Feature Preserved Volume Simplification

Wei Hong[1] and Arie Kaufman[1]
Center for Visual Computing (CVC)
and Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400

ABSTRACT

The goal of this work is to simplify volumetric datasets while preserving the sharp boundary features and the topology of the 3D scalar field. The simplification is performed on the volumetric datasets defined as a tetrahedral mesh. The sharp boundary features are detected and preserved in the simplification to conserve the salient details of the tetrahedral mesh. The topology of the 3D scalar field is preserved by maintaining critical points extracted from the original dataset. No critical vertices are removed or generated during the simplification process. A gradient magnitude based error metric is used to estimate the error associated with a simplification step. The simplified tetrahedral mesh is rendered using a hardware-accelerated unstructured volume rendering algorithm. This method produces results which are visually similar to the original dataset.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*surfaces and object representations*

General Terms

Algorithms

Keywords

Critical Point, Volume Simplification, Tetrahedral Mesh, Topology Preservation

1. INTRODUCTION

Manipulating and rendering highly complex volumetric datasets can be time and space prohibitive. This is particularly true for scientific datasets defined on irregular grids. Several volume rendering algorithms for irregular grids have been developed with graphics hardware acceleration [8, 13, 14], but still can't render very large dataset interactively.

[1]Email:{weihong,ari}@cs.sunysb.edu

The goal of a volume simplification algorithm is to meet the performance requirements while still maintaining the highest possible image quality. Volume simplification is different from surface simplification in which the shape impressions are extremely important. The volumetric datasets are visualized by direct volume rendering or isosurface extraction. Thus, volume simplification is driven by preserving primarily the interior structures. Geometric error metrics are not as important for volume simplification. Instead, data error metrics are of paramount importance in volume simplification.

This paper presents an algorithm in which the original, large dataset is simplified for faster visualization, while the boundary features and the topology of the 3D scalar field is well preserved. The topology of the 3D scalar field can help preserve the important iso-surfaces inside the volumetric dataset. The remainder of this paper is organized as follows. We review previous related work on polygonal mesh and tetrahedral mesh simplification in Section 2. Section 3 describes our methods to detect features from the original tetrahedral mesh. A feature preserved volume decimation algorithm is presented in Section 4. The implementation and results are reported in Section 5.

2. RELATED WORK

A wide variety of algorithms have been developed for the simplification of a polygonal mesh. Some polygonal simplification techniques have advanced to consider data features during the simplification process. Hamann et al. [9, 10] and Gieng et al. [6, 7] have developed algorithms that simplify triangle meshes by removing triangles which are ordered according to a weight based partially on the curvature of a surface approximation, on the topology of the mesh due to a triangle collapse, and on a predicted error of the collapse operation. Schroeder [15] has proposed an algorithm based on a fast local decimation to yield a guaranteed reduction level, modifying the topology as necessary to achieve the desired result. Bajaj et al. [1] have presented an approach to simplify the scalar fields over unstructured grids which preserves the topology of functions defined over the triangulation, in addition to bounding the errors. Our simplification algorithm is working on the volumetric dataset defined on the tetrahedral mesh.

Compared with polygonal mesh simplification, only a few attempts have been made towards polyhedral mesh simplification. Trotts et al. [17] have extended the method of Gieng et al. [7] to tetrahedral meshes. They use a single edge collapse as an atomic decimation operation. The predicted deviation is estimated from the original scalar field due to a tetrahedron collapse. However, they avoid many of the topological considerations of Gieng et al.'s work. Staadt and Gross [16] have discussed some fundamental issues for robust implementations of progressively refined tetrahedralizations generated through sequences of edge collapses. Besides preserving the topological and geometrical features such as a boundary, the algorithm elegantly handles the tetrahedron boundary intersections at concave interiors. Cignoni et al. [3, 4] have integrated error evaluation for both domain and field approximations to emphasize accurate error evaluation. Local error accumulation, gradient difference, and brute force strategies are explored to predict and evaluate errors while incrementally simplifying a tetrahedral mesh. Chopra and Meyer [2] have introduced an algorithm to take all of the four vertices of a tetrahedron, and fuse them onto the barycenter of the tetrahedron, while still taking care of complex mesh-inconsistency problems. Van Gelder et al. [18] have presented a method for rapidly decimating volumetric data defined on a tetrahedral grid. They compared the mass-based and density-based decimation error metrics, and the mass-based metric is found to be superior.

