High-Quality Simplification with Generalized Pair Contractions

Pavel Borodin
University of Bonn
borodin@cs.uni-bonn.de

Stefan Gumhold
University of Tübingen
gumhold@uni-tuebingen.de

Michael Guthe
University of Bonn
guthe@cs.uni-bonn.de

Reinhard Klein
University of Bonn
rk@cs.uni-bonn.de

Abstract

Because of its simplicity, the vertex pair contraction operation became very popular in mesh simplification. It allows for topological changes during simplification. In this paper we propose the use of the additional pair contractions of a vertex onto an edge or onto a triangle and the connection of two edges. The generalized contraction operations have several advantages over simple vertex pair contraction. They allow to repair cracks and self-intersections and to sew unconnected components with less error. Due to this property a simple high-quality out-of-core simplification is possible.

Keywords: mesh simplification, mesh repair, out-of-core simplification

1 Introduction

Because of their simplicity, flexibility and wide support by graphics accelerators, polygonal meshes have become the most commonly used surface representation in computer graphics. They are used for rendering of objects in a broad range of disciplines like medical imaging, scientific visualization, CAD, movie industry, etc. New acquisition techniques allow for the generation of highly detailed objects with permanent increasing polygon count. The handling of huge scenes composed of these high-resolution models rapidly approaches the computational capabilities of any graphics accelerator. Level of detail techniques become inevitable. In order to build such a level of detail representation a large collection of simplification algorithms exist, that produce high quality approximations of complex models with a reasonable amount of polygons.

Since the generation of 3D models is application-driven and mostly automatic, numerous models do not have consistent connectivity information. Typical artifacts in these models are T-vertices, degenerate triangles, self-intersections, gaps, small holes or very close but topologically not connected surface parts. During simplification the artifacts can lead to unnecessarily large errors or even to further self-intersections in the simplified model. The pair contraction operation introduced independently by Popovic and Hoppe [18] and Garland and Heckbert [6] allows to contract any two vertices independent of whether they are topologically adjacent or just geometrically close. The vertex pair contraction facilitates topological modifications but is not general enough tot connect close or even intersecting surfaces with small error early in the simplification.

In [2] Borodin et al. generalized the vertex pair contraction operation by contraction of a vertex onto an edge. This improves the sewing potential of the pair contraction simplification algorithm. In this paper we completely generalize the pair contraction approach and allow contraction of a vertex with another vertex, onto an edge or triangle and connection of two edges. The new operations allow to connect close and intersecting surface parts that are not topologically incident on the early stages of simplification.

The paper is structured as follows. First we discuss related work. Then we introduce the generalized pair contraction operations. Section 4 details our use of a spatial grid, which is used to find all potential contraction pairs. Next we detail the simplification algorithm. Selected applications of our method are listed in section 6. Finally we present results that demonstrate the advantages of the generalized pair contraction and conclude with some avenues for future work.

2 Related Work

Mesh Repair. Due to the differences in the inherent structure 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 [20] generate polygonal models from registered 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 [10] 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 the field of mesh decimation and can adapt to the resolution. Barequet and Kumar [1] 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. [8] 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 allowing to join the boundaries of the components while guaranteeing the manifoldness. Murali and Funkhouser [16] first classify the regions of space as either solid or not, and generate a consistent set of polygons describing the boundary of solids. Nooruddin and Turk [17] repair the polygonal models by converting them into a volumetric representation, they subsequently eliminate the topological noise by morphologic open and close operators, and finally reconstruct the polygonal mesh of the so-defined implicit function. More recently, Borodin et al. [2] introduced the vertex-edge contraction to remove T-vertices, degenerate triangles, gaps and holes progressively.

Simplification. Since 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 family of *vertex clustering* methods has been introduced by Rossignac and Borrel [19] and has been refined in numerous more recent works, see e.g. [15]. 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 the induced topological changes. The already mentioned *vertex contraction* operator [6, 18] offers more control over the topological modifications, however, without further processing it possibly generates non-manifold meshes.

Out-Of-Core Simplification. To simplify models of ever increasing size a number of out-of-core simplification algorithms have been developed. El-Sana and Chiang [5] sort all edges according to their lengths and use this ordering as decimation sequence. Lindstrom [13] uses vertex clustering to reduce the number of vertices. As the representing position of each vertex cluster is computed from an accumulated quadric error metric, the memory requirement of the algorithm is proportional to the size of the output model. For cases where neither input nor output model fit into main memory an out-of-core vertex clustering [14] was developed. The multiphase algorithm [7] uses vertex clustering to reduce the complexity of the input model followed by a greedy simplification approach and achieves high quality results.

An other way for out-of-core simplification is to split the model into smaller blocks, simplify these blocks and stitch them together for further simplification. In [11] this approach is applied to terrain and in [4] to arbitrary meshes. The approach has the problem that special care has to be taken at patch boundaries. However as we will show in this paper the general pair contractions solve them very elegantly.

Recently, stream decimation algorithms [21, 12] for out-of-core simplification have been developed, but the resulting model is not optimal with respect to mesh size and Hausdorff distance of the simplified model to the original.

3 Generalized Pair Contractions

As already mentioned above, the vertex pair contraction operation not always sews together geometrically close but not incident surface parts. In some cases a vertex on one part of a mesh is close to another part, but too far from any vertex on that part. In this case any vertex contraction between the two parts would introduce distortions and often also a large geometrical error. Sometimes it is more favorable to contract a vertex directly with an edge or triangle, what allows to connect the nearest parts of a mesh and to close the most narrow gaps first.

