Progressive Gap Closing for Mesh Repairing

Pavel Borodin, Marcin Novotni, Reinhard Klein
Computer Graphics Group
Institute of Computer Science II
University of Bonn
Römerstr. 164, D-53117 Bonn, Germany

Abstract

Modern 3D acquisition and modeling tools generate high-quality, detailed geometric models. However, in order to cope with the associated complexity, several mesh decimation methods have been developed in the recent years. On the other hand, a common problem of geometric modeling tools is the generation of consistent three-dimensional meshes. Most of these programs output meshes containing degenerate faces, T-vertices, narrow gaps and cracks. Applying well-established decimation methods to such meshes results in severe artifacts due to lack of consistent connectivity information. The industrial relevance of this problem is emphasized by the fact that as an output of most of the commercial CAD/CAM and other modeling tools, the user usually gets consistent meshes only for separate polygonal patches as opposed to the whole mesh.

In this paper we propose a solution, which interprets the above issue as a mesh boundary decimation task. As suggested by Garland and Heckbert [4] and Popović and Hoppe [12], adding a vertex pair contraction operation enables to join unconnected regions of the mesh. In addition to this and the usual edge-collapse operation, we introduce a new vertex-edge collapse operation. This provides extra support for closing gaps and stitching together the boundaries of triangle patches lying in near proximity to each other. In our method, the decimation process is error controlled and conducted in a progressive manner in terms of the error. Therefore, the user is enabled to visually inspect and interactively influence the procedure.

Keywords: mesh repair, gap closing, CAD, simplification

1 Introduction

Today, polygonal surfaces have become ubiquitous as three-dimensional geometric representation of objects in computer graphics and even in some engineering applications. The fact that this particular representation is arguably the most widespread one is due to its simplicity, flexibility and rendering support by 3D

graphics hardware. Models of this type are used for rendering of objects in a broad range of disciplines like medical imaging, scientific visualization, CAD, movie industry, etc. Essentially, a set of polygons not containing consistent connectivity information suffices for rendering purposes. Since the generation of 3D models is application-driven, numerous models contain artifacts like T-vertices, degenerate triangles, gaps and holes.

However, as a natural consequence of recent advances in computer graphics field, we no longer want to be able only to render images of objects, but also to process and analyze the already available models. New demands and applications have arisen where *"better behaved"* polygonal models are desired, in a sense that they do not contain the above artifacts. Signal processing techniques on meshes aim at analyzing the geometry and improving the visual quality of models. Compression and progressive transmission facilitate robust transfer of 3D meshes through the Internet and their efficient storage. Mesh simplification algorithms reduce the complexity of highly detailed models by optimally approximating the geometry within a prescribed tolerance. An example of applying a common mesh decimation algorithm to a model containing gaps between the patches is demonstrated in Figure 1. Computing geodesic distances has gained a lot of attention in the last few years, since this numerical method was recently found very useful in a number of applications [11]. The effect of lack of consistent connectivity of vertices is depicted in Figure 2. There are numerous variants of these geometric modeling techniques, and some of them are applicable to meshes containing the artifacts described above. Generally however, in order to achieve optimal results, consistent vertex connectivity information is required.

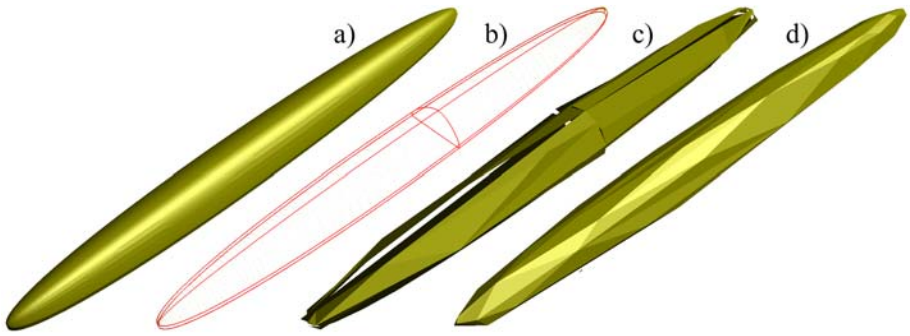

Figure 1: Result of applying mesh decimation to an unrepaired and repaired mesh. The original model is shown in a), it consists of 5500 faces. b) depicts the boundary, c) and d) demonstrate the result of the decimation applied to the unprocessed mesh and the repaired model, respectively. The simplified models contain 80 triangles.

