proposed a unique approach for meshing PSCs [7] based on Delaunay refinement. One novelty of this technique is that all non-smooth and non-manifold features (the faces in $\mathcal{D}_{\leq 1}$) are protected throughout the meshing algorithm by sampling them with balls. These protecting balls are turned into weighted points and Delaunay refinement is performed using a weighted Delaunay triangulation [13]. This algorithm was the first certified algorithm that could in theory successfully handle PSC

inputs; however, since it employs some expensive numeric computations, Dey and Levine [12] devised a more practical version of the algorithm. Their version reduces the refinement triggers to a single "topological disk" test that is combinatorial in nature.

3. ALGORITHM

We present a Delaunay refinement algorithm that handles inputs that are *almost* PSCs. This algorithm is guided by a user parameter $\gamma \geq 0$ that acts as a tolerance for error in the input. The output is a Delaunay mesh for some PSC that is presumably represented by the possibly flawed input PSC.

3.1 Input and Output

Let $\mathcal{D}$ denote the (potentially erroneous) input our algorithm meshes. For our algorithm, $\mathcal{D}$ need only be a collection of surface patches, curves, and vertices, but it is not required to have the usual properties for a complex. In particular, the boundary curves of patches may not have geometry which matches the patch they bound; and "adjacent" patches may not share the same boundary curve. Specifically, for a given $\gamma \geq 0$, our algorithm produces a mesh that approximates $\mathcal{D}$ under the following types of errors (some of which are shown visually in the top row of Figure 2):

- Gaps (not necessarily uniform) of any size between the boundary curves of nearby patches in $\mathcal{D}$.
- Misaligned boundary curves; endpoints may not match or pieces may only partially match.
- Patch intersections between nearby patches provided they intersect close to some boundary curve.
- Duplicated boundary curves between patches that are geometrically equivalent.

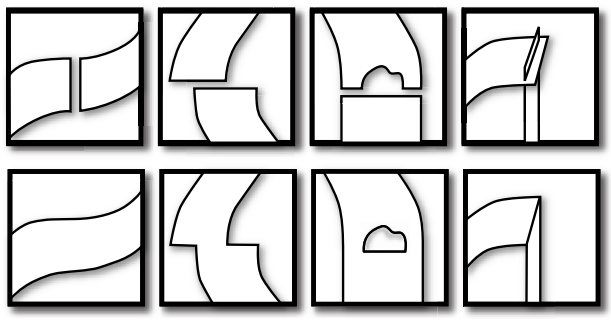

Figure 2: Common CAD model errors. From left to right: uniform gaps, misaligned patches, non-uniform gaps from trimmed curves, and intersecting patches. The bottom row shows how these are fixed. We merge at small gaps and intersecting regions. For non-uniform gaps some boundary elements may not get merged.

Our algorithm assumes that the input provides a usable geometric representation of each surface patch σ, but it ignores any connectivity information between patches through the boundary curves of each σ. We use the original geometric information of each curve in $\mathcal{D}$ to devise a new set of curves and vertices to mesh. These curves are covered by a set of protecting balls of size γ or greater. Faces from the input $\mathcal{D}$ that intersect the same protecting regions are merged in our final mesh, effectively fixing small errors from the input (see the bottom row of Figure 2).

3.2 Structures

In this section we review some of the basic geometric structures which our algorithm uses. Refer to [11, 13] for additional details.

Let S be a point set sampled from the underlying space of $\mathcal{D}$. We associate each point $p \in S$ with a real-valued weight $r \geq 0$; alternatively, the weighted point p can be represented as a ball with radius r centered at p. The weight r skews the distance metric locally with respect to p. The *squared weighted distance* of any point $x \in \mathbb{R}^3$ from p is given by $\mathrm{d}(x,p) = \|x-p\|^2 - r^2$. Under this new distance metric, we define the Voronoi diagram, $\mathrm{Vor}\, S$, and Delaunay triangulation, $\mathrm{Del}\, S$. These structures follow a similar rule as their unweighted counterparts with distance between points computed using weighted distance [13].

