Efficient Decimation of Complex Triangle Meshes

Swen Campagna Leif Kobbelt Hans-Peter Seidel

Technical Report 3/98
Computer Graphics Group
University Erlangen-Nürnberg, Germany *

Abstract

Due to their simplicity triangle meshes are used to represent surfaces in many applications. Since the number of triangles often goes beyond the capabilities of computer graphics hardware, a large variety of mesh simplification algorithms has been proposed in the last years. In this paper we identify major requirements for the practical usability of general purpose mesh reduction algorithms. The driving idea is to understand mesh reduction algorithms as a software extension to make more complex meshes accessible with limited hardware resources. We show how these requirements can be efficiently attained and discuss implementation aspects in detail. We present a mesh decimation scheme that fulfills these design goals and which has already been evaluated by several users from different application areas. We apply this algorithm to typical meshes to demonstrate its performance.

1 Introduction

Triangle meshes are a common way to represent surfaces in computer graphics and geometric modeling. A large variety of applications use triangles as basic geometric primitives to visualize and represent their results. This includes iso-surface extraction from volume data, medical applications, terrain visualization, engineering applications and many more. Since computer graphics hardware accelerates the display of triangles meshes, even smooth surface representations are converted into that format, e.g., for interactive display of CAD models, architectural walkthroughs, or virtual reality. Thus, contemporary software packages directly rely on triangle meshes as a universal surface representation, e.g., animation- [1] or mesh modeling [20, 11] systems. Unfortunately, in most cases the triangle count of detailed meshes is very high. This prohibits straight forward solutions for tasks like interactive visualization and inspection.

Since the complexity of triangle meshes apparently grows faster than the capabilities of computer graphics hardware, there is much ongoing effort in the area of mesh simplification. Many algorithms have been proposed for the decimation of triangle meshes. [15, 16, 13] give an overview over the large variety of techniques. Recently, an attempt was made to investigate the generic nature of mesh decimation algorithms [10]. "Simplification envelopes" [3] provide a general framework for algorithms maintaining a global error bound.

Up to now, research in the field of mesh simplification has focused on the development of new decimation algorithms and seems to have gained a status of maturity. For every sort of application a suitable simplification algorithm can be found. All of them are limited by the tradeoff between speed and the quality of the resulting mesh.

*Computer Sciences Department (IMMD9), University of Erlangen-Nürnberg, Am Weichselgarten 9, 91058 Erlangen, Germany, Email: {campagna|kobbelt|seidel}@informatik.uni-erlangen.de

We claim that besides specific technical properties the following aspects are essential for practical use:

- Robustness of the algorithm: most real world data contain topological inconsistencies, e.g. complex vertices. Thus, algorithms assuming the input data coming as (bounded) 2-manifolds would fail. This aspect can be handled by topology modifying algorithms [5, 17, 12, 7].
- Intuitive parameters to steer the reduction: most people using mesh reduction software are not experts in the field of computer graphics algorithms. Thus, intuitive parameters to control the simplification process are strongly required.
- Thoughtful use of computer hardware resources: mesh reduction techniques have to be fast, e.g. with respect to Schroeder's recent definition [17] of 10^8 reduced triangles per day. Yet, the reduction process has to fit into the memory of current computer systems.

The performance of computer systems is increasing steadily, considering CPU speed, the size of main memory, and the triangle throughput of computer graphics hardware. Looking at contemporary systems, PCs or graphics workstations, we can observe that all of them are able to store triangle meshes into their main memory which they would never be able to display with acceptable frame rates. It makes sense to extrapolate this observation into the future, i.e., the ratio of main memory size to graphics performance will not change significantly.

Since computers can easily handle larger meshes than they can display actually, mesh reduction algorithms are required to bridge the gap. This can be considered as a pipelining process:

A triangle mesh is generated and modified in the main memory and displayed through the use of a mesh reduction algorithm which has to be performed on the same computer system. Hence, *the maximum size of a triangle mesh that may be displayed on a specific computer is obviously limited by the number of triangles that may be processed by a mesh simplification algorithm on the same system*.

In the following sections we discuss in detail, how each of the above requirements for the practical usability of a simplification algorithm can be achieved. We combine these partial solutions to a mesh simplification algorithm and verify the achievement of the design goals by applying our implementation to data sets from different application areas.