Our work is based on the basic vertex decimation method. We use a gradient magnitude based error metric to preserve the important surfaces in the volumetric datasets, which is different from Van Gelder et al.'s data-based error metric. Moreover, sharp boundary features are detected in a preprocessing step and preserved during the vertex decimation to preserve the boundary of the tetrahedral mesh. Bajaj et al. [1] only detect critical points on the surface and preserve the topology of a 2D scalar field, while we detect critical vertices in the original volumetric dataset defined on the tetrahedral mesh and preserve them in the simplified tetrahedral mesh in order to preserve the topology of the 3D scalar field.

3. FEATURE DETECTION

It is important to preserve the details present in the original data even after simplification. In order to preserve the original topology of the 3D scalar field defined on the tetrahedral mesh and the boundary of the tetrahedral mesh, we detect critical vertices and sharp boundary feature vertices in a preprocessing step.

3.1 Critical Points

The topology of the scalar field is characterized by its critical points. A point x is a critical point on the C^2 scalar function $f(x)$ if all first order partial derivatives of f evaluated at x are zero, i.e., $\nabla f = 0$. Gerstner and Pajarola [5] have considered piecewise linear interpolation applied to tetrahedral grids, which leads to C^0 continuous functions, and critical points can only occur at mesh vertices. They classify vertices quite efficiently based on look-up tables.

Connected components in an edge graph of a surrounding polyhedron correspond to connected components in a neighborhood of a vertex. From this observation, Weber et al. [19] have given a definition for *regular and critical* points based on the number of connected components in their local neighborhood. Assume the number of "positive" connected components surrounding point x is n_p, and the number of "negative" connected components surrounding point x is n_n. If $n_p = n_n = 1$, the point x is a regular point. If $n_p = 1$ and $n_n = 0$, point x is a minimum. If $n_n = 1$ and $n_p = 0$, point x is a maximum. If $n_n + n_p > 2$, point x is a saddle.

If two vertices connected by an edge have the same function value, the entire edge can represent an extremum or a saddle. It is even possible that a piecewise curve or a region is critical. To avoid these types of problems, both of the above papers impose a restriction on the data that function values at vertices connected by an edge must differ. However, in real volumetric datasets, many regions are homogeneous. Thus, we detect these critical regions in our algorithm, and we show that this can make our volume simplification algorithm even more efficient. We start by region grow to extract all connected regions. A new tetrahedral mesh is obtained by contracting each region into one vertex at its geometric center, and modifying the corresponding connectivity information. We use the above definition to classify the vertices in the new mesh, because the function values at the vertices connected by an edge are different now. However, the look-up table can't be employed in the new tetrahedral mesh, because the surrounding polyhedra are not regular any more. Furthermore, we classify the vertices in the critical region as critical boundary vertices and critical interior vertices. The critical boundary vertices should be preserved as well as the isolated critical vertices. The critical interior vertices can be simplified as general interior vertices, and the associated error is zero.

3.2 Sharp Boundary Features

In Van Gelder et al.'s algorithm [18], a surface vertex can be merged only with another surface vertex. They use a weighted combination of the data-based error and a geometry error to evaluate the deviation. Merging a surface vertex into an adjacent surface vertex may still change the shape of the surface of the tetrahedral mesh. We use an efficient algorithm to classify the boundary vertices. The boundary vertices are identified as corner vertex, curve vertex, and general boundary vertex. The classification algorithm is based on the solid angle pre-computed at each vertex.