To generate a Delaunay mesh for $\mathcal{D}$ we take special subcomplexes of $\mathrm{Vor}\, S$ and $\mathrm{Del}\, S$. In particular, we are interested in elements of $\mathrm{Vor}\, S$ and $\mathrm{Del}\, S$ that provide an approximation of $\mathcal{D}$. For any space $\mathbb{X}$, the *restricted Voronoi diagram* is the collection of intersections of Voronoi elements with $\mathbb{X}$. Its Delaunay counterpart, the *restricted Delaunay triangulation* of S with respect to $\mathbb{X}$, is denoted $\mathrm{Del}\, S|_{\mathbb{X}}$. A simplex $\tau \in \mathrm{Del}\, S$ is in $\mathrm{Del}\, S|_{\mathbb{X}}$ if its dual Voronoi face V_τ has a nonempty intersection with $\mathbb{X}$. For any $\sigma \in \mathcal{D}$, $\mathrm{Del}\, S|_\sigma$ denotes the Delaunay subcomplex restricted to σ. Figure 3 illustrates an example restricted Delaunay triangulation in two-dimensions.

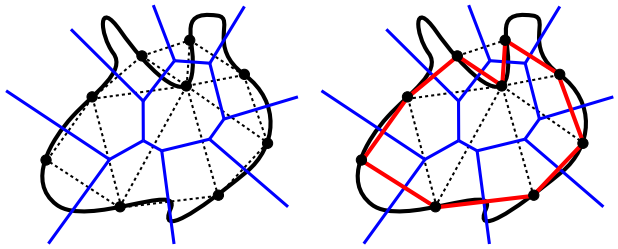

Figure 3: Restricted Delaunay triangulations. The left image shows a curve with nine sample points. Overlaid is the Voronoi diagram in blue and the Delaunay triangulation with dotted lines. On the right the restricted Delaunay triangulation is highlighted in red.

During Delaunay refinement, restricted Delaunay triangulations act as a mesh of a working approximation of the input. As a subset of the Delaunay triangulation, they inherit the Delaunay property. Since in three dimensions $\mathrm{Del}\, S$ contains many simplices spanning the convex hull of S, $\mathrm{Del}\, S|_{\mathcal{D}}$ ultimately filters a set that approximates the input $\mathcal{D}$.

3.3 Algorithm Overview

Our algorithm proceeds in three different phases. First, we PROTECT all curve elements that are extracted from $\mathcal{D}$. Guided by a tolerance parameter γ, we then REPAIR the information extracted to form the set $\mathcal{D}_{\leq 1}$ that we mesh. A

new set of balls is created to cover $\mathcal{D}_{\leq 1}$ which simultaneously covers up the gaps and intersections that are deemed errors in the geometry of $\mathcal{D}$. Finally, we REFINE a Delaunay mesh which approximates $\mathcal{D}$ by inserting additional points for each surface patch in $\mathcal{D}$.

4. PROTECTING CURVES

4.1 Protecting an Error-free Input

Our algorithm first tries to protect the regions around curves specified by $\mathcal{D}$ as best as possible. The algorithm in [12] requires that the set of curves for a PSC $\mathcal{D}$ are sampled using a set of balls that satisfies certain properties. In the algorithm, the union of these balls becomes a protected region; one guarantee is that no point is inserted within this region during the meshing algorithm.

In general, computing a set of protecting balls satisfying the desired protection properties would require a number of expensive numeric computations involving feature sizes and distances between curves. We use a recursive approach to ease this computational burden. The set of balls needs to satisfy three properties [12] to protect the curves of a PSC:

- (Complete coverage) Each curve must be covered completely with a sequence of balls, the first and last protecting balls for a curve located at the curve's endpoints. Two balls are called *adjacent* if they are consecutive in the sequence.

- (Deep coverage) Any two adjacent balls along a curve must intersect deeply, but not contain each others centers.

- (Separation) Let s be the piece of a curve contained in ball b. If any other ball b' also contains somes point from s, then b' must be adjacent to b.

These properties are justified because they will ensure that all curve features are preserved in the final mesh as sequences of edges. Instead of computing an exact set of ball radii which matches the feature sizes of the curves, we pick an initial set of ball radii which may potentially be too large. The meshing algorithm then drives the ball sizes to match the features as needed.

To satisfy these properties, balls at the endpoints of each curve (the elements of $\mathcal{D}_0$) are computed so that no two endpoint balls intersect. Next a walking procedure is used to cover up each curve with balls that satisfy the first and second properties. This procedure steps along the curve from a source endpoint to its destination, placing balls which deeply intersect the last placed ball. The radius of each ball is equal to the minimum radius of the two endpoint balls. When this walk comes near its destination endpoint, a terminating ball is placed which intersects both the ball at the destination as well as the previously placed ball.