But in some cases even these three operators will be not sufficient. Two unconnected regions of a mesh, which are very close to each other, not necessary have any vertices close enough to the other part in order to connect the parts without causing distortions. To enable sewing in such situations we also introduce a contraction operation which connects two close edges. In summary, we extend the vertex pair contraction operator to the generalized pair contraction operator by introducing the new types of contraction: vertex-edge, vertex-triangle and edge-edge.

In the vertex-edge and vertex-triangle contraction operations an *intermediate vertex* v' is created on the edge or triangle of the contraction pair just in order to contract it with the vertex v of the contraction pair. The latter is called *contraction vertex*. See Figures 2 and 3 for an illustration. In case of the edge-edge contraction operation we create two *intermediate vertices* as shown in Figure 4. Before describing the new contraction operations in more detail we explain the use of the quadric error metrics with generalized pair contractions and describe improvements necessary for the preservation of sharp features.

3.1 Order of Operations

For the ordering of operations and to find the optimal position of a new vertex after the contraction operation, our simplification algorithm uses the technique of quadric error metrics as presented by Garland and Heckbert [6]. For each face f of the original mesh a *fundamental error quadric* $Q_f(p)$ is defined as the symmetric homogeneous 4×4-matrix, which measures the squared distance d^2 of a point $p \in \mathbb{R}^3$ to the plane of f as $d^2 = (p,1)Q_f(p,1)^t$. Each vertex v in the original mesh is assigned an initial quadric constructed as the matrix sum of the fundamental quadrics of its incident faces, divided by the order of v and optionally weighted by their areas.

When contracting two vertices v_1 and v_2, the vertex quadrics are added yielding the quadric $Q = Q_1 + Q_2$. This quadric computes the sum over the two surface patches incident to v_1 and v_2 of the squared distances, divided by the order of the appropriate vertex and optionally area weighted. The location of the new vertex v_{new} is set in a way to minimize the quadric error $e_q = v^T Qv$.

In our algorithm we do not accumulate quadric errors, instead the error quadrics are always calculated on base of the current mesh. New quadrics are calculated at three locations in the algorithm:

- As mentioned above, in the preprocessing phase for each vertex we calculate the initial error quadric.
- When an intermediate vertex is created on an edge or a triangle, its quadric is calculated the same way: as a matrix sum of fundamental error quadrics of all faces incident to the newly created intermediate vertex, divided by the order of the vertex and optionally weighted by their areas.
- After performing the operation we recalculate the quadrics of the newly created vertex and all adjacent vertices from the faces currently incident to the vertices.

To ensure that due to certain special cases, the Hausdorff error introduced by a contraction operation is not greater than a predefined error threshold d_{max}, before performing the operation we calculate the one-sided Hausdorff distance d_v between all simplices of the original mesh, whose nearest points on the simplified mesh lie inside the neighbourhood $N_{s_1} \bigcup N_{s_2}$ of the contraction simplices and the neighbourhood N_v of the newly created vertex v. Here N_v consists of v and its incident faces and edges. If s_n is a vertex, N_{s_n} consists of the vertex itself and its incident faces and edges; if s_n is an edge N_{s_n} consists of the edge and its incident faces; finally if s_n is a face N_{s_n} consists only of the face itself. If d_v exceeds d_{max}, we reject the operation.

Since the error quadrics are not accumulated, we use an approximation $\widetilde{d_s}$ of the Hausdorff distance to organize the possible contraction operations in a priority queue:

$$\widetilde{d_s} = max(d_{s_1}, d_{s_2}) + \sqrt{1/2 * e_q}, \qquad (1)$$

where d_{s_1} and d_{s_2} are the accumulated errors of the simplices s_1 and s_2. The accumulated error of the simplex s is the one-sided Hausdorff distance from the neighbourhood N_s of the simplex s to the original mesh.

The first part of this sum ($max(d_{s_1}, d_{s_2})$) is the Hausdorff distance of the neighbourhood $N_{s_1} \bigcup N_{s_2}$ of the contraction simplices before the operation. The quadric error e_q represents the sum of the squared distances from the newly created vertex to the incident faces of two contraction vertices. Therefore, the second part of the sum in equation 1 ($\sqrt{1/2 * e_q}$) is an approximation of the distance between the meshes before and after the operation. Thus, the value $\widetilde{d_s}$ is a good estimation of Hausdorff distance between simplified and original meshes. This makes this error metric compatible with euclidian distances used to find the nearest simplices, as described in section 4.

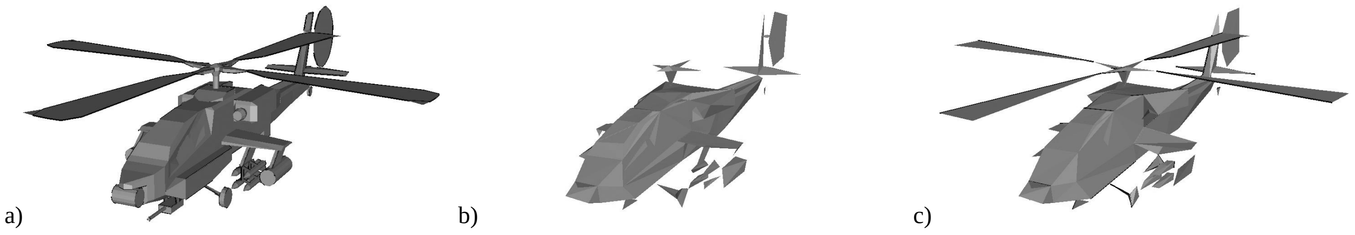


Figure 1: Simplification with and without feature preservation: a) original model (1972 vertices); b) reduced to 200 vertices without feature preservation; c) reduced to 200 vertices with features preserved.