The solid angle θ_0 of tetrahedron $T(P_0, P_1, P_2, P_3)$ at vertex P_0 is defined to be the surface area formed by projecting each point on the face not containing P_0 to a unit sphere centered at P_0. An equation for the computation of solid angle θ_0 is given by Liu et al.[12]:

$$\sin\frac{\theta_0}{2} = \frac{12v}{\sqrt{\prod\limits_{1\le i<j\le 3}[(l_{i0}+l_{j0})^2 - l_{ij}^2]}} \tag{1}$$

where v is the volume of T, and l_{ij} is the length of the edge connecting vertices P_i and P_j. The solid angle θ at a vertex v is the sum of the solid angles in all tetrahedra incident at v. For an interior vertex, the solid angle is 4π, while the boundary vertex is less than 4π. Suppose the solid angle at boundary vertex v is θ, we identify the type of v as follows:

1. v is a *corner* boundary vertex if $\theta \le \frac{\pi}{2}$ or $4\pi - \theta \le \frac{\pi}{2}$.
2. v is a *curve* boundary vertex if $\frac{\pi}{2} < \theta \le \frac{3\pi}{2}$ or $\frac{\pi}{2} < 4\pi - \theta \le \frac{3\pi}{2}$.
3. Otherwise, v is a *general* boundary vertex.

The sharp feature vertices of Spx dataset are shown in Figure 1(a) in yellow. All these feature boundary vertices are preserved in the simplified data to preserve the shape of the tetrahedral mesh. Only the general boundary vertices are subject to simplification.

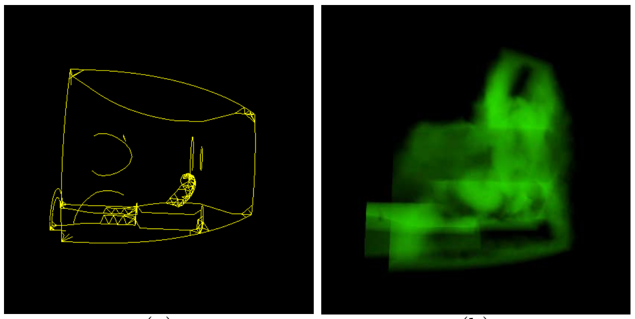
(a) (b)

Figure 1: (a) The sharp feature vertices of Spx dataset; (b) The gradient magnitude of Spx dataset is rendered using direct volume rendering.

4. FEATURE PRESERVED SIMPLIFICATION

The basic vertex decimation algorithm takes a tetrahedral mesh as input. It computes the error for all vertices, and inserts them into a priority queue. Then, the algorithm iteratively selects a vertex based on its error and deletes it by merging it with its destination vertex. When a vertex v_i is merged with its neighboring vertex v_j, it drags along with it all the edges that are incident on it. The destination vertex v_j is chosen from the neighbors of the vertex v_i such that the merging causes minimum change in the region surrounding the vertex v_i. It can be ensured that the merging operation does not produce intersecting tetrahedra by comparing the signs of the volumes of the original and stretched tetrahedra (Intersection Check). If merging v_i with v_j leads to a change in sign of the volume of the affected tetrahedra, then this operation is invalid. After the merging operation, the error of the neighboring vertices of vertex v_i are recomputed, and the priority queue is updated accordingly. We modify this basic decimation algorithm to preserve the sharp boundary feature vertices and critical vertices detected from the original dataset.

4.1 Gradient Magnitude Based Error Metrics

Van Gelder et al. [18] have already demonstrated that the mass-metric is superior to the density-metric. The mass-metric is based on the changes in *mass* (the integral of *density* over a volume) of the material during decimation. Unlike their method, we use gradient magnitude to modulate the density to define a gradient magnitude based error metric. The gradient magnitude is a very powerful feature for enhancing material boundaries by suppressing data which resides within homogeneous areas of a dataset [11]. To preserve the material boundaries in the simplified dataset, we use gradient magnitude to modulate the density value. For structured grids, the gradient can be estimated easily by using the Sobel operator. For unstructured grids, Cignoni et al. [3] has proposed a method to estimate the gradient at each vertex v by computing the weighted average of gradients in all tetrahedra incident at v. The weight to be associated with the contribution of each tetrahedron T is given by the solid angle of T at v. We assume that the density values inside each tetrahedron vary linearly, which can be defined as $f(x, y, z) = Ax + By + Cz + D$. This equation can be solved by using the density value at the four corners of each tetrahedron. Then, the gradient inside the tetrahedron is $\nabla f = (A, B, C)$. Figure 1(b) shows the direct volume rendering of the gradient magnitude of Spx dataset.