After walking to cover the curves, non-adjacent balls may intersect each other, potentially violating the separation property, so recursively they are shrunk and the gap between consecutive balls is filled with smaller balls which still deeply cover the curve. When balls are small enough, they satisfy the separation property while still covering the curve completely.

4.2 Protection in the Presence of Errors

Since our input may have errors, the system of curves that the input specifies may not form a 1-dimensional PSC. In particular, if curves are duplicated or very close to each, our separation procedure may recurse indefinitely. To allow our algorithm to terminate, we make two modifications to the separation algorithm:

1. Balls centered at curve endpoints are limited to have radii no smaller than γ.

2. When pairs of balls are selected to be separated, the centers of the pair must be at least a distance of γ apart.

The first condition ensures that the endpoint balls do not become too small. The second prevents separation from recurring indefinitely. In fact, any pair of balls which are a distance of γ or less will be selected as candidates for merging together in the REPAIR stage which follows. Hence, attempting to separate them further is unnecessary work.

5. MODEL REPAIR

Following the PROTECT stage, geometry repair is performed in two iterative phases. During an individual phase, pairs of balls are first nominated for merging according to phase-specific different rules. To find these pairs efficiently, we sweep the space with a plane that searches the set of balls which have not been paired. After no pairs remain which can be nominated, all selected pairs are merged.

Each ball pair is merged by creating a new ball which is the smallest possible ball containing the two. Thus, the size of merged balls is dependent on the gaps between curves in the original input. This larger ball is added to the set of balls, and the original pair is removed. This new ball is given topological information which specifies that it covers a new, unique curve. This new curve is set incident to the 2-faces which were incident to the curves that the balls in the pair were protecting. After all nominated pairs are merged, this updated set of balls is then checked again to see if any balls in the new set satisfy the conditions for merging. Geometry repair is illustrated in Figure 4.

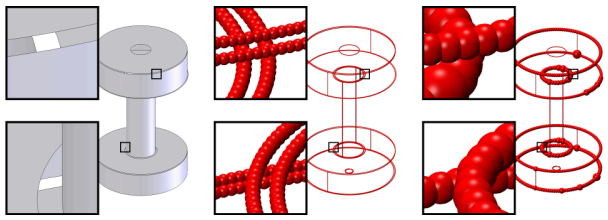

Figure 4: The merging process. Left: the DUMBBELL model has gaps shown in the zoomed regions. Middle: the initial protecting ball set produced by our algorithm. Right: the final set produced by approach.

5.1 Merging Nearby Curves

Intuitively, the final set of protecting balls will act as a "glue" between patches; the balls should be chosen so that adjacent patches meet within the balls as they would on a

shared boundary curve. This property is exploited to cover up gaps and intersections between boundary patches.

For example, when a small gap exists between two curves, the protection step would have generated two sets of tiny, separated balls to preserve the gap. The gap is removed by merging the curves using a single set of larger balls that cover up both curves and the space between them. Similarly, for two surface patches with a slight intersection near their boundary curves, the region of intersection is covered with a single set of balls that erases the intersection.

To combine nearby curves, the first phase performs ball merging to combine balls which have centers within distance γ. To prevent the merge process from eliminating surface elements, we also require that each ball in the pair lies on curve elements which have disjoint sets of adjacent 2-faces.

Once there is no ball pair with centers within distance γ, we begin the second phase. This phase enforces that the protection conditions are still maintained. Since balls have been merged, along a single curve they may now be large enough to contain each others centers (violating the deep coverage condition). We again proceed iteratively. First, we nominate pairs of balls such that at least one ball in the pair contains the center of the other. We then merge all such identified pairs, discarding old balls, and repeatedly check for more violating pairs.

5.2 Computing A New Set of Curves

After all violating pairs of balls are merged, the final set of balls we have produced protects a new set of curves which we treat as $\mathcal{D}_{\leq 1}$. However, while the centers of these balls are a sample for $\mathcal{D}_{\leq 1}$, we do not yet explicitly know which centers are adjacent to each other. To compute a polygonal approximation for each curve in $\mathcal{D}_1$, we use the intersections between balls to decide connectivity between the ball centers. We again employ a sweep algorithm to determine which pairs of balls intersect. Each intersecting pair forms an edge for some curve. To determine the sequence of edges that make up a curve, we first search the set of balls and mark all centers which are incident to more than two edges as elements of $\mathcal{D}_0$. Next, to compute the new elements of $\mathcal{D}_1$ we take sequences of remaining edges which connect two elements of $\mathcal{D}_0$.