Figure 2: Demonstration of the impact of inconsistent vertex connectivity on an algorithm computing geodesic distances on triangular meshes. The left side hippo's mesh is a closed watertight one, it can be seen that the algorithm produces appropriate wavefronts. In contrary, for the hippo on the right containing a long hole along its back, the wavefront breaks at the ends of the hole, thus producing erroneous results.

The aim of the work presented in this paper is to eliminate these artifacts and repair the input mesh by applying topological modifications while retaining the overall geometry.

In the proposed method we suggest to approach the problem of eliminating the artifacts listed above as a simplification task. Without loss of generality, we formulate our technique for triangle meshes. Since we intend to proceed by altering the topology of the triangle mesh, we clearly need topology modifying operators. Garland and Heckbert [4] and Popović and Hoppe [12] already generalized the common edge contraction operator, where two vertices lying on a common edge are contracted, to vertex contraction, where two vertices not necessarily connected by an edge are contracted. This way unconnected regions of the mesh can be joined. We further generalize this operation by introducing a *vertex-edge* contraction, where a vertex is unified with its projection on an edge. The intuition behind our algorithm is that the gap closing should proceed by attracting the boundaries of the mesh towards each other, which is achieved by utilizing the vertex-edge contraction operator. In other words, we apply the methodology of mesh decimation to the decimation of boundaries targeted at gap closing. Similarly to the common simplification methods, the procedure is error controlled as well; furthermore, it is performed in a progressive manner according to a monotonically increasing error. Our method essentially generates a sequence of meshes where the gaps and holes are progressively removed (see Figure 3). As explained in Section 2, the transition between the neighboring models in the sequence is accomplished very by applying a single contraction operator or its inverse. We enable the user to navigate between these meshes, thus facilitating the visual inspection of the results and interactive control of the process by determining a desired error tolerance. As pointed out by Weihe and Willhalm [15], stitching of mesh boundaries is a highly non-trivial process, since some of the gaps may be intentionally modeled, while others might be results of erroneous modeling or tessellation procedure. We take this issue into account by allowing the user to manually select/deselect areas to be considered during the stitching process.