Now we can compute the error ϵ_{ij} associated with the process of merging v_i into v_j as follows:

$$F(v_0, v_1, v_2, v_3) = \tfrac{1}{4}\textstyle\sum_{k=0}^{3} M(v_k) \cdot D(v_k) \quad (2)$$

$$G(v_0, v_1, v_2, v_3) = F(v_0, v_1, v_2, v_3) \cdot V(v_0, v_1, v_2, v_3) \quad (3)$$

$$\epsilon_{ij} = \sum_{t_k \in T_i} (G(v_i, v_{k1}, v_{k2}, v_{k3}) - G(v_j, v_{k1}, v_{k2}, v_{k3})) \quad (4)$$

where $M(v)$ is the gradient magnitude at vertex v, $D(v)$ is the density value at vertex v, $V(v_0, v_1, v_2, v_3)$ is the volume of the tetrahedron formed by v_0, v_1, v_2, v_3, T_i is the set of tetrahedra incident on v_i, and v_{k1}, v_{k2}, v_{k3} are the vertices other than v_i of tetrahedron t_k. The volume of the tetrahedron should be greater than zero, otherwise v_i can't be merged to v_j.

4.2 Interior Vertex Decimation

For the interior vertex decimation, the main goal of our algorithm is to preserve the isolated critical points and critical regions. Simply preserving all these critical vertices in the simplified tetrahedral mesh can't guarantee that these vertices still are critical in the simplified tetrahedral mesh. We need to check each decimation operation to guarantee that the original critical vertices are preserved. New critical minimum and maximum vertices are not generated during the vertex decimation. Unfortunately, new saddle vertices may be generated during the merging operation. Thus, we need to perform two checks when computing the error metrics: *criticality preserving* (CP) check and *no criticality generated* (NCG) check.

- CP check: When a vertex v is merged with its destination vertex, if there are some critical vertices among its neighboring vertices, we need to check whether these critical vertices are still critical.

- NCG check: When a vertex v is merged with its destination vertex, we should check all its non-critical neighboring vertices to guarantee that there is no critical vertex being generated.

If an interior vertex v is an isolated critical vertex or a critical boundary vertex, v is not inserted into the priority queue. Thus, all these vertices will be preserved in the simplified tetrahedral mesh. If an interior vertex v is a critical interior vertex, it can be safely decimated without any check. Because a critical interior vertex can only be merged with another critical interior vertex or critical boundary vertex, neither of them can change the criticality around vertex v.

For a general interior vertex v, after v is merged with its destination vertex, we need to check if any new critical vertex is generated in its neighborhood. If there are critical vertices in its neighborhood, we also need to check whether the criticality is damaged. If all these check are passed, this merging operation is valid.

4.3 Boundary Vertex Decimation

All corner boundary vertices and curve boundary vertices are directly preserved by not inserting them into the priority queue. A general boundary vertex can only be merged with another boundary vertex. For a general boundary vertex, a geometry error can be combined with the field error to guide the decimation [18]. When merging a general boundary vertex v_i to another boundary vertex v_j, the original n triangles t_i are replaced by $n-2$ new triangles t'_i. The two triangles sharing the edge (v_i, v_j) are deleted. The geometric difference between the old triangles and the new triangles can be measured using the magnitude of the vector difference between the average surface normals:

$$\epsilon_{gi} = \frac{1}{A_{total}} \| \sum_{i=1}^{n} N_{t_i} \cdot A_{t_i} - \sum_{i=1}^{n-2} N_{t'_i} \cdot A_{t'_i} \| \tag{5}$$

where N_{t_i} is the surface normal of the old triangle t_i, and $N_{t'_i}$ is the surface normal of the new triangle t'_i. The final error ϵ_{v_i} is a weighted combination of the geometry error and data error.