Handling of Boundaries and Features. In order to preserve boundaries we proceed as proposed by Garland and Heckbert. For each face incident to a boundary edge we generate a perpendicular plane running through the edge. Then we compute fundamental quadric for this constraint plane and add it to the quadric sums of both vertices incident to the boundary edge, divided by the order of appropriate vertex and optionally weighted by the squared length of the edge.

A further enhancement to the error quadrics is necessary to preserve sharp features. Figure 1b shows the simplified helicopter of Figure 1a using the introduced error quadrics as discussed thus far. Sharp features as the propeller are destroyed. To handle very sharp edges properly we process them in the same way as boundaries.

As *feature edges* and *feature vertices* we define all edges and vertices, whose incident faces lie inside chosen small angle α_{max}. Note, that according to this definition boundary edges are also feature edges. For each detected feature edge e or feature vertex v we find the average plane of all its incident faces and compute its fundamental quadric. This fundamental quadric is then added to the quadric sums of both vertices incident to the feature edge or to the quadric sum of the feature vertex. Figure 1c shows the helicopter simplified with feature preservation to the same number of vertices as in 1b.

3.2 Vertex-Edge Contraction

The exterior case of this operation, when both the vertex and the edge lie on boundaries, was described in [2]. Here we allow pairs of an arbitrary vertex and an arbitrary edge. We proceed as follows (see Figure 2):

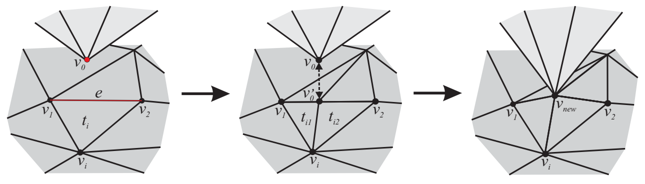


Figure 2: Vertex-edge contraction operation performs no reduction, but increases the connectedness of the model.

- Project the contraction vertex v_0 onto the edge $e = (v_1, v_2)$.
- Insert the intermediate vertex v'_0 on the edge at the geometric position of the projection and interpolate its quadric (see section 3.1).
- Split each triangle t_i, incident to the edge e, into two triangles $t_{i1} = (v_1, v'_0, v_i)$ and $t_{i2} = (v_2, v'_0, v_i)$.
- Perform a vertex contraction of v_0 and v'_0.

This operator perfectly allows to sew borders and close parts of the mesh. It doesn't decrease the number of vertices but increases the connectedness of the model. In the case, when both the vertex and the edge are incident to the same triangle, the vertex-edge contraction is topologically equivalent to an edge flip.

In the manifold case a second flip of the resulting edge would re-produce the original configuration. This can easily lead to an infinite number of successive edge flip operations. To avoid this problem we allow the edge flip, only if an additional criterion is fulfilled: the minimum angle among all affected triangles has to increase after the operation by a non-zero constant. As the minimum angle cannot increase by a constant infinitely, we avoid infinite sequences of edge flips. At the same time, triangles with acute angles can be removed, what improves the overall triangle shape.

3.3 Vertex-Triangle Contraction

Here we generalize the vertex pair contraction further by connecting an arbitrary vertex with an arbitrary non-incident face. While the vertex-edge contraction operation is chosen by the algorithm mostly on boundaries, this one connects geometrically close surface parts. The vertex-triangle contraction can be split in four steps (see Figure 3):

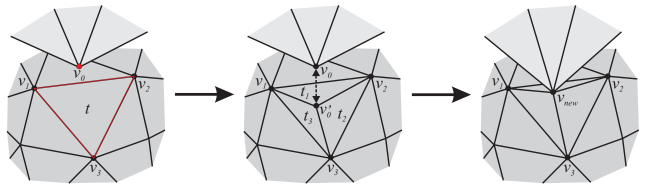


Figure 3: The vertex-triangle contraction operation performs no reduction, but increases the connectedness of the model.

- Project the contraction vertex v_0 onto the triangle $t = (v_1, v_2, v_3)$.
- Insert the intermediate vertex v'_0 on t at the projection point and interpolate its quadric (see subsection 3.1).
- Split the triangle t into three triangles $t_1 = (v_1, v_2, v'_0)$, $t_2 = (v_2, v_3, v'_0)$ and $t_3 = (v_3, v_1, v'_0)$.
- Contract the vertices v_0 and v'_0.

The vertex-triangle contraction operator neither performs a reduction, but does increase the connectedness of the mesh.

3.4 Edge-Edge Contraction

The third generalized pair contraction operation is useful only in cases, when two surface parts are close or intersecting, but the distance between any vertex from one part to another part is significantly larger than the distance between two edges.

We proceed as in the vertex-edge contraction, but insert intermediate vertices on both edges (see Figure 4):

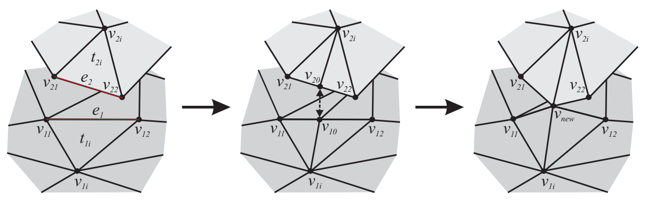


Figure 4: The edge-edge contraction operation increases the connectedness of the model by cost of the insertion of one vertex.