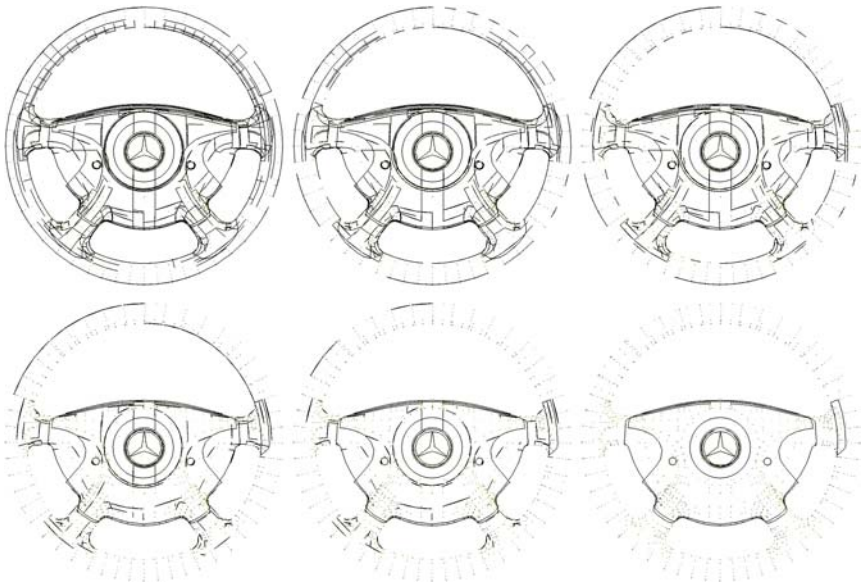

Figure 3: Representative stages of repairing a steering wheel model, demonstrating the progress of our algorithm. Only the boundary edges are rendered as wireframe. In addition the vertices of the mesh are rendered to give a feeling where the surface is. About 2700 models were generated in sequence between the leftmost original mesh and the rightmost final one. After the generation of this sequence, the user is free to navigate back and forth between the models and select the desired one; she/he can also choose to proceed further with the gap closing.

1.1 Related Work

Considerable amount of research and development has been conducted in the area of polygonal mesh and CAD data repairing in the recent years.

Mesh Reparation. Due to differences in inherent structures of meshes generated by various modeling tools and 3D acquisition techniques, the approaches handling the errors and degeneracies vary depending on the source of the data.

Turk and Levoy [14] generate polygonal models from registrated range data, they remove overlaps by clipping them, utilizing a technique called mesh zippering. The meshes coming from 3D scanners and volumetric data often contain artifacts of the reconstruction process: small handles and tunnels. Guskov and Wood [6] conceptualized these as *topological noise*, identified and eliminated them by cutting and sealing the mesh, thus reducing the genus and topological complexity of the model.

Due to the industrial relevance of the problem, a lot of work has been devoted to repairing polygonal models generated by modeling tools, mainly CAD systems. Our

algorithm differs from the available techniques as it employs a well established operation borrowed from mesh decimation field and as it is progressive. Barequet and Kumar [2] determine corresponding edges within an error tolerance and stitch them together in one pass. Butlin and Stops [3] present a method for repairing CAD data for analysis and exchange purposes. Guéziec et al. [5] generate manifold surfaces from non-manifold sets of polygons by identifying the topological singularities and decomposing the model into manifold components by cutting along these singularities. They also describe a stitching operation which allows to connect boundaries of the components, while guarantees the manifoldness. Murali and Funkhouser [9] first classify the regions of space as either solid or not, and generate a consistent set of polygons describing the boundary of the solids. Nooruddin and Turk [10] repair the polygonal models by converting them into volumetric representation; they subsequently eliminate the topological noise by morphological open and close operators, and finally reconstruct the polygonal mesh of the so-defined implicit function.

Simplification. Since the mesh simplification is one of the fundamental operations on polygonal meshes, there is an extensive amount of literature on this topic. However, we are interested only in methods allowing topology changes during the process. The *vertex clustering* family of methods has been introduced by Rossignac and Borrel [13] and has been refined in numerous more recent works (see e.g. [8]). The algorithms of this family essentially proceed by applying a 3D grid to the object and for each cell contracting all the vertices inside the cell. Although the degenerate faces are subsequently removed, it is difficult to influence the fidelity of the result due to lack of control over induced topological changes. The already mentioned *vertex contraction* operator [4, 12] offers more control over the topological modifications, however, without further processing it possibly generates non-manifold meshes.

1.2 Our Setting

The input of our algorithm is a general *polygon soup* – a set of unorganized polygons without any explicit topology information a la STL file format [1]. We assume the mesh to be composed of triangular faces. Notice that this does not imply any loss of generality, since the polygonal faces may easily be converted into triangular ones. The mesh to be processed by our method will possibly include the following artifacts:

- *degenerate faces* without finite area,
- unwanted *gaps* and *cracks* between regions of the mesh resulting from erroneous scan data reconstruction or modeling and/or tessellation of analytical surfaces,
- *holes* in the model due to missing polygons,
- *T-vertices* lying on interior of an edge of a face.

The vertex-edge decimation operator will possibly introduce non-manifold edges and/or vertices; therefore the output of our system will also be a non-manifold mesh in the general case. However, if a manifold surface is desired, the methods presented in [5] may be applied to our results, the mesh will be cut exactly along non-manifold features, thus preserving the consistent connectivity of the manifold components.