On curves that were unaffected by this merging process, some of the protecting balls may now be quite small compared to those at their endpoints. Recall that our covering algorithm covers each curve with balls whose radius equals the minimum radius of the balls at the endpoints of the curve. Thus, some unaffected curves were initially protected with balls that were sized relative to gaps which have been removed. After merging, the balls at the endpoints are not constrained by these gaps, and we can afford to cover these curves with larger balls. To generate a sparser set of balls, we recompute the size of endpoint balls and rerun the routine from the PROTECT stage on the curves that do not contain merged balls. An example of the protecting ball sets and the meshes generated by our algorithm with and without this additional protection phase is shown in Figure 5.

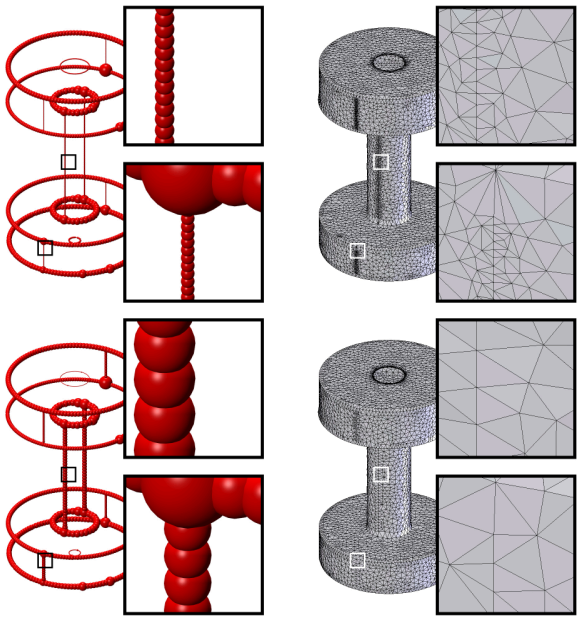

Figure 5: Reprotection helps reduce the final number of samples used to cover the curves. Top: our initial set of repaired balls produces a dense sampling along the vertical edge DUMBBELL. Bottom: after reprotection the number of samples better matches the density at other curves.

6. MESHING SURFACE PATCHES USING REFINEMENT

After the PROTECT and REPAIR stages, we take the set of protecting balls and insert them as the initial set S of weighted points in a weighted Delaunay triangulation. We generate a Delaunay mesh for $\mathcal{D}$ using Delaunay refinement with insertions to the set S triggered by conditions similar to the algorithm in [12]. This algorithm computes $\mathrm{Del}\, S|_{\mathcal{D}}$ and then performs various checks on the triangulations. Each check that fails either splits a ball or inserts a new point to correct the topology and geometry of the mesh. These points are computed as the intersection of Voronoi edges with the input surface patches.

To discover the correct topology, the refinement algorithm iteratively checks a *disk condition* which requires that the neighborhood of each vertex p in $\mathrm{Del}\, S|_{\mathcal{D}}$ is a topological disk in each patch. Vertices which lie on boundary curves are allowed a disk for each adjacent patch. When a vertex fails to satisfy the disk condition, a point is inserted which invalidates the Delaunay property of at least one of its neighbor triangles. Since feature size is never computed, some protecting balls may cause the disk test to fail because they are too large. If the point to insert is of distance less than the radius of the largest protecting ball, the protecting ball is split instead.

The algorithm is also parameterized by a user variable λ which controls the size of the triangles in the mesh. The algorithm iteratively inserts a point to split the largest triangle with circumradius greater than λ. It is guaranteed that for any λ, the algorithm terminates and outputs a mesh homeomorphic to some PSC. When λ is chosen small enough, the output mesh will be homeomorphic to the desired PSC.

When all vertices satisfy the disk test and all triangles are of size λ or less, the algorithm terminates. It then outputs the restricted Delaunay triangulation as the output mesh.

7. REPAIR RESULTS

We implemented our repair and meshing algorithm in C++ using the CGAL library [6]. The input to our software is a model in STEP format. Some additional experimental results are shown in Figures 7 and 8. Timings to generate these meshes are shown in Table 1.