- Find the shortest distance between edges $e_1 = (v_{11}, v_{12})$ and $e_2 = (v_{21}, v_{22})$.
- Insert the intermediate vertices v_{10} and v_{20} at the projection points and interpolate their quadrics (see section 3.1).
- Split each triangle t_{1i} incident to the edge e_1 into two triangles.
- Split each triangle t_{2i} incident to the edge e_2 into two triangles.
- Contract the vertices v_{10} and v_{20}.

We allow edge-edge contraction, only if the projection points lie inside the edges.

4 Spatial Search Data Structure

In order to avoid a quadratic algorithm to pair close simplices for the contraction operations a spatial search data structure supporting nearest simplex queries is necessary. The data structure must be able to handle vertices, edges and triangles. Furthermore it must be dynamic as simplices are eliminated and sheared during the simplification process. We chose to use a regular grid for the spatial search because it performs well in static and dynamic environments [22] and is easy to implement. In each grid cell we stored a list of the simplices partially or completely contained in the cell.

In the beginning of the simplification process we used a grid with uniform edge length of twice the average edge length of the model, such that each simplex was in average contained in one or two cells allowing for fast insertion and removal. During the simplification process we kept track of the increasing average edge length and every time it exceeded the grid edge length, we destroyed the grid, created a new one of double grid edge length and inserted all remaining simplices to the new grid. In the case of models with low variance in the simplex size we achieved with this simple strategy, that in average each simplex had to be entered in a constant number of grid cells only.

4.1 Simplex Insertion and Removal

To insert a vertex we simply compute the enclosing cell and add the vertex to its list of contained simplices. Edges and triangles can penetrate more than one cell into which they had to be entered. In the case of an edge a simple incremental algorithm could be used to trace the penetrated cells along the edge. For triangles we implemented a not optimal but simple solution. First we collected all cells intersected by the bounding box of the triangle. Then we sorted out the actually penetrated cells by a marching cubes like strategy. Each of the grid vertices from intersected cells was classified to be *above*, *below* or *outside* of the triangle in consideration. Finally, the triangle was inserted in all grid cells with at least one vertex classified above and one classified below. As each triangle was in average inserted into a small number of cells the presented strategy worked reasonably fast.

For fast removal we stored for each vertex a pointer to the list item in the enclosing cell such that it could be removed in constant time. Similarly, we kept for each edge and each triangle a list of pointers to list items such that any simplex could be removed in time proportional to the number of penetrated cells.

Simplices incident to contraction vertices move in space during the contraction operation. It turned out that insertion and removal was so fast that it did not pay off to implement an optimized move operation for the regular grid. Instead we simply removed the simplex before the contraction operation and inserted it again afterwards.

4.2 Distance Sorted Nearest Neighbour Queries

For our simplex pairing strategy as described in the next section it is necessary to find for each vertex besides the adjacent vertices, the closest non-incident simplex, and for each edge the closest non-incident edge. We designed a general algorithmic scheme to find the closest simplex of a vertex or an edge, i.e. the *seed simplex*. The scheme exploits two priority queues, the *cell queue* to store the next to be considered grid cells sorted by increasing distance to the seed and the *simplex queue* to store the non-incident simplices from the considered cells also in increasing distance from the seed. The crucial idea is that the head of the simplex queue is the closest simplex to the seed, only if the head of the cell queue, i.e. the closest not yet considered cell, is further apart from the seed. Now we can state the closest simplex search algorithm

- given a seed vertex or edge, initialize the simplex and cell queues to be empty
- lookup the *seed cells* penetrated by the seed, add all contained non-incident simplices to simplex queue and the cells adjacent to the seed cells to the cell queue
- while the simplex queue is empty or the distance of the closest not considered cell is smaller than the closest simplex
  - extract the head of the cell queue, add all contained non-incident simplices to the simplex queue and add the not considered adjacent cells to the cell queue
- return the closest simplex from simplex queue

For the closest simplex algorithm the following distance computations needed to be implemented: distance from vertex to vertex, edge, triangle or cell and the distance from edge to edge or cell. To avoid the square root operation we sorted the cells and simplices by the squared distance. As we were looking for the closest simplex, the distance computations from a vertex to an edge or triangle could be discarded if the orthogonal projection of the vertex did not fall inside the edge or triangle, because in this case there must be a vertex incident to the edge or an edge incident to the triangle closer to the seed vertex. The squared distance from a vertex at location p to an axis aligned box with lowest coordinates vector l and highest coordinates vector h can be computed by

$$dist = \sum_{\alpha \in \{x,y,z\}} \max\{0, l_\alpha - p_\alpha, p_\alpha - h_\alpha\}^2. \qquad (2)$$

The distance from an edge to a cell is quite complicated to compute. We used the following simple and efficient strategy. We parameterized the edge over $\lambda \in [0,1]$ as $p = o + \lambda \cdot v$, plugged this expression into equation 2 and minimized $dist(\lambda)$ over $[0,1]$. Equation 2 was only evaluated on values where the selection of the max-function changed and on the extremes in-between. As the distance from a point moving on a straight line to a convex object can only assume one minimum in a connected region, it was sufficient to follow λ until it increased.

5 Simplification Algorithm

Our simplification algorithm can be split into the initial preprocessing phase and the decimation loop itself. The algorithm proceeds according to an increasing error, that is caused by contracting two simplices, which we call *correspondence pair*. All correspondence pairs are ordered in a priority queue.

5.1 Preprocessing

After the acceleration grid (see section 4) has been initialized, we identify the corresponding pairs for subsequent decimation operations.