1.3 Paper Overview

The overview of the paper is as follows. In the next section we describe our new vertex-edge contraction operator, which is followed by the detailed description of the gap closing algorithm in the Section 3. We present and discuss the results in Section 4. Finally, we conclude and draft some directions for future research in Section 5.

2 Vertex-Edge Contraction

Let us describe our notation and terminology: $\mathcal{V}$ is a set of N abstract vertices, $|\mathcal{V}| = N$. The abstract polygonal surface $\mathcal{S}(\mathcal{V})$ contains the topological information of the model, it is composed of subsets of $\mathcal{V}$. Since we work with triangle meshes, the subsets are vertices, edges and triangles. In order to embed the mesh into the three-dimensional space $\mathbb{R}^3$, we assign a geometric position in space to each abstract vertex. Let $\mathcal{P} = \{ p_i \in \mathbb{R}^3 \mid 1 \le i \le N \}$. We now define the triangular mesh $\mathcal{M}$ as the pair $(\mathcal{S}, \mathcal{P})$.

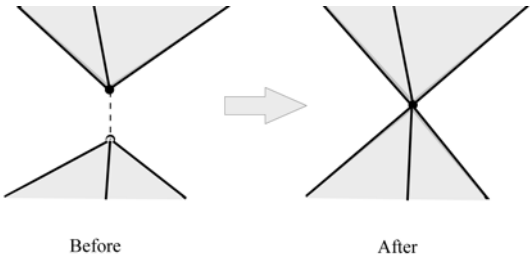

Figure 4: Vertex contraction operation

The decimation methods utilizing the *edge contraction* operator proceed by iteratively contracting edges. As already pointed out above, this operator does not provide enough topological flexibility during the decimation process. It is possible to close holes in the mesh by iterative application of the operator; however, the disconnected regions of the mesh will remain in separate components. The *vertex contraction* operator (cf. Figure 4) is a natural generalization of the edge contraction, and it seems to be the appropriate compromise between topological flexibility and control over the topological changes it induces. This operator contracts vertices not necessarily lying on a common edge, therefore it allows for stitching together the boundaries of the mesh. Note that we only want to process the boundaries of the

mesh, in which case even the vertex contraction may be not flexible enough. In order not to introduce distortions in case of narrow gaps, it is sometimes more favorable to project a vertex directly onto an edge. We call this operator a *vertex-edge contraction*, which is made up of the sequence of following operations (cf. Figure 5):

1. Project the vertex v orthogonally onto the edge e.
2. Insert a vertex v' into the edge at the geometric position of the projection.
3. Split the triangle t_1 into two triangles $t'_1 = (v_0 , v' , v_2)$ and $t_2 = (v_1 , v_2 , v')$.
4. Perform a vertex contraction to v and v'. The new position of the vertex will be a convex combination of v and v', we move the new vertex v_{new} into position $\lambda v + (1 - \lambda) v'$.

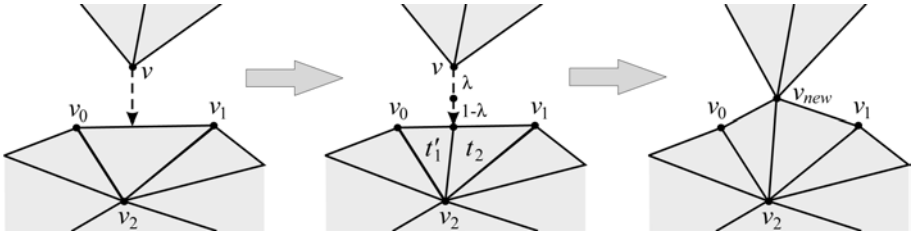

Figure 5: Vertex-edge contraction operation

At this point, it is necessary to mention that during the process we maintain for each boundary edge a list of vertices being projected onto it. In case the geometric position of the edge changes, the edge is destroyed or no longer belongs to the boundary due to modifications in its vicinity, we recompute the correspondences for all vertices in the list. Furthermore, note that if the vertex v is projected onto a vertex incident to the edge e, only a simple vertex contraction is performed. Thus, the vertex-edge contraction is a further generalization of the vertex contraction. The projection of vertices is conducted according to an error measure; see the next section for details on that.

