Data–Dependent Surface Simplification

Karin Frank, Ulrich Lang

HLRS
Computing Center University Stuttgart
Allmandring 30a, D-70550 Stuttgart, Germany

Abstract In Scientific Visualization, surfaces have often attached data, e. g. cutting surfaces or isosurfaces in numerical simulations with multiple data components. These surfaces can be e. g. the output of a marching cubes algorithm which produces a large number of very small triangles. Existing triangle decimation algorithms use purely geometric criteria to simplify oversampled surfaces. This approach can lead to coarse representations of the surface in areas with high data gradients, thus loosing important information.
In this paper, a data-dependent reduction algorithm for arbitrary triangulated surfaces is presented using besides geometric criteria a gradient approximation of the data to define the order of geometric elements to be removed. Examples show that the algorithm works sufficiently fast to be used interactively in a VR environment and allows relatively high reduction rates keeping a good quality representation of the surface.

1 Introduction

In Scientific Visualization, a common approach consists in extracting isosurfaces and cutting surfaces from a data set in order to detect essential features of the data. Especially when visualizing numerical data from high performance computing (HPC) simulations, there are often multiple data components which can be mapped onto the surfaces, e. g. by a color map. The resulting colored surfaces can be still very large. Moreover, they are often produced by a marching cubes type algorithm leading to a large number of very small triangles. Thus, the simplification of those surfaces is an important task to make interactive visualization of scientific data practicable.

Existing simplification algorithms employ geometric criteria to determine the order of the reduction process. Their goal is to obtain a coarse representation of the surface in areas of low curvature, and a finer representation in more wrinkled parts of the surface. But important features in the data components may be situated in areas of rather flat shape, e. g. turbulences in flow fields mapped on a cutting plane. This shows that in the case of data attached to the surface purely geometric criteria are not sufficient to get an appropriate representation containing less triangles without loosing essential information.

We developed a surface simplification algorithm involving data-dependent criteria, namely, a gradient approximation of the data attached to the surface.

As surfaces may be of a simple shape (think of a cutting plane or sphere) geometry related criteria may seem to the user less dominant than data-dependent criteria. Therefore, our priority criterion is a linear combination of a curvature and a gradient approximation, weighted by a user-defined parameter. The gradient approximation scheme allows both scalar and vector data to be taken into account.

We decided to stay with topology preserving methods, although the algorithm can be easily adopted to allow topology changes. But, topology changes seem to be more important in the visualization of geometric models, where parts of the scene can be at such a distance from the observer that holes or cracks get smaller than pixel size. In scientific visualization, this seems to be a less urgent issue.

Obviously, the more criteria are used to control the simplification process, the less reduction can be achieved. However, the algorithm is still able to reduce large surfaces by a factor up to 10 without blurring important features in the data. Since it has a time complexity of $O(N \log N)$, with N being the number of original vertices, it works sufficiently fast to be used interactively even in a virtual reality environment.

1.1 Related work

We do not pretend to give a complete survey on surface simplification algorithms. In the last years, this topic has been developing rather fast. For a historical overview and a vast bibliography, see the paper of A. Guéziec [10].

Our algorithm belongs to the class of geometry removal methods. Direct inspirations came mostly from papers by H. Hoppe et al. [13], [15], W. Schroeder et al. [17], B. Hamann [12], A. Guéziec [10], R. Klein et al. [14], and M. Garland and P. Heckbert [8]. In these articles, different removal strategies (vertex removal, edge collapsing, triangle removal) are proposed involving various kinds of geometric priority criteria.

Data dependencies are common in the well-developed field of data dependent triangulation (see e. g. [7],[2]). There the data is usually given as function values in vertices on the 2D plane, thus creating a height field. Unfortunately, the heuristics given in these papers are not applicable directly.