For the priority queue algorithm to work properly, the operation causing the minimum error needs to be the first element in the queue. This can either be a manifold operation (edge collapse or edge flip) or a non-manifold one (vertex contraction or edge-edge contraction). As the error caused by manifold operations is not related to the distance by which the contraction vertices move, we need to consider all possible manifold-operations. In case of non-manifold operations, the caused error is at least half the distance between the contracted simplices. Therefore, we search the corresponding simplices of each vertex or edge with a maximum search distance of $2 * l$ and consider only the closest corresponding simplex. In case of the search for the corresponding simplex of a vertex, l is the length of the edge incident to the vertex that causes the minimum error if contracted. When we search for the corresponding edge of an edge, l is simply the length of the edge itself.

For each vertex v we proceed as follows:

- For all incident edges the error that would be introduced if the edge was collapsed is computed according to equation 1. We choose the edge with the minimum error ε_m and calculate its length l. Note, that the grid data structure automatically discards incident edges and triangles.
- Setting v as a seed vertex, we search in the grid with a maximum search distance $2 * l$ for the closest non-incident simplex (vertex, edge or triangle) using the algorithm described in 4.2.
- If a simplex is found within the $2 * l$ distance, we compute the error ε_n, which will be introduced by contracting vertex v with the found simplex according to 1.
- We compare ε_m and ε_n and assign the operation with the minimal error $min(\varepsilon_m, \varepsilon_n)$ to the vertex v.

For edges only non-incident edges have to be considered as candidates. For each edge e_0 with length l we search the grid with a maximum search distance of $2 * l$ using e_0 as a seed edge for the algorithm described in 4.2. During this search the algorithm guarantees that the points realizing the minimal distance between e_0 and the found edge e_1 are in the (open) interior of the two edges. For such two edges we compute the introduced error according to 1. Note, that we find the corresponding edge only for some edges.

After the search is complete, references to the found simplices are stored for all vertices and some edges. Vice versa, each simplex stores references to all vertices and edges pointing onto it.

Last but not least, the pairs are inserted into a priority queue according to their associated errors.

5.2 Decimation Loop

At each step of the decimation loop we first take the pair with the minimal error from the queue.

Prior to perform the operation determined by this pair, the following tests are performed:

- *Normal test.* Here we check that the normals of triangles affected by the operation do not change by more than α_{max}.
- *Minimum angle test.* This test is performed only if the operation is an edge flip. We check if the minimum angle among all affected triangles increases by at least β_{min}. As described in section 3.2, this avoids endless loops while allowing to improve the shape of triangles.
- *Error test.* We calculate local Hausdorff distance from original to simplified mesh after performing the operation, and check if it doesn't exceed the global simplification threshold d_{max}, as described in section 3.1.
- *Collision test.* Finally, the collision detection is done, as described by Gumhold et al. in [9]. This test allows to detect and avoid self-intersections which could occur as a result of the contraction operation.

When the operation is discarded because one of the above tests has failed, we put it into the second queue of *discarded operations*, otherwise we perform it.

After performing the operation we have to update all affected pairs. To explain this process, let's define V_c as the set of vertices that collapsed to a vertex v, E_c and T_c as the sets of edges and triangles changed by the operation, E_r and T_r as the sets of edges and triangles which were removed. We proceed as follows:

- For each vertex in V_c and each edge in $E_c \cup E_r$, we remove previous correspondences.
- We find correspondences for v and each edge in E_c as described above.
- For each vertex, which corresponds to an edge in $E_c \cup E_r$ or a triangle in $T_c \cup T_r$, we search again for the best correspondence.
- For each edge e, which corresponds to an edge from $E_c \cup E_r$, we find a new correspondence.

5.3 Double Queue Strategy

As described before, some pairs are discarded if one of the above tests fail. But it's not desirable to simply reject the operation. During simplification the neighbourhood of the considered pair might change and the reason of its rejection vanish. Therefore, we do not want to loose the operation.

On the other hand we cannot simply insert the discarded operation back into the priority queue, since its error is the smallest one and it would be taken from the queue again in the next step causing an infinite loop.

One strategy to keep invalid operations is to assign the discarded operation to all simplices that caused the invalidity. If a simplex is