The goal of our method is to construct a series of meshes $\mathcal{M}_0 , \mathcal{M}_1 , \dots , \mathcal{M}_M$ by incrementally applying the vertex-edge contraction operator, where $\mathcal{M}_0$ is the input mesh. The user should be able to choose the desired model $\mathcal{M}_i$, therefore we allow him/her to navigate between the meshes. Thus, we essentially generate a sequence of meshes in a sense of multiresolution representations (see e.g. [7]). The forward navigation is clearly accomplished by applying the contraction according to some well-defined order. However, in order to undo the operations during the backward navigation, we have to define an *inverse* vertex-edge contraction operator, and store some data during the generation of the sequence. The data to be stored and the inverse operators for vertex contraction are described in [4, 12]. We focus only on

formulating an inverse operator of the vertex-edge contraction (see Figure 5):

1. Project v_{new} orthogonally onto the edge e determined by vertices v_0 and v_1, this way we reconstruct the position of the vertex v'.

2. Compute v as follows: $v = \frac{v_{new} - \lambda v'}{1 - \lambda}$.

3. Delete the triangle t_2, which can unambiguously be determined, since we define it to be incident to v_1 and non-incident to v_0.

4. Restore the triangle $t_1 = (v_0 , v_1 , v_2)$. Note that v_2 can also always unambiguously be determined, since e is a boundary edge.

Note that we do not have to delete any vertex, since we reuse v to store v_{new}. Given a mesh $\mathcal{M}_i$, $0 < i \leq M$, in order to fully restore the mesh $\mathcal{M}_{i-1}$, we only need to store the indices of vertices v_0 and v_1 as "split information". The projection direction is the vector pointing from v_{new} towards v', in order not to be forced to recompute the projection, we additionally save the λ. We store the split information for each projected boundary vertex; to retain the ordering, we include the indices of vertices of this kind in an array as the decimation proceeds. Moreover, note that in order to navigate between the already generated meshes, it is not necessary to maintain for every boundary edge lists of vertices corresponding to it. The ordering of features to be contracted, which we store in an array during the procedure, fully determines the process for these cases. We only have to store this information for the mesh $\mathcal{M}_M$, where M is the largest index in the mesh sequence, since the correspondences are needed only if the user chooses to continue to generate further meshes.

3 Gap Closing

The algorithm for gap closing essentially consists of a preprocessing phase and the boundary decimation process itself. The method proceeds according to an increasing error computed as distance between feature pairs that are possible candidates for a contraction operation. This in turn implies that our approach has the nice *progressive* property, which means that always the contraction corresponding to the smallest error is performed. The progressivity is not only a numerically pleasant feature, it is also greatly appealing on the user level, since the user can follow the process in an intuitive manner.

3.1 Preprocessing Phase

1. *Reading the mesh*: we first read the input triangle soup and convert it to an *indexed face set* representation where every vertex is stored only once, and the triangles are determined by the indices of according vertices. We accomplish this by lexicographically sorting the vertices in a priority queue.

Let v_{stored} be a stored vertex and v_{input} the actual input vertex, in case the distance $d(v_{stored}, v_{input}) = \|v_{stored} - v_{input}\| < \varepsilon$ for some ε, the vertices are considered to be the same and v_{input} is not added to the priority queue.

2. *Identification of boundaries*: find all the edges e and vertices v that are elements of the boundary B. The boundary edges are those having only one incident triangle, the boundary vertices are simply the vertices incident to boundary edges.