4.4 Simplification Algorithm

Our simplification algorithm detects all critical vertices and sharp feature boundary vertices first. All these feature vertices are preserved in the simplified tetrahedral mesh, and their properties should be preserved, i.e., critical vertices should still be critical vertices of the simplified tetrahedral mesh. Moreover, no new critical vertex can be generated during the simplification process.

Following is our simplification algorithm:

1. Compute error and find destination vertex for each vertex v, and put them into the priority queue:
   (a) If v is a critical interior vertex, find its destination vertex and check the intersection.
   (b) If v is a general interior vertex, perform intersection check, CP check, and NCG check when computing the error and finding the destination vertex.
   (c) If v is a general boundary vertex, compute the weighted geometry-based and data-based errors. All above checks are done during the computation.
   (d) Otherwise, just go to the next vertex.
2. Choose an error threshold ϵ, and while there exist vertices with error less than ϵ:
   (a) Select and remove from the priority queue vertex v with the minimum error.
   (b) Replace vertex v with its destination vertex in all tetrahedra which are incident on vertex v.
   (c) Recompute the error of neighboring vertices of v in the priority queue.

5. IMPLEMENTATION AND RESULTS

Our algorithm is implemented on a Windows platform PC with 2.26GHz Intel Pentium 4 CPU with an Nvidia Geforce4 Ti4400 with 128M RAM. The resulting images are rendered using 3D texture based direct volume rendering algorithm [13, 14].

The four datasets we used are listed in Table 1, ordered by the number of tetrahedra. While the first Spx dataset is an unstructured grid, the Plane and Car datasets are converted into tetrahedral meshes from regular grids, and the Blunt Fin is converted into a tetrahedral mesh from a curvilinear grid. Our simplification algorithm is very efficient. The time spent in the simplification process for the four datasets is listed in Table 2. For the Spx dataset, the hardware-based volume rendering can obtain real-time rendering frame rates, while for the other datasets it is interactive (see Table 3).

Table 1: Characteristics of our test datasets.

Dataset	Vertices	Tetrahedra	Facets	Boundary
Spx	2,896	12,936	27,252	2,760
Blunt	40,921	187,318	381,355	13,438
Plane	93,389	394,480	817,859	57,798
Car	132,124	597,092	1,221,268	50,928

Table 2: Running time of the simplification.

Dataset	50% Decimated(Sec)	80% Decimated(Sec)
Spx	1.8	2.9
Blunt	21.1	33.3
Plane	44.3	71.9
Car	60.5	98.4

Spx dataset provides concave boundary and curved sharp boundary features to verify the sharp boundary features preserved simplification. Figure 2 shows the comparison of sharp boundary features preserved and non-preserved simplification. From the figure, we can observe that when the sharp boundary features are preserved in the simplification, the boundary of the tetrahedral mesh is preserved well. Although the boundary of the tetrahedral mesh can be preserved by not decimating the boundary vertices, many boundary tetrahedra with very small minimum solid angle are generated, which are not desired.

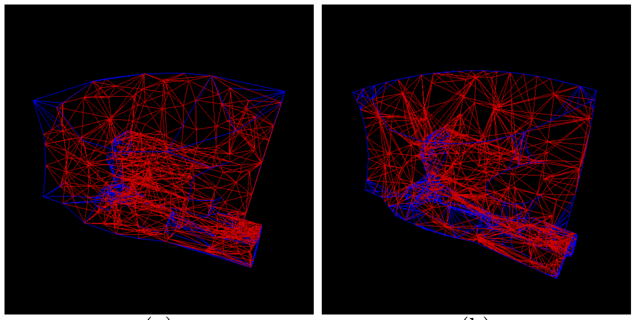

Figure 2: The 80% decimated Spx dataset (a) without preserving sharp boundary features; (b) with sharp boundary features preserved.