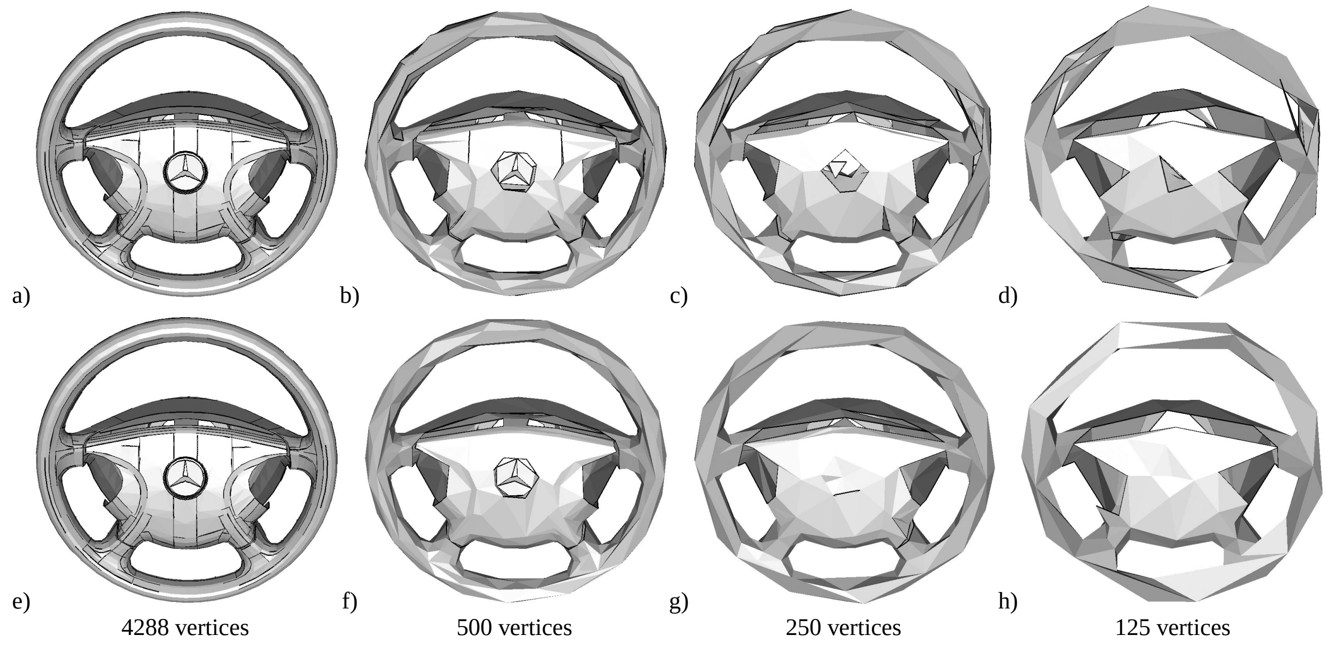


Figure 5: Simplification with only vertex pair contractions (a - d) and with generalized pair contractions (e - h). While in the first case some gaps between patches remain even at the coarsest resolution, in the second case they all are closed at the early stage.

removed or changed in a contraction operation, all of the discarded operations attached to it are reconsidered. Unfortunately, this strategy demands for complex data structures and a large amount of computations. Therefore, we used a simpler heuristic strategy.

We put each discarded operation into a second queue of *discarded operations*. We considered one randomly selected simplex from this queue once in k simplification steps and after establishing a new correspondence for it we reinserted it into the first queue. We did it in the way to ensure that the frequency of reinsertion of discarded elements into the first queue is approximately proportional to the size of the second queue. When the first element was put into the second queue we initialized k by the number of elements in the first queue and decremented it by one in each simplification step. Each time a further element was put into the second queue k was updated as follows: $k := k * \frac{N_2}{N_2+1}$, where N_2 is the number of elements in the second queue. If k became zero it was set to $\frac{N_1}{N_2}$, where N_1 is the number of elements in the first queue.

6 Applications

In many cases the well-established decimation techniques such as vertex pair contraction simplification are entirely sufficient. At the same time, for some classes of models and in some certain applications the proposed method can deliver much better results.

6.1 Controlled Topology Modifying Simplification and Mesh Repair

Meshes from different sources like remote sensing, medical scanning, CAD and even scientific computing contain degenerate faces, T-vertices, narrow gaps and cracks. In subsequent processing steps and applications including finite element analysis, surface smoothing, model simplification, stereo lithography and milling such degeneracies lead to severe artifacts, often 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. Mesh repair aims at creating a similar model but without its flaws. To achieve this goal the geometry as well as the topology of the given mesh has to be modified.

An important issue in the repair process is the ability of the user to specify the exact size of features like holes, gaps and cracks that should be removed from the model and of course, in an intuitive solution small features should disappear before larger features. Using our new generalized pair contraction algorithm this can easily be achieved. During the simplification process features are removed in the order of increasing Hausdorff distance to the original mesh and therefore according to the size of the features themselves. In order to repair a mesh, the user specifies a maximum size of features that should be removed and then simplification operations are subsequently performed until the specified threshold is reached. An example is given in Figure 5, where gaps and cracks in the triangulation of a steering wheel are successively removed and holes are closed in the expected manner. Note, that without mesh repair operations a standard simplification technique will fail to simplify the model adequately.

In addition to its ability to repair meshes in an intuitive and efficient way the new method using generalized pair contractions often results in much better decimation results than standard simplification techniques with vertex-vertex contractions only (see Figure 6). Furthermore, our algorithm is particularly useful for the simplification of the models consisting of a large number of unconnected parts, such as industrial machines. Such models are generated by CAD/CAM and other geometric modeling tools.

6.2 Out-of-core Simplification

Since the generalized pair contractions close gaps more efficiently than vertex pair contractions a simple and fast out-of-core simplification is possible by cutting the model into subparts and simplifying each subpart independently. Gaps are automatically closed when all

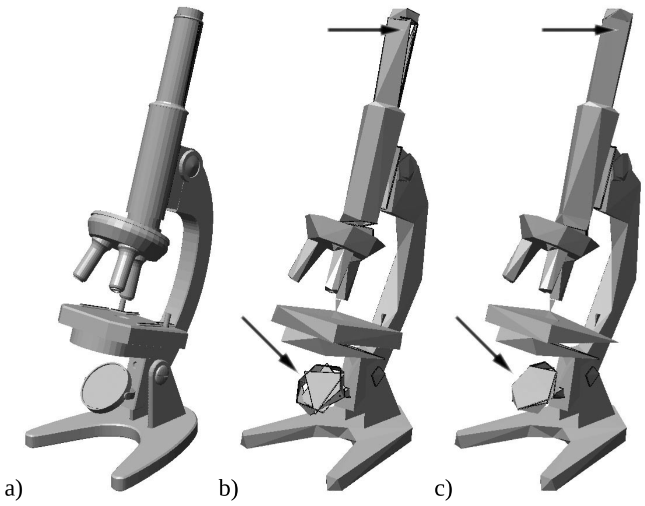


Figure 6: The original model of microscope with 4467 vertices (a), simplified with only vertex pair contractions (b) and with generalized pair contractions (c) to 250 vertices.

subparts are simplified together. To simplify gigabyte models the partitioning and independent simplification is applied recursively.