3. *Identification of corresponding vertex-vertex and vertex-edge pairs*: in order to accomplish a pairing between vertices and edges to be contracted, for all boundary vertices we find the nearest boundary edge that is non-incident to the vertex. If an orthogonal projection of the examined vertex onto the corresponding nearest edge is possible, we store the edge as the paired feature; otherwise we store the nearest vertex of the edge. We additionally store all corresponding vertices for each boundary edge.

4. *Ordering of the pairs*: For each feature pair we compute the distance between the features as an error measure. We subsequently include all the pairs into a priority queue sorted by this error.

An example of the preprocessing phase is depicted in Figure 6.

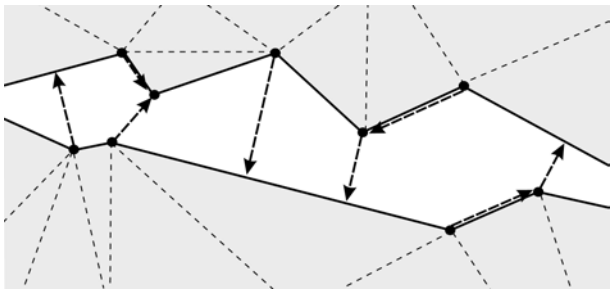

Figure 6: Result of the preprocessing step, the dashed arrows indicate the correspondences. Note that some arrows point along an edge, which possibly implies an edge contraction.

3.2 Decimation step

For a boundary decimation step we first pop the vertex with minimal error from the queue, then we perform a vertex or vertex-edge contraction depending on the type of the corresponding nearest feature. Finally we maintain the correspondences for each vertex corresponding to a modified edge; we accordingly maintain the priority queue as well. The pseudo code for the decimation step can be found in Algorithm 1.

Algorithm 1: Decimation step

```
1:  Vertex v = pqueue.pop_min();
2:  Feature f = v.nearest_feature();
3:  if is_vertex(f) then
4:     vertex_contraction(v, f);
5:  else
6:     vertex_edge_contraction(v, f);
7:  end if
8:  EdgeSet E = {modified_edges()};
9:  for all Edge e ∈ E do
10:    for all Vertex v ∈ {e.corresponding_features()} do
11:       v.maintain_correspondences();
12:       pqueue.reinsert(v);
13:    end for
14: end for
```

Note that if a vertex v is projected onto an edge in the vicinity of one of its vertices v_i, it is reasonable to *snap* the projected vertex v' to the vertex v_i, this way creating triangles with very small edges will be avoided. In our implementation if $d(v', v_i) < \varepsilon$, where ε is the global error threshold, we snap the vertices v_i and v'. Thus, in this case a vertex contraction will be applied to v and v_i.

4 Results

The first example shown in this section demonstrates a steering wheel model kindly provided by DaimlerChrysler AG. As shown in Figure 8, this model gives an insight into artifacts resulting from tessellating a complex model of a tessellated trimmed NURBS surface. Every kind of artifact listed in the introduction can be a found in the model. As it can be seen in the close-up, there are holes even inside relatively flat triangular regions and narrow gaps between the original NURBS patches as well. We managed to close the undesired gaps and holes. Note however, that the gaps intentionally modeled by the CAD tool operator are preserved. The impact of the mesh repairing on performance of mesh processing methods is demonstrated by an example of triangle mesh simplification. The simplified original model is hardly recognizable even after a relatively small amount of decimation steps. In contrary, the repaired model retains the overall shape after a considerable reduction of number of triangles (the original model contains 6540 triangles, it has been simplified to consist of 1229 triangles).

An example of a different nature coming from a 3D acquisition tool is the 3D scan of a woman (see Figure 7). The model consists basically of one connected component, however, it contains holes with jagged boundary and T-vertices. As shown in Figure 7, our method succeeds to remove the artifacts from this mesh as well.