Dataset	Protect	Merge	Meshing	# of faces
Body	2.969	41.764	7155.22	45092
Bracket	0.25	5.842	1779.47	15658
Cap	0.672	12.5	8709.92	37742
Corner	0.203	5.281	1376.13	19246
Dumbbell	0.094	2.844	1557.09	14334
Rear Cover	0.219	1.25	3040.16	30590
Snape Ring	0.219	6.988	1672.45	14452
Stator	0.141	0.563	897.309	17606

Table 1: Timings for protection, ball merging, and and mesh generation. All times are in seconds.

All eight of the models in the experiments contained various types of errors. In particular two of the models (Rear Cover and Stator) have non-uniform gaps that have been merged. Four of the models (Body, Corner, Rear Cover, and Stator) have intersecting curves that have been merged with balls large enough to overlap the intersection. The remaining models (Bracket, Cap, Dumbbell, and Snape Ring) have uniformly sized gaps which have been also been merged.

In regards to the time to mesh, our algorithm requires a primitive to compute intersections between dual Voronoi edges of Delaunay triangles and the input shape. This computation is commonly used for smooth surface meshing, and is known to be one of the more expensive computations used (in addition to the expense of computing the Delaunay triangulation). We use a library of our own design for intersections, but are exploring the possibility of using other libraries such as SISL [27].

For general B-Rep objects we found the intersection computation to be one of the major bottlenecks in the mesh generation step. Our computation first computes the intersection points between each Voronoi edge and each (untrimmed) patch. For toroidal and cylindrical patches, this computation requires solving quartic and quadratic equations respectively. The main expense is then to determine whether these intersection points lie within the sampled set of boundary curves for the trimmed surface patch, hence it is dependent on the sampling density of the curve features.

8. MODEL SIMPLIFICATION

Our repair and meshing algorithm can also be used to simplify inputs with certain types of small features. The performance of downstream applications which use meshes is often dependent on the size of the mesh. For some applications, a mesh which sacrifices geometry by using fewer elements is desirable. For example, a mesh which approximates a coarse version of an input shape with less elements can first be used for rapid prototyping needs, and then if necessary a second, finer mesh can be generated to improve accuracy of the computation.

Since our algorithm already merges curve features to repair gaps and intersections, we can use the same approach to merge curves, and consequently remove features, that a user does not want to keep in the final mesh. Many common shapes have grooves which are modeled as skinny toroidal or cylindrical patches. Similarly, chamfered or beveled corners have a skinny planar piece with two boundary curves very near to each other. Preserving these small geometric features requires an excess number of mesh elements, often only needed for an aesthetic purpose.

If the merging parameter γ is set high enough, the balls which cover the boundary curves of a single patch are merged to create a set of balls which cover the patch completely. During the merging step we track patches on which this occurs and remove any such patch from the final mesh. The result is a mesh which approximates the input shape with fewer total elements. An example where we remove grooves on a cylinder to simplify the shape is shown in Figure 6.

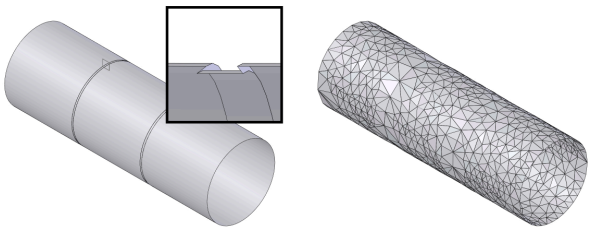

Figure 6: Removing small features from a cylindrical model. Left: the original shape has two small grooves (four curves each) which would require a large number of samples to preserve. Right: our algorithm merges the curves and meshes a simplified shape instead.

9. CONCLUSIONS

We propose a novel approach for repairing and simultaneously generating a Delaunay mesh of CAD models with inconsistencies due to design mistakes, software problems, or processing issues. Experimental evidence shows that our algorithm effectively handles sufficiently small gaps, overlaps, and intersections having a tolerance which is small compared to the other features in the model. Generating Delaunay meshes for these types of models, and in general the repair process, has been a significant challenge in the past.

In addition, while most CAD applications require watertight output, our algorithm does not require that the input shape be a manifold since our meshing strategy handles PSCs. We preserve non-manifold features when they are larger than γ. Even a simple shape like the Dumbbell model would cause most traditional repair algorithms to fail because of the two boundary curves at the top and bottom of the cylinder.