The model is partitioned by cutting the geometry of the node into eight subparts and storing it in its children if it contains more than T_{max} triangles. The partitioning is repeated until no node has to be cut anymore. If no geometry is contained in a node it is marked and not partitioned further. In this way a sparse octree is build.

Since the whole geometry of a node does generally not fit into the main memory, the vertices and normals of the mesh are stored in blocks and swapped in and out from disk using a last-recently-used (LRU) algorithm. The indices of the triangles need not to be stored in memory and therefore, can be streamed from the geometry file of the node to the files of its children. This is accomplished by loading the actual triangle from the geometry file of the node, cutting it and then saving the generated triangles in the child geometry files. After the triangle is cut it is not needed any more. Therefore, only the actual triangle and the triangles generated from it are stored in memory. When first saving all triangles in the root node, the vertex normals are calculated.

At each partitioning step every triangle is cut with the three planes dividing the node into its children and the resulting triangles are stored in the appropriate geometry files. When a triangle edge is cut, the normal of the new point is calculated by linear interpolation. Note that new vertices may have the same coordinates as existing vertices, but this is resolved when the whole tree is build. After partitioning the triangles of a node and storing it in its children, the geometry file of this node is not used any more and is deleted.

When the partitioning is complete new indices for the leaf node triangles are calculated and duplicate points are removed.

The total complexity of the partitioning algorithm is $O(n\log n)$, since on each level of the octree all triangles need to be processed once.

After the partitioning the geometry contained in the leafs of the octree is stored on disk. Starting from the geometry of these nodes the model is simplified recursively from bottom to top with a constant resolution *res* depending on the node size using the following algorithm:

- At every level of the octree gather the simplified geometry from all child nodes that are two levels below the current node (or the original geometry if there is no pre-simplified geometry at this depth). Its approximation error ε_{prev} is then the maximum error of the simplified geometry in these child nodes or zero.
- Simplify the resulting geometry as long as the Hausdorff distance ε_h to the gathered geometry is less than $\varepsilon_s = \frac{e_{node}}{res} - \varepsilon_{prev}$, where e_{node} is the edge length of the currents nodes bounding cube and *res* is the desired resolution in fractions of e_{node}.
- Store $\varepsilon = \varepsilon_h + \varepsilon_{prev}$ as approximation error in the current node.

By using the children at two levels below the current node instead of its direct children the simplified geometry contains less triangles, since the approximation of the real geometric error is better. This is due to the fact that the difference between the estimated geometric error ε and the real geometric error ε_{real} is low, since:

$$\varepsilon_{real} \geq \varepsilon_s = \frac{e_{node}}{res} - \varepsilon_{prev} \geq \frac{e_{node}}{res} - \frac{e_{node}}{4\cdot res} = \frac{3}{4}\frac{e_{node}}{res} = \frac{3}{4}\varepsilon \quad (3)$$

and thus $\frac{3}{4}\varepsilon \leq \varepsilon_{real} \leq \varepsilon$.

Starting with the combination of already simplified geometry greatly reduces the computation cost and still leads to high quality drastic simplifications. This is due to the fact that because of the constant resolution the number of triangles in the gathered geometry remains almost constant independent from the depth of the current node and therefore, the number of triangles in the base geometry of this part of the object. This means that the total simplification time depends only linearly on the number of leaf nodes and thus linearly on the number of triangles in the base geometry. Therefore, the total time for this out-of-core simplification algorithm sums up to $O(n\log n)$.

7 Results

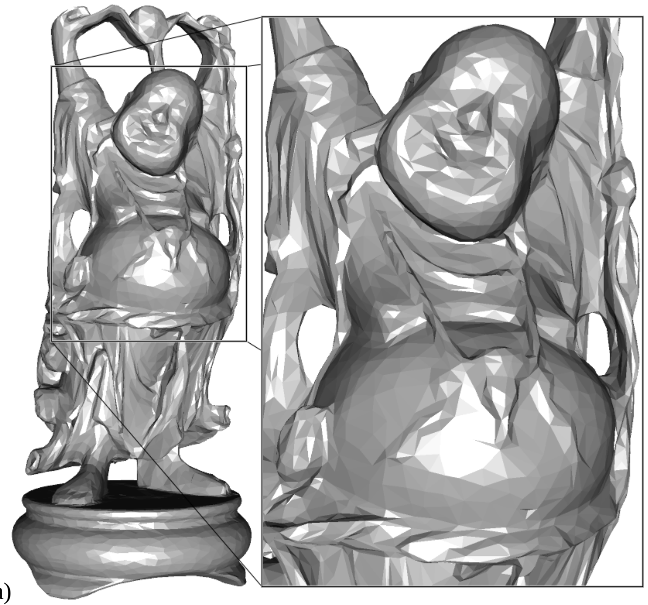
The first example in this section demonstrates the model of a steering wheel[1]. A lot of artifacts, resulting from improper tessellation of a trimmed NURBS surfaces, are present in this model. Figure 5 (a - d) shows several steps of a simplification algorithm which performs only vertex pair contractions. Many gaps between separate patches, hardly recognizable in the original model, have not been sewn together. Some of them have been transformed to real holes and remain even in a coarse model with only 125 vertices. In contrary, the new simplification algorithm which performs also generalized pair contractions allows to close such narrow gaps already at the early stages as shown in Figure 5 (e - h). In a model reduced to 500 vertices all these artifacts have been eliminated, only the large holes inherent to the model remain.