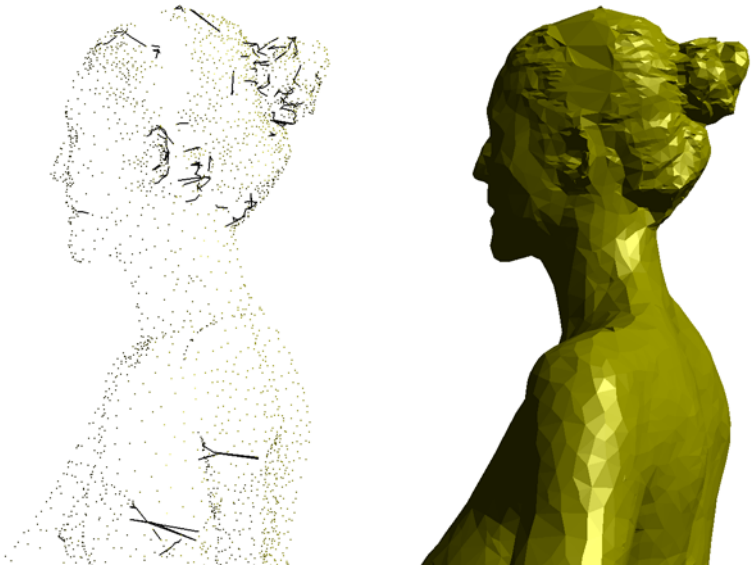

Figure 7: The results for a scan. The boundaries identified in the original model are depicted on the left side. Our procedure removed all the holes, the result can be seen on the right side.

5 Conclusions and Future Work

We presented an algorithm for repairing meshes containing various artifacts due to incorrect modeling or acquisition process. Our method removes the inconsistency of vertex connectivity and thus produces high fidelity models with properties, which are important prerequisites for most of the geometry processing and numerical simulation methods.

Essentially, our technique accomplishes the removal of undesired artifacts by simplifying the boundary of the mesh. For this purpose we generalized the vertex contraction operator into the vertex-edge contraction. The needed topological modifications were already possible by applying the vertex contraction, however, our vertex-edge contraction operator provides additional flexibility. Our system is capable of creating a sequence of meshes generated during the boundary decimation process, which allows the user to choose the desired model. With additional interactive functionality, the user is enabled to select the regions of the mesh she/he wants to repair.

As for further development, we see plenty of room to develop interactive tools facilitating the effective work with the system. A 3D brush for instance could be used to navigate along a narrow gap and selecting it for subsequent *zippering*.

References

[1] Stereolitography interface specification. SLA CAD, 3D Systems Inc. (Valencia, CA), 1989, p/n 500650S01-00.

[2] Gill Barequet and Subodh Kumar. Repairing CAD models. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 363–370, 1997.

[3] Geoffrey Butlin and Clive Stops. CAD data repair. In *5th International Meshing Roundtable*, pages 7–12, 1996.

[4] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *ACM SIGGRAPH Computer Graphics Proceedings*, pages 209–216, 1997.

[5] Andre Guéziec, Gabriel Taubin, Francis Lazarus, and Bill Horn. Cutting and stitching: Converting sets of polygons to manifold surfaces. *Trans. on Visualization and Computer Graphics*, 7(2), pages 136–151, 2001.

[6] I. Guskov and Z. Wood. Topological noise removal. In *Graphics Interface*, 2001.

[7] Hugues Hoppe. Progressive meshes. *Computer Graphics*, 30 (Annual Conference Series), pages 99–108, 1996.

[8] Kok-Lim Low and Tiow Seng Tan. Model simplification using vertex-clustering. In *Symposium on Interactive 3D Graphics*, pages 75–82, 188, 1997.

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

[10] F. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. Technical Report GITGVU -99-37, Georgia Institute of Technology, 1999.

[11] Marcin Novotni and Reinhard Klein. Computing geodesic distances on triangular meshes. Submitted for publication to Winter School on Computer Graphics 2002.

[12] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *ACM SIG-GRAPH Computer Graphics Proceedings*, pages 217–224, 1997.

[13] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*. Springer-Verlag, 1993.