However, some CAD models may still have problems which are not fixed by our approach. In the case of large gaps our algorithm either produces large protecting balls, which may result in hiding input features, or may leave some gaps only partially mended. As a consequence, the mesh we generate may fail to capture the input shape. Moreover, if two dimensional elements intersect deeply, our algorithm may fail to terminate because we do not explicitly compute intersections between surface patches. Extending the repair framework to handle these types of inconsistencies is an important avenue

of future work.

10. ACKNOWLEDGMENTS

This work is supported by the NSF grant CCF-0635008.

11. REFERENCES

[1] G. Barequet, C. A. Duncan, and S. Kumar. RSVP: A geometric toolkit for controlled repair of solid models. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):162–177, 1998.

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

[3] S. Bischoff and L. Kobbelt. Structure preserving CAD model repair. *Computer Graphics Forum*, 24(3):527–536, 2005.

[4] P. Borodin, M. Novotni, and R. Klein. Progressive gap closing for mesh repairing. In J. Vince and R. Earnshaw, editors, *Advances in Modelling, Animation and Rendering*, pages 201–213. Springer Verlag, July 2002.

[5] G. Butlin and C. Stops. CAD data repair. In *Proceedings of the 5th International Meshing Roundtable*, pages 7–12, 1996.

[6] CGAL. Computational Geometry Algorithms Library. http://www.cgal.org.

[7] S.-W. Cheng, T. K. Dey, and E. A. Ramos. Delaunay refinement for piecewise smooth complexes. In N. Bansal, K. Pruhs, and C. Stein, editors, *SODA*, pages 1096–1105. SIAM, 2007.

[8] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Symposium on Computational Geometry*, pages 274–280, 1993.

[9] 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(15):1109–1119, 2007.

[10] P. Degener and R. Klein. Texture atlas generation for inconsistent meshes and point sets. In *Shape Modeling International*, pages 156–168. IEEE Computer Society, 2007.

[11] T. K. Dey. *Curve and surface reconstruction: algorithms with mathematical analysis*. Cambridge University Press, New York, 2007.

[12] T. K. Dey and J. A. Levine. Delaunay meshing of piecewise smooth complexes without expensive predicates. Technical Report OSU-CISRC-7108-TR40, Department of CSE, The Ohio State University, July 2008.

[13] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, England, 2001.

[14] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):136–151, 2001.

[15] W. D. Henshaw. An algorithm for projecting points onto a patched CAD model. In *Proceedings of the 10th International Meshing Roundtable*, pages 353–362, 2001.

[16] J. Hu, Y. K. Lee, T. Blacker, and J. Zhu. Overlay grid based geometry cleanup. In *Proceedings of the 11th International Meshing Roundtable*, pages 313–324, 2002.

[17] T. Ju. Robust repair of polygonal models. *ACM Transactions on Graphics*, 23(3):888–895, 2004.

[18] Y. K. Lee, C. K. Lim, H. Ghazialam, H. Vardhan, and E. Eklund. Surface mesh generation for dirty geometries by shrink wrapping using cartesian grid approach. In *Proceedings of the 15th International Meshing Roundtable*, pages 393–410, 2006.

[19] A. A. Mezentsev and T. Woehler. Methods and algorithms of automated CAD repair for incremental surface meshing. In *Proceedings of the 8th International Meshing Roundtable*, pages 299–309, 1999.

[20] T. M. Murali and T. A. Funkhouser. Consistent solid and boundary representations from arbitrary polygonal data. In *ACM Symposium on Interactive 3D Graphics*, pages 155–162, 196, 1997.

[21] F. S. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[22] P. S. Patel, D. L. Marcum, and M. G. Remotigue. Automatic CAD model topology generation. *International Journal for Numerical Methods in Fluids*, 52(8):823–841, 2006.

[23] N. A. Petersson and K. K. Chand. Detecting translation errors in CAD surfaces and preparing geometries for mesh generation. In *Proceedings of the 10th International Meshing Roundtable*, pages 363–371, 2001.

[24] J. Qi and V. Shapiro. Epsilon-regular sets and intervals. In *International Conference on Shape Modeling and Applications (SMI 2005)*, pages 310–319, 2005.

[25] A. Sheffer, M. Bercovier, T. D. Blacker, and J. Clements. Virtual topology operators for meshing. *Int. J. Comput. Geometry Appl.*, 10(3):309–331, 2000.