2 Robustness

Besides wavelet-methods [6, 4], simplification envelopes [3], and re-tiling of surfaces [19], most known mesh reduction algorithms iteratively reduce a given mesh. A sequence of simple topological

operations is applied to the current mesh removing certain geometric entities in each step. Possible basic operations include vertex removal and re-triangulation [18], general edge collapse [8], half edge collapse [10], and vertex collapse [5, 17].

To the best of our knowledge, every simplification algorithm chooses one type of a topological operation and uses only this one for the whole reduction. Most current iterative mesh decimation algorithms use a priority queue to obtain optimal approximations to the original mesh. Their generic structure is as follows:

```
For all geometric entities {
  Measure cost for applying operator
  Put result into priority queue
}
Loop until queue empty {
  Perform operator with least cost
  Update costs in queue
}
```

The most costly part is measuring and updating the cost for the potential application of a topological operation. It is intrinsic to each algorithm, how the priority for every possible operation is calculated. Possible topological inconsistencies in the mesh complicate the computation even more. Topological problems may include, e.g., complex vertices [18] or edges, i.e., the mesh is not 2-manifold [12]. Further problems are distinct topological entities that represent the same geometry, e.g. two distinct triangles that use the same vertices.

Topological inconsistencies may occur for different reasons. They could be used to represent a feature or preprocessing algorithms generate inconsistent meshes. We encountered many cases where the data could have been represented by an ideal 2-manifold. But for algorithmic reasons the generating programs, e.g. data acquisition tools or surface meshers, produced complex entities (cf. Fig. 5).

Different strategies can be used to cope with possible topological inconsistencies:

- The mesh can be repaired in a preprocessing step [2]. Since this is usually done by a separate algorithm, valuable information may be lost. But this allows an algorithm to make certain simplifying assumptions.
- The handling of all possible problems may be incorporated into the algorithm. Unfortunately, this complicates implementation and the algorithm itself becomes more involved.
- Object-oriented techniques can be used to separate the data-structures representing the mesh from the algorithms working on it.

Since the last item combines the advantages of the others, we will discuss this strategy in detail next.

Fig. 1 shows how to separate the raw mesh from algorithms working on that data by object-oriented software design. The raw mesh data is encapsulated into a data-structure that may exploit meta-knowledge about that data. This encapsulating object has to provide the following interface:

- Access to geometric entities, e.g. lists of vertices, edges, or triangles.
- Methods to inform the data structure about requirements of the algorithm, e.g. 2-manifolds.
- Methods to ask the data structure, if a topological operation is possible with respect to the given requirements.
- Methods to commit basic operations on the data.

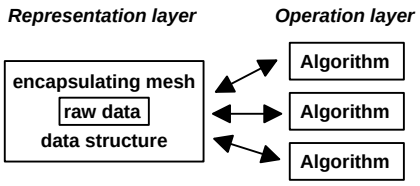


Figure 1: Separating the triangle mesh from the algorithms working on that data.

The encapsulating data structure can treat topological problems in several ways. It can try to repair them on-the-fly, e.g. by splitting complex vertices [17]. Repairing in a preprocessing step could incorporate the methods presented in [2]. Finally, it can simply mask such regions of the mesh, i.e., not allow the modification of the affected sub-mesh.

The separation into a representation and operation layer does not change the structure of an algorithm, but simply decouples independent aspects of processing triangles meshes. Every iterative mesh decimation algorithm can be formulated using such an encapsulating data structure. Thus, it makes sense to encapsulate everything to treat complicated and complex meshes into a separate class. The algorithms can then abstract from these basic tasks and concentrate on their main issue. This eases implementation aspects and helps to make programs more robust in the sense of fault-tolerant software.

3 Intuitive parameters

To steer the iterative mesh reduction and to terminate the simplification process, it is necessary that the user defines some parameters. People familiar with computer graphics techniques are able to use such algorithms after a short time. But in practice, people have to use these programs who may not understand the underlying algorithm and the parameters for geometric or topologic values. Thus, we want to draw the attention on this aspect of mesh decimation algorithms in this section.