[14] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. *Computer Graphics*, 28 (Annual Conference Series), pages 311–318, 1994.

[15] Karsten Weihe and Thomas Willhalm. Why CAD data repair requires discrete algorithmic techniques. In *2nd Workshop on Algorithm Engineering*, 1998.

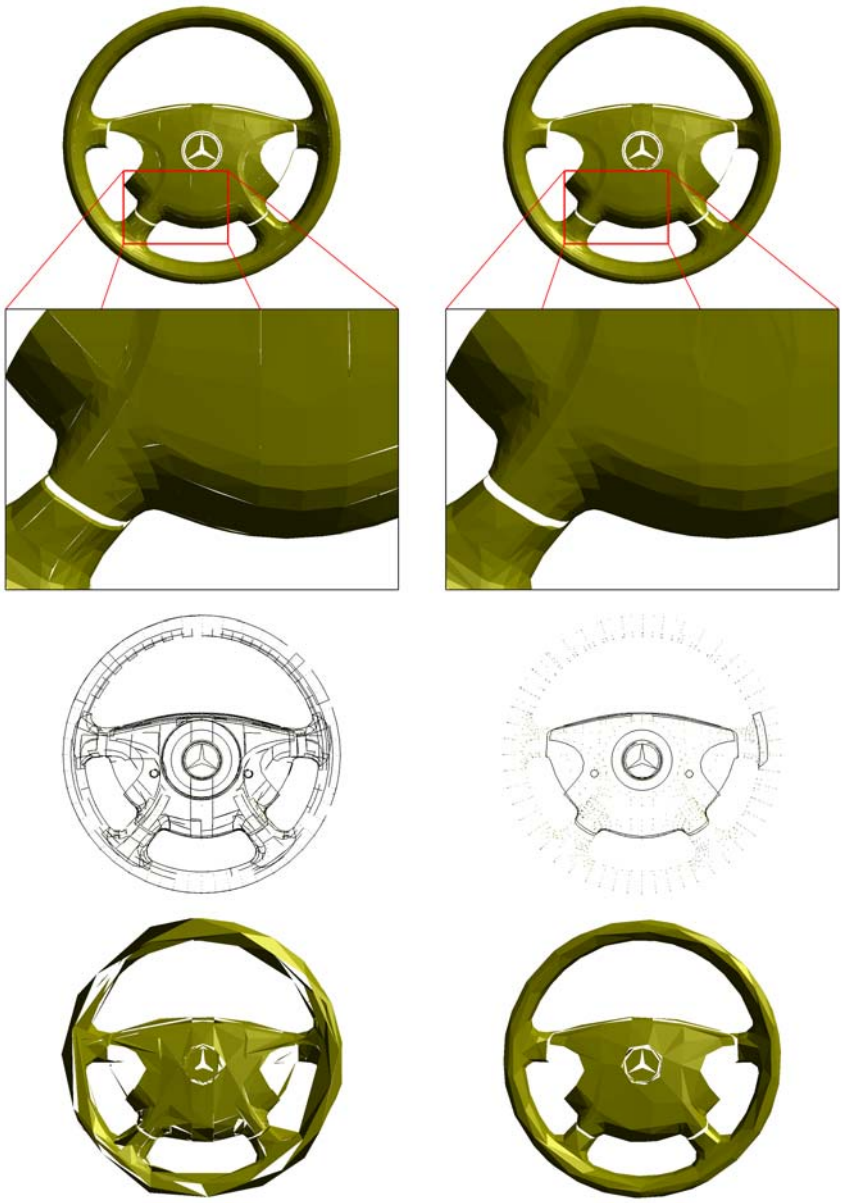

Figure 8: Results for a steering wheel. The images for original model are shown on the left and for repaired model on the right. The top image depicts the solid model, the one below shows a close-up on the highlighted region. Note the holes on the left side and that they have been removed by our procedure, which is even more obvious by looking at the boundaries. The bottom images demonstrate the impact of the decimation of models.