[26] X. Sheng and I. R. Meier. Generating topological structures for surface models. *IEEE Computer Graphics and Applications*, 15(6):35–41, 1995.

[27] SISL. The SINTEF Spline Library. http://www.sintef.no/math_software.

[28] J. P. Steinbrenner, N. J. Wyman, and J. R. Chawner. Fast surface meshing on imperfect CAD models. In *IMR*, pages 33–41, 2000.

[29] Z. J. Wang and K. Srinivasan. An adaptive cartesian grid generation method for 'dirty' geometry. *International Journal for Numerical Methods in Fluids*, 39(8):703–717, 2002.

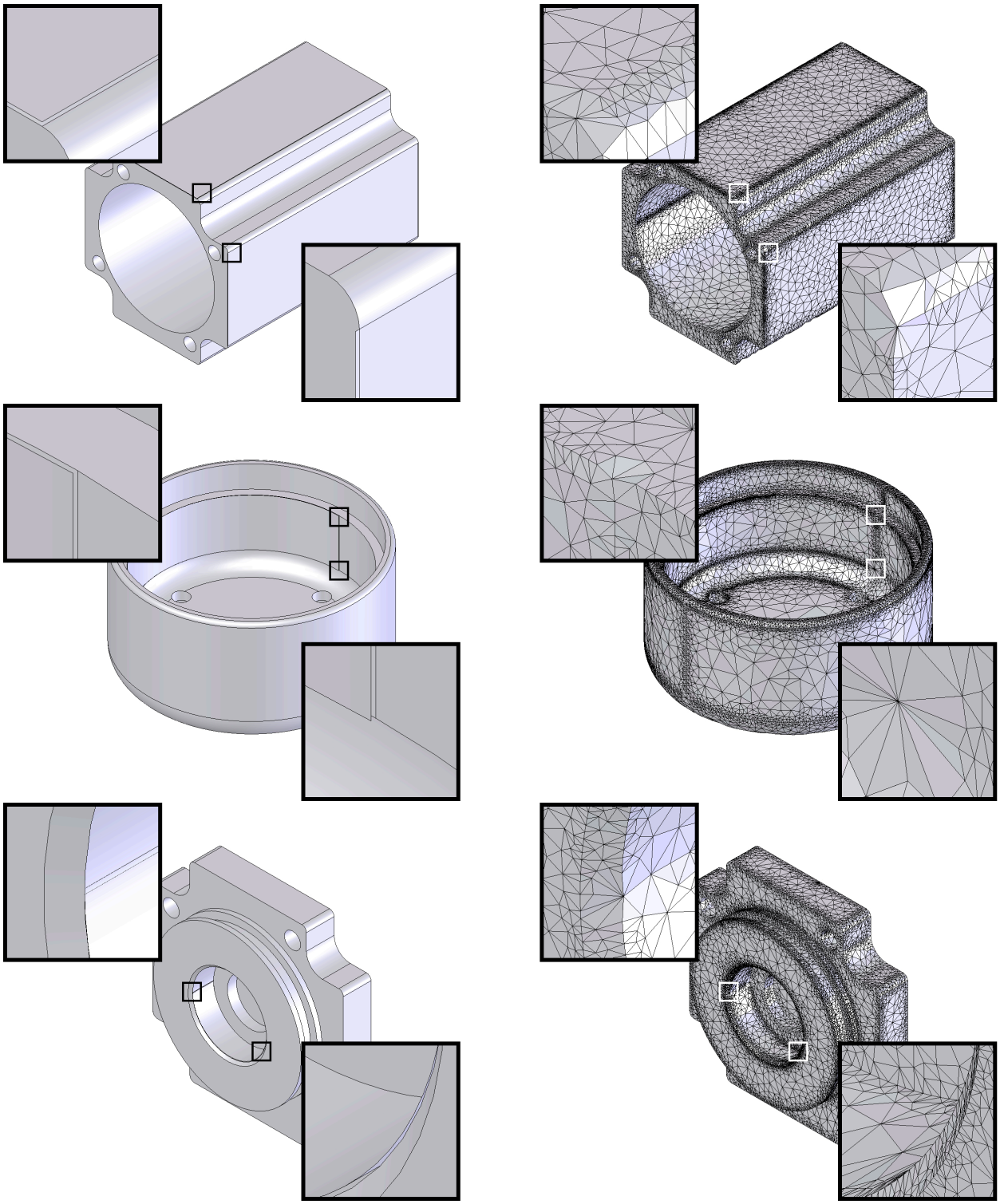

Figure 7: Original models (left) and repaired output meshes (right) for the Body, Cap, and Rear Cover models.