As described previously, most current mesh reduction algorithms use a priority queue to decide which topological operation to perform next. This means that all currently possible operations have to be considered and a cost has to be computed for performing this operation. This cost is a scalar value that is used to sort the set of candidates. Since most algorithms impose several conditions on potential operations, the problem arises, how to combine these different measures into a single scalar value. Several strategies are possible:

- The easiest case occurs if only a single measure is used that directly provides a scalar value, e.g. the geometric deviation of the current from the original mesh [9].
- If the cost function considers more than one aspect, each of them providing scalar values, these can be combined by a weighted sum. E.g., [14] combines the geometric deviation and a measure of the tilt of the surface normals, or [8] uses a scalar energy function that incorporates a large number of different aspects of the generated mesh.
- One adequate aspect is chosen to provide the scalar value, while the others are used for a binary decision that define the *set of candidates* which are actually considered for elimination.

The first strategy considers only a small portion of the information given in a mesh. The second strategy requires the user to choose the weighting factors that may not be intuitive. Further, these factors need not to be invariant under geometric transformation of the

mesh. Scaling of objects to the unit cube does not help since the bounding box of an object usually varies under rotation. We find the third strategy most promising because of several reasons. First, it reduces the set of candidates that have to be further considered by using binary decisions. This reduces computation costs. Further, it makes the need for counter-intuitive weighting factors unnecessary. We have applied this strategy in our implementation of the algorithm by Ronfard and Rossignac [14] and found the resulting program much more easy to handle.

Usually, authors of mesh reduction algorithms can produce best results when using their according implementations, since they know exactly how the parameters affect the reduction process. But non-experts often have difficulties to find the optimal setting. Keeping this in mind, we now want to describe the ingredients for a mesh simplification algorithm with only very few *intuitive* parameters by following the third of the above strategies.

First, we use the half-edge collapse as topological operator, i.e., vertices are pulled into one of their neighbors, since it eliminates degrees of freedom that would have to be optimized involving further user-adjustable parameters.

Next, we use the one-sided Hausdorff distance to measure the deviation of the current mesh from the original vertices [10] as a binary decision. Thus, the user only has to choose an intuitive global error bound to identify the current set of candidate operations for the reduction algorithm. If a specific reduction rate is desired, the program can increase an initial (e.g. automatically chosen) error bound until the desired reduction rate is reached.

Finally, a fairing oracle is used to measure the quality of the generated surface [10] and to steer the greedy reduction algorithm, i.e., to compute the priority of a potential half-edge collapse in the candidate set. We let the user choose between the use of an order 1 or order 2 fairing oracle, since this can be translated into "his language". Order 1 should be used for technical applications or for meshes to be passed to further processing programs, since it is related to local distortion of the mesh (first order derivatives). Order 2 should be chosen for the visualization of meshes, because it considers the local curvature of the modified mesh (second order derivatives). Note that the fairing oracle measures the quality of the modified surface. Hence, "better" configurations are chosen automatically because of the use of the priority queue without the need for any further parameters. Since our implementation of the fairing oracles is closely related to performance issues, we postpone the details to the next section.

4 Thoughtful use of hardware resources

In the previous section, we have presented the ingredients for an easy to use mesh reduction algorithm. In this section, we focus on implementation issues beyond the separation into an encapsulating data structure and the abstract reduction algorithm as discussed in Section 2. We show, how the memory resources can be efficiently used by reducing edge-based algorithms to more compact vertex-based structures. Further, we show how high performance can be achieved at the same time by efficiently implementing the above ingredients.

A triangle mesh with n vertices has about $3n$ undirected and about $6n$ directed edges (Euler's formula). Thus, each iterative mesh simplification algorithm using a priority queue for the potential vertex removals or edge collapses as topological operations, has to consider that number of geometric entities. A straight forward implementation would have to store a maximum of the same number of candidate topological operations in the queue.

Collecting all edges emanating from one vertex reduces the edge-based oracles and operations to vertex-based ones by local pre-selection of the best candidate. If one edge collapse is performed, all edges starting from the removed vertex have to be removed from the priority queue anyway. Thus, it is not necessary to store all valid potential edge collapses in the priority queue, but only the best one for each vertex. This enables the efficient use of the half-edge collapse as topological operator for the iterative mesh reduction, since only a priority queue for n potential entries is required instead of $6n$.

Efficiency gains result from the exploitation of geometric coherence which avoids recalculation of intermediate results. Since the priorities of all edges emanating from one vertex are calculated at the same time, intermediate information can be stored and reused for neighboring edges.