In volume data visualization, there are techniques to compress data given on a three-dimensional grid by representing areas where the data are not changing essentially in a more efficient way, e. g. using a finite element approximation [9] or other basis representations (see e. g. [22]). Since we do not have access to the data on the original grid, but only to a subset of the data defined on a 2-manifold in $\mathbb{R}^3$, these methods are not applicable, either.

2 The algorithm

We use a typical geometry removal approach, deleting one geometry element at a time. The general outline of the algorithm is as follows:

```
ALGORITHM:
    Preprocessing operations (retrieving connectivity information)
    Construct priority queue
    while (priority queue is not empty)
    {   Take next geometry element
        Perform topology-preserving tests
        if (element can be removed)
        {   Collapse geometry element
            Update data structures and priority queue
        }
    }
```

In the following, the main steps of the algorithm will be explained in more detail.

2.1 Geometry removal

There are basically three possibilities to remove geometry elements:

- Vertex removal
- Edge collapsing
- Triangle removal

When removing a single vertex, one has to re-triangulate the resulting hole in the surface. This operation can be avoided by using the edge collapsing approach instead. Here, the two points adjacent to the edge are merged into one (see Fig.1), and the surrounding triangles merely change their shape. The simplest and fastest way to compute the new point the edge collapses in is to take the midpoint of the edge. But for convexely shaped surfaces, this leads to flattening of the surface which is sometimes not desirable. Therefore, A. Guéziec [10] developed a method of calculating the new point according to a volume preserving strategy. However, for flat surfaces like cutting planes this additional effort is unnecessary.

If the preferred generic operation is triangle removal, again the hole in the surface resulting from removing all adjacent faces must be handled. This can be done in two ways. Firstly, it can be re-triangulated, which results in comparatively large, possibly flattened spots. On the other hand, a midpoint can be chosen according to some rule, avoiding re-triangulation. For instance, B. Hamann [12] approximates the surface locally by a bi-quadratic function and computes the midpoint as the value of the approximating function in the origin of the local coordinate system. Besides being numerically sensitive, this procedure is quite expensive. But on well-shaped grids which do not contain very small triangles it leads to surfaces of a high quality.

We implemented and compared all three approaches with respect to time and memory efficiency. As a result, we prefer the vertex removal and edge collapsing strategies to triangle removal because they allow a finer tuning of the iterative process, causing less deformation of the surface in each iteration step. Hence in the present paper, we decided not to report on the triangle removal algorithm.

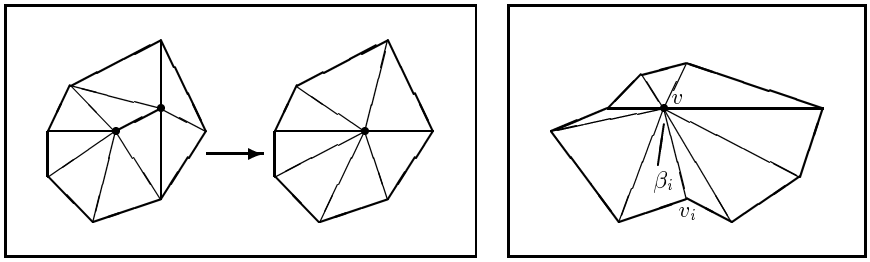


Figure1. The edge collapsing operation (left). Total angle $\Theta(v) = \sum_{i=1}^{n} \beta_i$ (right).

2.2 The data-dependent priority criterion

The heart of any geometry removal algorithm is a priority queue of geometry elements (vertices, edges, or triangles) providing the order in which they have to be removed. In existing surface simplification algorithms, the priority criterion is purely geometrical, e. g. based on a curvature estimation using an approximating function [12], [20], edge length [10], triangle area, distance to an average plane [17], or some energy function [15].

These geometric criteria are sufficient as long as no data is attached to the surface. But e. g. in the case of cutting surfaces through numerical data, which often are of a rather simple geometry, a purely geometric approach does not make sense. Therefore, we combine geometric criteria with data-dependent criteria approximating the gradient of the data.