Figure 6 shows the model of a microscope. The original model in 6a looks perfect but some triangles in the upper part of the tube are missing. Figure 6b depicts the model simplified with vertex pair contractions only. As can be seen, after simplification to 250 vertices the holes in the tube not only remain but have increased. Furthermore, the mirror at the bottom originally consisting of several independent parts becomes strongly corrupted. The model in 6c was simplified with generalized pair contractions. As expected, the holes in the tube were closed and also the mirror is simplified adequately.

Finally we compare our out-of-core simplification (Figure 7b) with standard quadric error metric simplification (Figure 7a). Both simplified models have 18338 faces. At the same time, their corresponding relative Hausdorff errors (over the bounding box diagonal) are 0.26% and 0.79% respectively. It is clearly visible, that

[1]This model was kindly provided by DaimlerChrysler AG

a)



b)

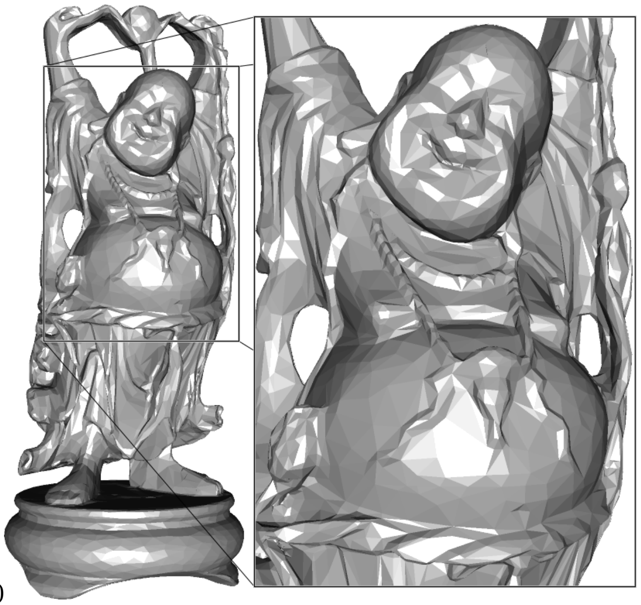


Figure 7: Simplification results for the Happy Buddha model: a) QEM simplification; b) out-of-core simplification with generalized pair contractions. Both models were reduced to 18338 faces. Their corresponding relative Hausdorff maximum errors (over the bounding box diagonal) are 0.79% and 0.26% respectively.

details (e.g. the necklace and the mouth) and silhouettes are better preserved by our algorithm. The very small error of our method is a result of rejection of contraction operations which would introduce a too large Hausdorff error (see section 3.1).

8 Conclusion

In this work we presented an important strategy to generate high quality simplified models. We introduced the generalized pair contraction operations. They not only allow to remove gaps and holes, but also seamlessly integrate the automatic connection of close surface parts in the most general setting and the resolution of initial self-intersections during the simplification process.

In future work we want to find out how generalized pair contractions work in combination with different error metrics.

References

[1] Gill Barequet and Subodh Kumar. Repairing cad models. In *IEEE Visualization '97*, pages 363–370. IEEE, November 1997. ISBN 0-58113-011-2.

[2] Pavel Borodin, Marcin Novotni, and Reinhard Klein. Progressive gap closing for mesh repairing. In *Advances in Modelling, Animation and Rendering*, pages 201–213. Springer, July 2002.

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

[4] Paolo Cignoni, Claudio Rocchini, Claudio Montani, and Roberto Scopigno. External memory management and simplification of huge meshes. In *IEEE Transactions on Visualization and Computer Graphics*. IEEE, 2002.

[5] Jihad El-Sana and Yi-Jen Chiang. External memory view-dependent simplification and rendering. *Computer Graphics Forum*, 19(3), 2000.

[6] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31(Annual Conference Series):209–216, 1997.

[7] Michael Garland and Eric Shaffer. A multiphase approach to efficient surface simplification. In *IEEE Visualization*, pages 117–124. IEEE, 2003.

[8] André 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):136–151, 2001.

[9] Stefan Gumhold, Pavel Borodin, and Reinhard Klein. Intersection free simplification. In *The 4th Israel-Korea Bi-National Conference on Geometric Modeling and Computer Graphics*, pages 11–16, February 2003.

[10] Igor Guskov and Zoe J. Wood. Topological noise removal. In *Graphics Interface*, 2001.

[11] Hugues Hoppe. Smooth view-dependant level-of-detail control and its application to terrain rendering. In *IEEE Visualization*, pages 35–52. IEEE, 1998.

[12] Martin Isenburg, Peter Lindstrom, Stefan Gumhold, and Jack Snoeyink. Large mesh simplification using processing sequences. In *Visualization 2003*, 2003. to appear.

[13] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *ACM Siggraph*, 2000.

[14] Peter Lindstrom and Claudio T. Silva. A memory insensitive technique for large model simplification. In *IEEE Visualization*. IEEE, 2001.

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

[16] 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.

[17] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques, 1999.

[18] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH*, pages 217–224, 1997.

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

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

[21] Jianhua Wu and Leif Kobbelt. A stream algorithm for the decimation of massive meshes. In *Graphics Interface Proceedings*, 2003.

[22] Gabriel Zachmann. Optimizing the collision detection pipeline. In *The First International Game Technology Conference GTEC*, January 2001.