With this knowledge we can efficiently implement functions `calcVertexPrio()` and `updateVertexPrio()` for calculating and updating the priorities of a vertex, i.e., for all edges emanating from the vertex, and for identifying the "best" edge. `updateVertexPrio()` may be used, if it is known that the geometric deviation of the modified geometry has not changed since the last update and thus only re-calculation of the fairing oracle is necessary. Using these functions, our mesh simplification algorithm has the following structure:

```
ALGORITHM: simplify mesh

INPUT:
  M: original triangle mesh
  d: max. geometric deviation
  o: order 1 or 2

OUTPUT:
  R: reduced triangle mesh

  For all vertices v of M {
    p = calcVertexPrio( v, d, o );
    add (p,v) to queue;
  }
  Loop until queue empty {
    get next vertex v from queue;
    if ( removal of v possible ) {
      performCollapse( v->e );
      For all vertices v' that require
      recalculation of the priority {
         p = calcVertexPrio( v', d, o );
         update (p,v') in queue;
      }
      For all vertices v' that require
      update of the priority {
         p = updateVertexPrio( v', d, o );
         update (p,v') in queue;
      }
    }
  }
```

Our order 1 fairing oracle is implemented as follows. It uses the function `round(t)` to determine the "roundness" of a triangle t, i.e., the ratio of the longest edge to the radius of the inner circle. First, we calculate for all triangles t neighboring the vertex v the maximum value r_o of their roundness. This can be done in a preprocessing-step, i.e., at the beginning of `calcVertexPrio(v)`, since this value is the same for all potential edge collapse operations starting at v. Next, we calculate the maximum value r_n of the roundness for the modified triangles. If $r_n < r_o$, we assign this decrease to the priority for performing that edge collapse. Otherwise, we still allow that collapse, if the value of r_n is below a prescribed maximum value, but assign those collapses a priority less than that of the "enhancing" ones. This gives an expert user the freedom to adjust one further intuitive parameter, if he really desires, but frees non-expert users from the need to handle that parameter. The support of this oracle is shown in Fig. 2.

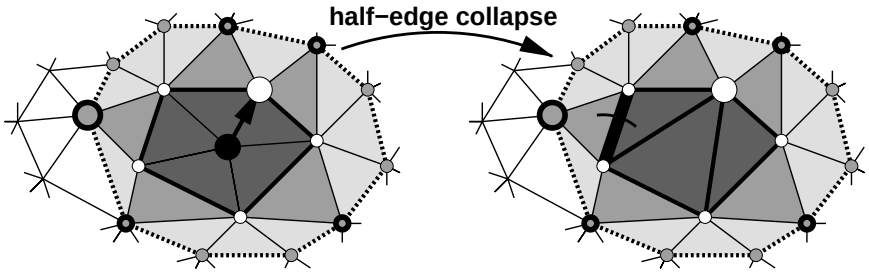


Figure 2: After performing an half-edge collapse, the vertices (white) on the border of the modified geometry (dark gray, the support of the order 1 fairing oracle) have to be recalculated. If using an order 2 fairing oracle, the priorities of further vertices (gray) have to be updated as well, because of the larger support (dark, medium, and light gray). Since our order 2 fairing oracle is designed to have a smaller support (not including the light gray regions), fewer priorities have to be updated (only the bold outlined gray vertices). E.g., the priority of the fat gray vertex, despite only a single dihedral angle changes (arc in the right figure). This fact can be exploited to speed up the update of priorities.

Our order 2 fairing oracle is efficiently implemented in the following way. We sum over the dihedral angles within the modified sub-mesh and those of the modified triangles and their neighbors that are not changed. The larger this sum is, the more cost is assigned to a potential edge collapse. For expert users, we provide the additional parameter α that disallows edge collapses that would create geometry with a dihedral angle larger than α. As in the order 1 case, we allow worse angles as long as they improve the situation locally. For non-expert users, we default the parameter α to $\pi/2$ to avoid degeneration of the geometry. The support of this order 2 fairing oracle is again shown in Fig. 2.

The performance-bottleneck for the order 2 fairing oracle is the fact that a large number of priorities have to be updated after each collapse operation. As discussed in the caption of Fig. 2, we reduced the number of vertices that need to be updated by the above implementation of our order 2 fairing oracle. Computations can be further reduced by the heuristic assumption that the edge with the least cost starting from such a vertex remains the one with the least cost (cf. Fig 2) and no local search to find the "best" edge is necessary. This assumption is exact in most cases. Thus, only the priorities of some edges have to be updated. In the next section we verify that this strategy both speeds up the performance and still provides good results.