For the sake of efficiency, we use a comparatively simple priority criterion. We combine a curvature criterion with a gradient approximation, thus assigning each vertex v a weight

$$\text{weight}(v) \;=\; \alpha * \text{curv}(v) \;+\; (1-\alpha) * \text{grad}(v)\,, \tag{1}$$

where $\alpha \in [0,1]$ is a user-defined scalar parameter.

If the generic operation in the simplification process is edge collapsing or triangle removal instead of vertex removal, we add up the weights of all vertices belonging to the edge (or the triangle, respectively) in order to get a weight for the geometry element to be processed.

Gradient estimation Usually, the gradient of a function $f : \mathbb{R}^3 \to \mathbb{R}$ is defined as

$$\text{grad}f \;=\; \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right).$$

Since our data function is defined on an 2-manifold in $\mathbb{R}^3$ rather than on $\mathbb{R}^3$, and is not differentiable everywhere, this gradient of the data function is not defined. Hence, we have to look for other heuristics to get a measure of the rapidity of changes in the data values in the neighborhood of a vertex.

Various gradient approximation schemes for scattered data given in points on a 2D plane are described in [19]. A different approach can be imagined using again a biquadratic approximation, this time of the 3D manifold in 4D space built by the surface-attached data. Such an approximation is described in [11]. Besides being expensive, the least-square optimization used to find the best fitting biquadratic function is numerically sensitive to very small triangles which are often produced by the marching cubes algorithm. Moreover, this approximation can be used for scalar data only.

Let f_i be the data values attached to the vertices v_i, $i = 1, .., k$, adjacent to $v \in T$, and f be the data value attached to v itself. We calculate

$$\text{grad}(v) \;=\; \max_{i=1,\ldots,n} \frac{\|f_i - f\|_1}{\|x_i - x\|_1} \,. \tag{2}$$

Here, x_i and x denote the three-dimensional coordinates of v_i, v, respectively, and $\|\cdot\|_1$ is the usual sum vector norm assigning each vector $y = (y_1, \ldots, y_k) \in \mathbb{R}^k$ a non-negative number

$$\|y\|_1 \;=\; \sum_{i=1}^{k} |y_i| \,.$$

Instead of the $\|\cdot\|_1$-norm any vector norm could be used. If the Euclidean norm is used to compute the distance between vertices, this heuristic can be interpreted as an estimate for the maximal value of discretized directional derivatives

$$\frac{\partial f(v)}{\partial (x_i - x)} \;=\; (\text{grad} f(v), x_i - x) \,,$$

taken over all directions $x_i - x$. We preferred to avoid the rather expensive square root calculation, having in mind that in $\mathbb{R}^3$ all norms are equivalent. Note that this gradient heuristic is independent of whether the data attached to the surface is scalar or vector valued.

Although the approximation of the gradient seems to be quite rough, experiments show that it leads to promising results. Moreover, it can hardly be surpassed in speed by other methods demanding additional computations. More sophisticated approximation schemes will be implemented and tested in the future, but we expect the gain in accuracy hardly to be worth the loss in efficiency.

Curvature estimation Triangulated surfaces are non-regular surfaces. Therefore, their curvature is not defined everywhere in the sense of differential geometry. To avoid this problem we use either simple heuristics or the rather expensive approximation of the triangulated surface by a smooth surface whose curvature can be calculated.