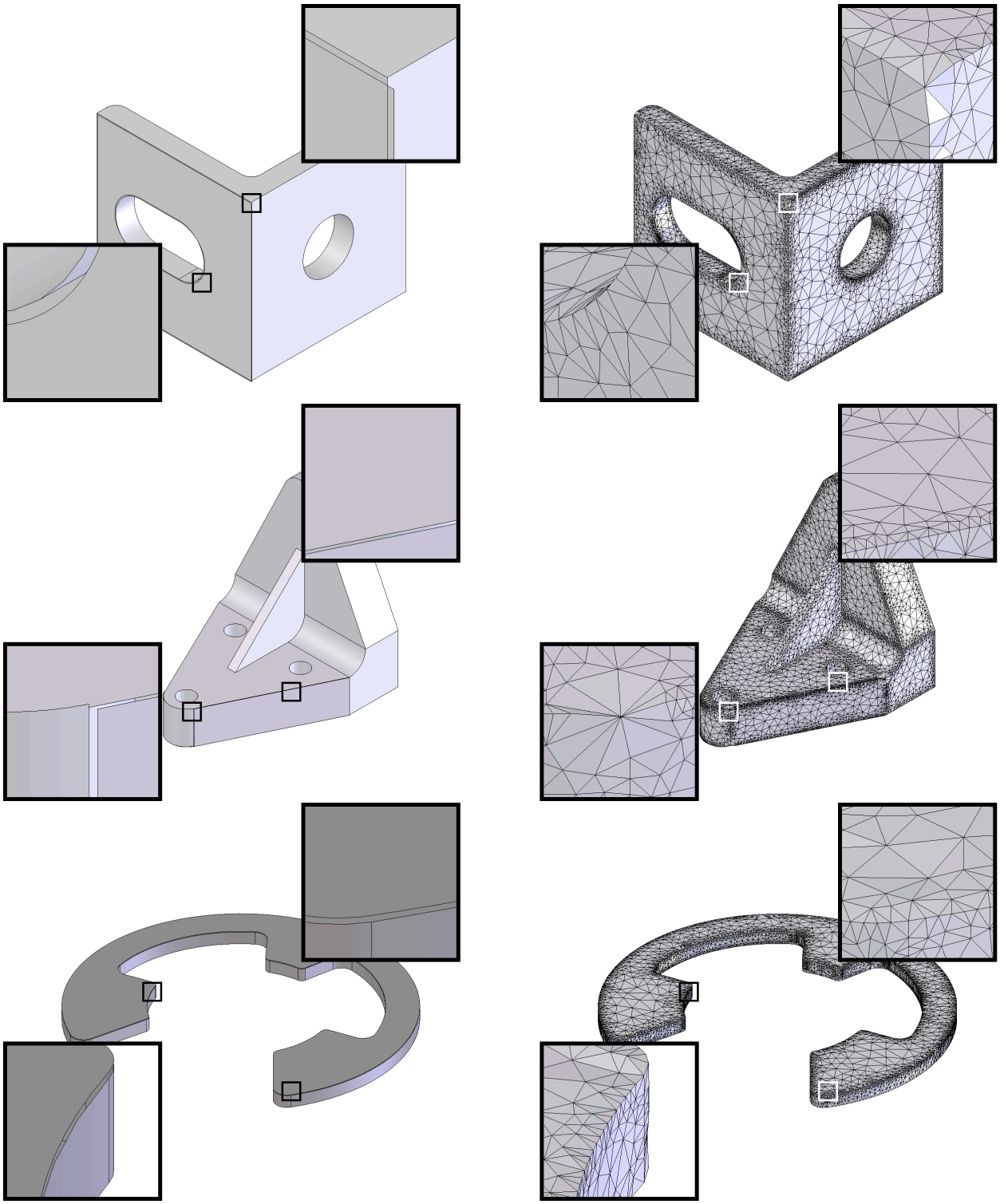

Figure 8: Original models (left) and repaired output meshes (right) for the Corner, Bracket, and Snape Ring models.