5 Results

We verify the usability of our simplification algorithm by applying it to meshes of different application areas in this section. All times are benchmarked on a SGI, R10000, 195MHz.

First, we used an iso-surface extracted from volume data. The original mesh consists of 81,132 triangles (cf. Fig. 3a). Table 1 shows statistics for applying our reduction algorithm with various error bounds and using either order 1, order 2, or fast order 2 fairing oracles. Our fast update strategy for the order 2 oracle generates results that are of the same visual quality, but it is clearly faster. Fig. 3b-c show reduced meshes for a global error bound of $\varepsilon = .01$ using order 1 or order 2 fairing oracles, respectively. The order 1 fairing oracle does not eliminate as many triangles as the order 2 method, since the triangles do not have the freedom to elongate and thus adopt to the local geometry. Because there is no need for updating as many vertices as for the order 2 method, the order 1 method is clearly the fastest way to reduce a mesh.

We applied our algorithm to a large variety of further models and got equally satisfying results. E.g., Fig. 4 shows a human jaw with teethes from a medical application (233,316 triangles). Table 2 gives the reduction statistics for that mesh. Fig. 6 shows the Stanford-buddha that has been generated by merging different scans (1,087,716 triangles), and Table 3 provides the statistics.

oracle	ε	#Δ coarse	Δ/sec.	$10^8\,\Delta$/day
order 1	.01	28,270 (34.8%)	2634	2.28
	.1	3,784 (4.66%)	1992	1.72
	1	416 (0.51%)	1445	1.25
order 2	.01	23,068 (28.4%)	1659	1.43
	.1	3,222 (3.97%)	1099	0.95
	1	344 (0.42%)	784	0.68
order 2 fast update	.01	23,110 (28.5%)	2181	1.88
	.1	3,214 (3.96%)	1602	1.38
	1	340 (0.42%)	1190	1.03

Table 1: Reduction statistics for the iso-surface shown in Fig. 3a consisting of 81,132 triangles (bounding box size: 46 x 46 x 65). The user supplies the choice of the order and the global error bound. Given is the number of triangles the algorithm generates and performance timings for removed triangles per second and day. The reduction process requires about 10MB memory.

oracle	ε	#Δ coarse	Δ/sec.	$10^8\,\Delta$/day
order 1	10^{-3}	66,348 (28.4%)	2234	1.93
	10^{-2}	8,238 (3.53%)	1399	1.21
	10^{-1}	620 (0.27%)	999	0.86
order 2 fast update	10^{-3}	53,206 (22.8%)	1880	1.62
	10^{-2}	5,272 (2.26%)	902	0.78
	10^{-1}	330 (0.14%)	457	0.39

Table 2: Reduction statistics for the jaw and teeth model consisting of 233,316 triangles (bounding box size is 4.16 x 7.07 x 5.72). The reduction process requires about 26MB memory.

oracle	ε	#Δ coarse	Δ/sec.	$10^8\,\Delta$/day
order 2	10^{-3}	286,646 (26.4%)	2345	2.03
fast	10^{-2}	30,404 (2.80%)	1542	1.33
update	10^{-1}	3,774 (0.35%)	1004	0.87

Table 3: Reduction statistics for the scanned buddha statue consisting of 1,087,716 triangles (bounding box size is 8.13 x 19.8 x 8.14). The reduction process requires about 117MB memory.

Note that our algorithm clearly meets the Schroeder bound of 10^8 reduced triangles per day [17] even for high reduction rates. For extreme reduction rates, the performance drops below, because of the expensive geometric deviation test (Hausdorff). But it still achieves a competitive triangle per second rate.

The Buddha mesh example demonstrates that even very large

meshes can be processed within a PC's memory. Hence, the proposed mesh reduction scheme opens the PC platform to applications that have to visualize this kind of data sets without requiring expensive graphics hardware. Yet the scheme provides sophisticated features like global error tolerances and fairness control.

Our current implementation uses a maximum of about 110 bytes per input triangle to store geometric and topological information as well as redundant information to speed up the evaluation of the global distance measure and the fairness oracle. If we trade redundancy for computing time by not caching any intermediate results, we can reduce the memory requirements down to 65 bytes per triangle (assuming 4 bytes pointers and integers).

Finally, we should note that by the use of the half-edge collapse our algorithm can be used to generate a compact progressive mesh representation [8] of a triangle mesh. This enables a large variety of applications, including hierarchical representation, progressive transmission, view-dependent rendering, or load adaptive display.