In the 50's and 60's, the Russian mathematician Aleksandrov [3] developed a theory of non-regular surfaces, in particular, of polyhedral surfaces. The polyhedral metrics they employed to define analogues of Gaussian and mean curvatures were used by Alboul and van Damme [2] for the reconstruction of surfaces from scattered data, recently. See their report [1] for a compact survey on the basic

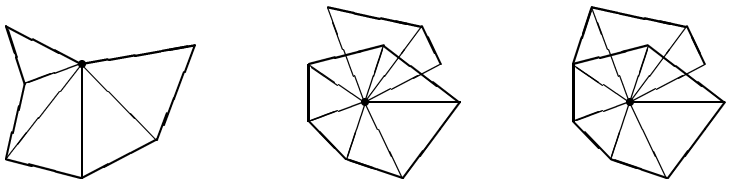


Figure2. Star of a boundary vertex (left) and a non-manifold vertex(middle and right)

ideas of Aleksandrov, and the monograph [4] for a detailed explanation. It turns out that the Gaussian curvature of a vertex v in a non-regular surface with polyhedral metric can be calculated very easily. Let β_i be the inner angle of the triangle (v_i, v, v_{i-1}) which is next to v (see Fig. 1). Then the total angle $\Theta(v)$ in v is defined as

$$\Theta(v) \;=\; \sum_{i=1}^{n} \beta_i \,.$$

If the point x is not a vertex of the triangulated surface, we set $\Theta(x) = 2\pi$. Then, for any point x on the surface the curvature is defined as

$$\mathrm{curv}(x) \;=\; 2\pi - \Theta(x) \,. \tag{3}$$

This expression is also known as the angle deficit. Note that for vertices only we can have a non-zero curvature. In boundary vertices, the curvature is defined as

$$\mathrm{curv}(x) \;=\; \pi - \Theta(x) \,. \tag{4}$$

Besides being mathematically sound, this geometric criterion can be calculated efficiently.

2.3 Topology preserving tests

All vertices are treated equally, assigning them a weight according to (1). But, when a vertex (an edge) is extracted from the priority queue, we perform some tests to preserve feature edges and the topology of the surface. Essentially, it suffices to look for two basic properties of the vertex to be removed, or the vertices adjacent to the edge to be removed, respectively:

- Does the vertex lie on a boundary or a feature edge?
- Is the manifold property satisfied?

Vertices lying on the boundary or on a feature are removed only in the case that the boundary (the feature edge) is continuing in the same direction. This is in some sense equivalent to Schroeders [17] distance to edge criterion, with the difference that for us it is merely an additional test, not the only criterion. Note that the boundary includes also the boundary of interior holes of the surface.

Vertices whose star, i. e. the set of all triangles sharing the vertex, does not have a closed boundary polygon, are assumed to be boundary vertices. If the boundary polygon consists of non-connected components or contains multiple edges outgoing from one vertex (see Fig. 2), the vertex is a complex one. In other words, the surface does not satisfy the 2-manifold property in the neighborhood of this vertex, locally. Complex vertices we do not remove, thus preserving splits of the surface.

3 Complexity of the algorithm

For the computation of the weights as well as for the topology preserving tests, we need the star of a vertex, i. e. the set of all triangles sharing that vertex. Therefore, we store the stars of all vertices in a special array, updating it after each iteration step. The computation of this is linear in the number of vertices in the surface. The priority queue is organized in a heap structure, hence its construction takes linear time as well. So we have a complexity of $O(N)$ for the preprocessing step, N being the number of vertices in the original surface.

Let us have a look at a single iteration step. Extracting the first element of a priority queue can be done in constant time. The topology preserving tests consist merely in sorting the edges of the boundary polygon of the star, hence they take $O(r^2)$ operations, r being the number of triangles in the star. The main effort in each iteration step is enclosed in the updating procedure. Here, we have to update the weights of all vertices belonging to the boundary polygon of the star, which leads to changes of their positions in the priority queue. Changing position, removing, or inserting an element in the priority queue takes $O(\log N^*)$ operations, N^* being the current length of the priority queue. Hence, each iteration step has a complexity of $O(r \log N^*)$. Usually, fans are not growing very much, so r is bounded by a constant number and can be neglected.

In the case of edge collapsing, the computation of the new vertex to be inserted takes a constant effort. When Guéziec's volume preservation strategy is applied, the effort grows up to $O(r)$, where the constant in the $O(\cdot)$-expression can be quite large. However, the re-triangulation required by the vertex removal strategy takes the most time of those three approaches. Although B. Chazelle has shown that an arbitrary polygon with r vertices can be triangulated in $O(r)$ operations [6], his optimal algorithm is of rather theoretical importance. There is a randomized triangulation algorithm by R. Seidel [18] requiring $O(r \log^* r)$ operations which is almost linear in practice. However, in our implementation we currently use a simpler triangulation algorithm [5] with an $O(r \log r)$ effort.

Usually, we have reduction rates of a factor 5–12. This means that we have to perform $O(N)$ removal operations. So the overall complexity of the algorithm can be estimated by $O(N \log N)$. However, this is probably slightly overestimated in practice, because the length of the priority queue is decreasing linearly with the number of iterations and can be next to nothing at the end of the iteration. Hence in practice, the algorithm behaves almost linear, as illustrates Fig. 3 below. These CPU timings were measured on an SGI R10000 (195 MHz).

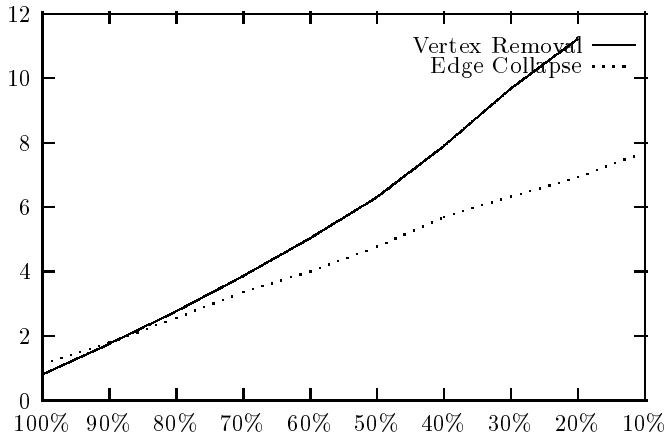


Figure3. CPU timings (in sec) required for the air flow isosurface with 50160 triangles

The example (Fig. 3–5) shows the air flow inside a car cabin. We simplified the isosurface corresponding to a temperature of 18.25°C, where the color represents the pressure inside the car cabin. Since the shape of the surface is not entirely simple, the user-defined parameter α was set to 0.5. The car surface itself was simplified by our geometric surface simplification algorithm. Both methods are integrated into COVISE, the collaborative visualization environment developed by our group (for more detail see [21], [16]).

4 Conclusions

In this paper, we propose a fast algorithm for the data-dependent simplification of arbitrary polygonal surfaces with vertex-attached, possibly multi-dimensional data. First experiments show that the $O(N \log N)$ complexity of the algorithm leads to a almost linear behavior in practice. The edge collapsing approach appears to be still substantially faster than the vertex removal approach due to the comparatively high effort of the re-triangulation procedure. In addition, the triangulation of the simplified surface resulting from the edge collapsing method is often more regular, as can be seen in Fig. 5 and 5. By improving the performance of the re-triangulation these disadvantages will be moderated in future. On the other hand, the vertex removal approach does not cause any interpolation problems for the vertex-attached data, as the edge collapsing does.

Although the proposed gradient criterion is quite rough, it is very efficient and seems to be a sufficiently exact heuristic to mark areas where the data function changes rapidly. The development and comparison of other data-dependent criteria will be subject to further research.