6 Conclusions

We have identified relevant requirements for the practical usability of mesh reduction algorithms: robustness, intuitive parameters, and scalable performance with respect to both CPU and memory requirements.

We proposed to encapsulate the raw mesh data into a data structure to simplify algorithms processing triangular meshes by abstracting from special cases. We discussed, how mesh simplification algorithms can be organized to enable easy usability even for non-expert users, while still leaving enough degrees of freedom for experienced users. Finally, we described implementation aspects to efficiently attain our recommendations. This includes how to reduce edge-based algorithms to a vertex-based structure to minimize memory requirements, and fast higher-order surface fairing methods.

Our implementation provides high CPU-performance, more than two times the Schroeder bound of 10^8 removed triangles per day, and may be run even on computer systems with limited memory. The results can clearly compete with other algorithms while not requiring a single parameter to be adjusted.

References

[1] Autodesk. 3D Studio MAXX.

[2] Gill Barequet and Subodh Kumar. Repairing CAD Models. In *IEEE Visualization '97, Conference Proceedings*, pages 363–370, 1997.

[3] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Grek Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification Envelopes. In *Proceedings of SIGGRAPH '96 (New Orleans, Louisiana, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 119–128, August 1996.

[4] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In *Proceedings of SIGGRAPH '95 (Los Angeles, California, August 6–11, 1995)*, Computer Graphics Proceedings, Annual Conference Series, pages 173–182, August 1995.

[5] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH '97 (Los Angeles, California, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 209–218, August 1997.

[6] M. H. Gross, R. Gatti, and O. Staadt. Fast Multiresolution Surface Meshing. In *IEEE Visualization '95, Conference Proceedings*, pages 135–142, 1995.

[7] Taosong He, Lichan Hong, Amitabh Varshney, and Sidney W. Wang. Controlled Topology Simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–183, June 1996.

[8] Hugues Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH '96 (New Orleans, Louisiana, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 99–108, August 1996.

[9] Reinhard Klein, Gunther Liebich, and W. Straßer. Mesh Reduction with Error Control. In *IEEE Visualization '96, Conference Proceedings*, pages 311–318, 1996.

[10] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A General Framework for Mesh Decimation. to appear in conference proceedings of Graphics Interface '98.

[11] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive Multi-Resolution Modeling on Arbitrary Meshes. to appear in conference proceedings of SIGGRAPH '98.

[12] Jovan Popović and Hugues Hoppe. Progressive Simplicial Complexes. In *Proceedings of SIGGRAPH '97 (Los Angeles, California, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 217–224, August 1997.

[13] Enrico Puppo and Roberto Scopigno. Simplification, LOD and Multiresolution Principles and Applications, 1997. Tutorial Eurographics '97.

[14] Rémi Ronfard and Jarek Rossignac. Full-Range Approximation of Triangulated Polyhedra. In *Computer Graphics Forum, Proceedings of Eurographics '96*, pages C67–C76, 1996.

[15] Jarek Rossignac. Simplification and Compression of 3D Scenes, 1997. Tutorial Eurographics '97.

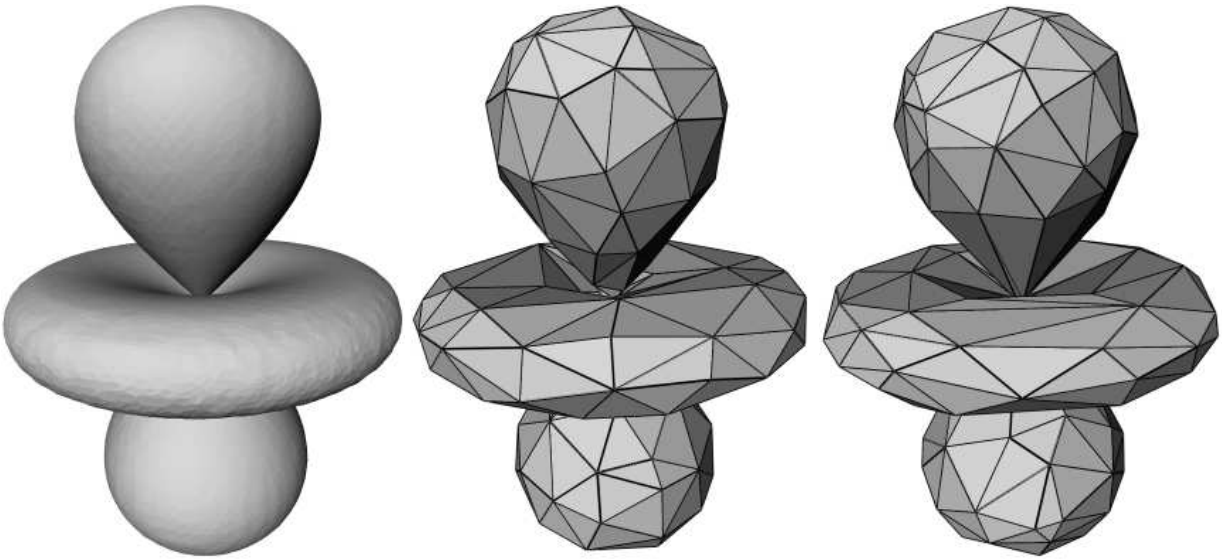
[16] Will Schroeder. Polygon Reduction Techniques, 1995. IEEE Visualization '95. Advanced Techniques for Scientific Visualization, Section 1.