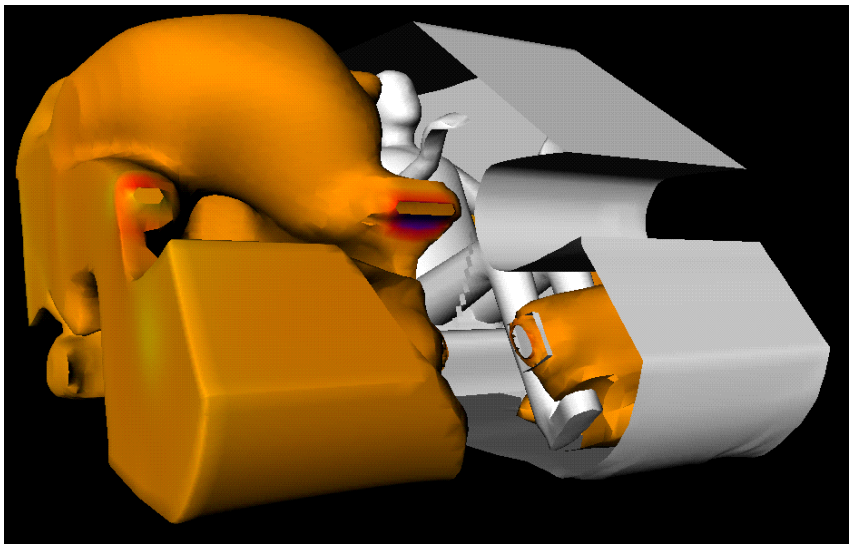


Figure4. The original isosurface (50160 triangles) and car cabin (29104 triangles). The data set is courtesy of Daimler Benz AG.

5 Acknowledgements

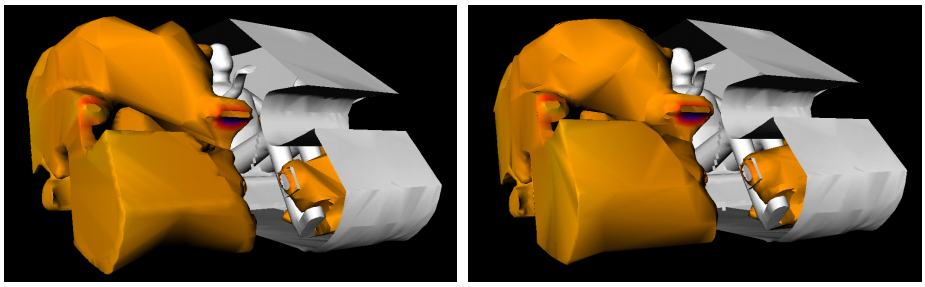
During the work on this subject the first author was supported by the European Community in the ESPRIT Project INDEX (EP 22745).

The car cabin air flow data set could be used by courtesy of Daimler Benz, a partner in the INDEX Project.

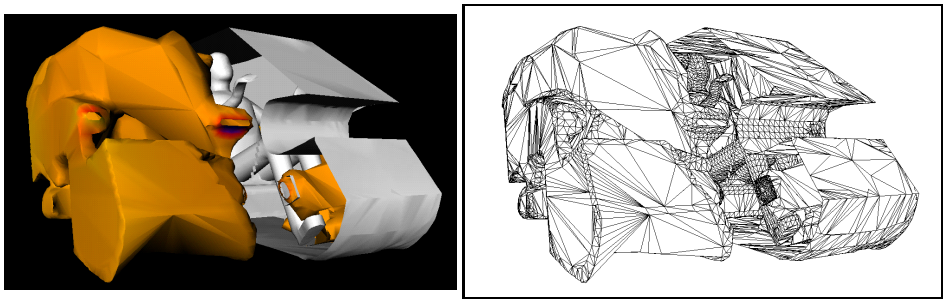
References

1. L. Alboul and R. van Damme. Polyhedral metrics in surface reconstruction: Tight triangulations. Memorandum No. 1275, Faculty of Applied Mathematics, University of Twente (NL), 1995.
2. L. Alboul and R. van Damme. Polyhedral metrics in surface reconstruction. In *Mathematics on Surfaces VI*, pages 171 – 200. Clarendon Press, 1996.
3. A.D. Aleksandrov. On a class of closed surfaces (in russian). *Mat. Sbornik*, 4:69 – 77, 1938.
4. A.D. Aleksandrov and V.A. Zalgaller. *Intrinsic Geometry of Surfaces*. AMS, 1967.
5. M.de Berg, M.van Krefeld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer, 1997.
6. B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485 – 524, 1991.
7. N. Dyn and S. Rippa. Data dependent triangulations for scattered data interpolation and finite element approximation. *Applied Numerical Mathematics*, 12:89 – 105, 1993.