[17] William J. Schroeder. A Topology Modifying Progressive Decimation Algorithm. In *IEEE Visualization '97, Conference Proceedings*, pages 205–212, 1997.

[18] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[19] Greg Turk. Re-Tiling Polygonal Surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[20] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive Multiresolution Mesh Editing. In *Proceedings of SIGGRAPH '97 (Los Angeles, California, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 259–268, August 1997.



(a) (b) (c)

Figure 3: An iso-surface extracted from volume data from a numerical simulation (courtesy Roberto Grosso). (a) shows the original mesh with 81,132 triangles, (b) and (c) simplified surfaces using order 1 or order 2 fairing oracles, respectively. Notice in (c), how the triangles near the center elongate and thus adopt to the local geometry.

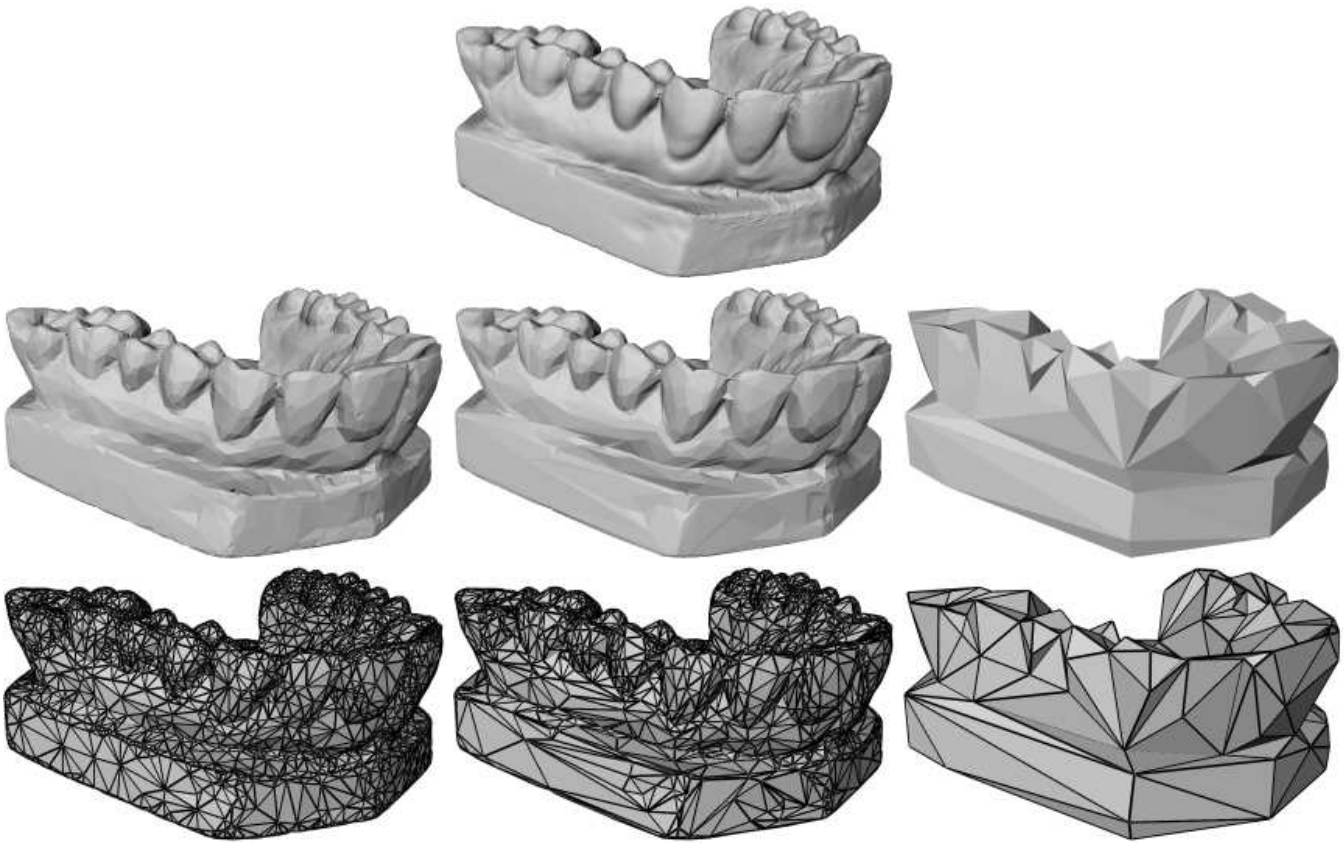


Figure 4: A surface from a medical application (courtesy Cyberware). Top: original mesh, 233,316 triangles. Left column: simplified mesh using order 1 fairing oracle. Note the uniform shape and distribution of the triangles all over the mesh. Middle column: using the order 2 fairing oracle at the same global error bound. Right column: again using order 2, but a larger error bound.

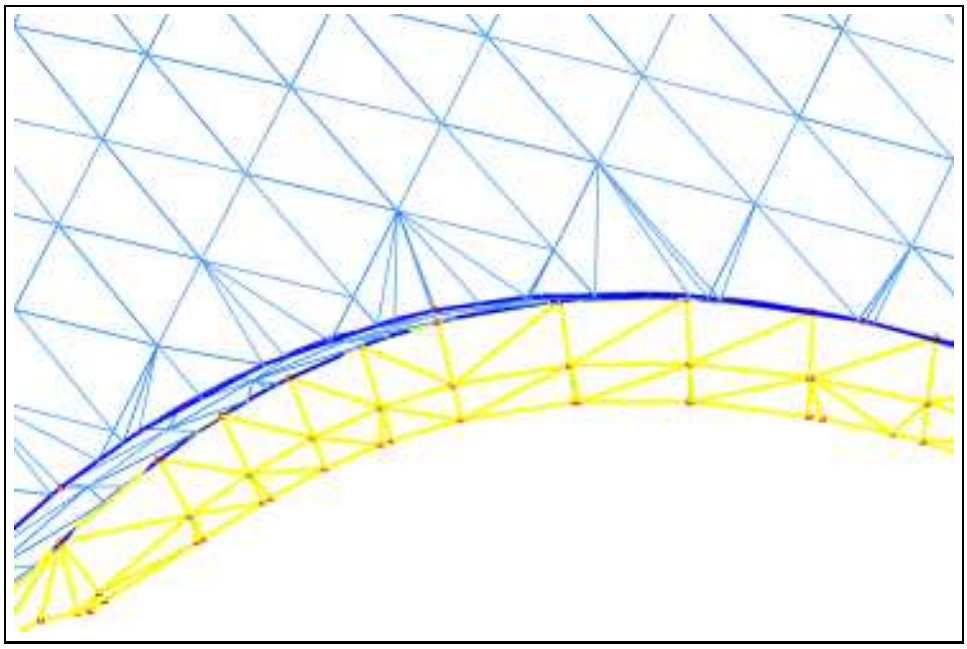


Figure 5: A typical triangle mesh generated by a commercial surface meshing software for CAD surface data representing a folded metal part. The software correctly unified geometrically identical vertices but coinciding edges and triangles were not detected. Dark blue: topological boundaries, yellow: complex edges, red: complex vertices.

(a) (b) (c) (d)

Figure 6: The complex Stanford-buddha statue, generated by merging multiple scans. (a) original mesh with 1,087,716 triangles, (b-d) simplified meshes using the order 2 oracle with fast update, 286,578, 30,392, and 3,774 triangles, respectively. Note how high-frequency detail is removed first due to the fairing-oracle.