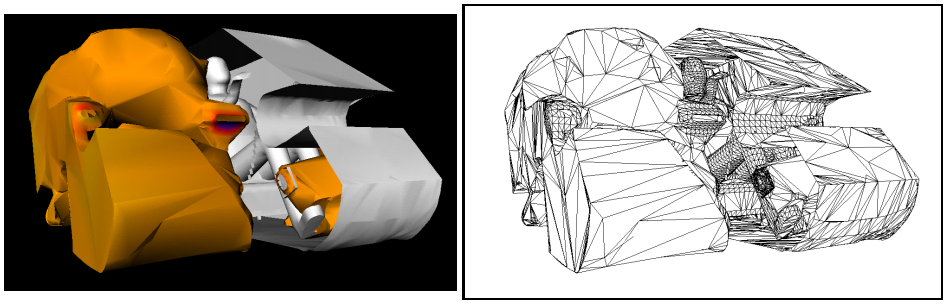
8. M. Garland and Heckbert P.S. Surface simplification using quadric error metrics. In *Computer Graphics*, 1997.
9. R. Grosso, Ch. Lürig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *IEEE Visualization '97*, pages 387 – 394. IEEE, 1997.
10. A. Gueziec. Surface simplification inside a tolerance volume. Technical Report RC 20440, IBM Research Report, 1997.
11. B. Hamann. Curvature approximation of 3d manifolds in 4d space. *CAGD*, 11:621 – 632, 1994.
12. B. Hamann. A data reduction scheme for triangulated surfaces. *CAGD*, 11:197 – 214, 1994.
13. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics (Proc. SIGGRAPH)*, 27(2):19 – 25, 1993.
14. R. Klein, G. Liebich, and W. Strasser. Mesh reduction with error control. In *IEEE Visualization '96*, pages 311 – 318. IEEE, 1996.
15. J. Popovic and H. Hoppe. Progressive simplicial complexes. *Computer Graphics (Proc. SIGGRAPH)*, 31:217 – 224, 1997.
16. D. Rantzau, U. Lang. A Scalable Virtual Environment for Large Scale Scientific Data Analysis. In: *Proceedings of the Euro VR Mini Conference 97*, Amsterdam, 10.-11. Nov. 1997, Elsevier 1998.
17. W. Schroeder, J. Zarge, and W.E. Lorensen. Decimation of triangle meshes. *Computer Graphics (Proc. SIGGRAPH)*, 26(2):65 – 70, 1992.
18. R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51 – 64, 1991.
19. S. Stead. Estimation of gradients from scattered data. *Rocky Mountain J. Math.*, 14(1):265 – 279, 1984.
20. G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *5th Intern. Conf. on Comp. Vision, Los Alamitos*. IEEE Comp. Society Press, 1995.
21. A. Wierse, U. Lang, R. Rühle. Architectures of Distributed Visualization Systems and their Enhancements. *Eurographics Workshop on Visualization in Scientific Computing*, Abingdon, 1993.
22. J. Wilhelms and A. Van Gelder. Multi-dimensional trees for controlled volume rendering and compression. In *1994 Symposium on Volume Visualization*. ACM SIGGRAPH, 1994.



Isosurface reduced by the Vertex removal (left) and Edge collapse (right) strategies to 25% (12540 triangles) (Frank et.al., Fig. 5).



Isosurface reduced by the Vertex removal strategy to 20% (10032 triangles) (Frank et.al., Fig. 6).



Isosurface reduced by the Edge collapse strategy to 15% (7523 triangles) (Frank et.al., Fig. 7).