Figures 3-6 show images of volume rendering of the four datasets before and after our simplification. Each figure is rendered using the same transfer function from the same view point. The rendering frame rates are shown in Table 3. The rates include visibility sorting, decomposing tetrahedra into triangles, and rendering triangles which are all view dependent. The three-dimensional ray integrals are computed in a preprocessing step, and stored in a $64 \times 64 \times 64$ 3D texture map, which is updated when the transfer function is modified.

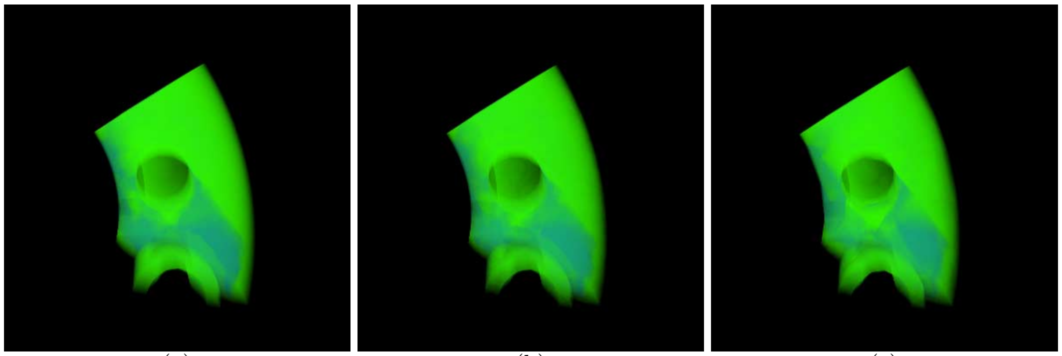
(a) (b) (c)

Figure 3: Volume rendering of the Spx dataset: (a) the original dataset with 2,896 vertices and 12,936 tetrahedra; (b) the 50% decimated dataset; (c) the 80% decimated dataset.

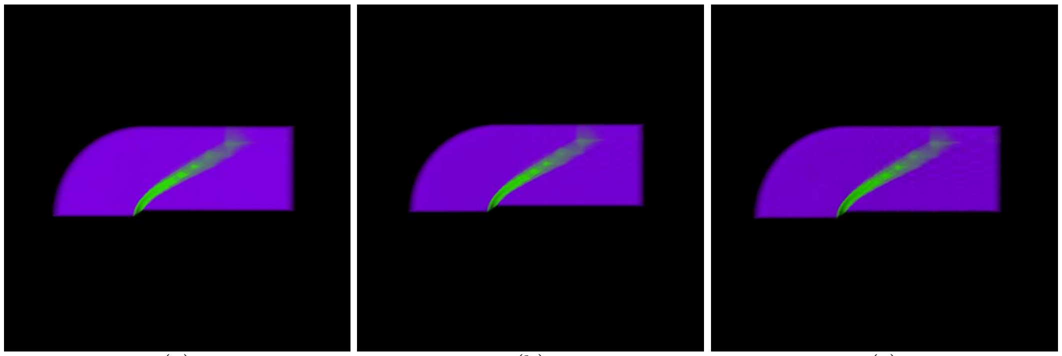
(a) (b) (c)

Figure 4: Volume rendering of the Blunt Fin dataset: (a) the original dataset with 40,921 vertices and 187,318 tetrahedra; (b) the 50% decimated dataset; (c) the 80% decimated dataset.

Table 3: Frame rates for rendering our datasets.

Dataset	Original	50% Decimated	80% Decimated
Spx	37.6	91.7	158.7
Blunt	2.8	5.4	13.3
Plane	1.3	2.3	6.2
Car	0.8	1.4	3.8

6. CONCLUSIONS AND FUTURE WORK

We have presented a feature preserved volume simplification algorithm. The volumetric dataset is defined as a tetrahedral mesh. The boundary of the tetrahedral mesh is well preserved by detecting sharp boundary feature vertices and preserve them in the simplification. The topology of the 3D scalar field is preserved in all simplified datasets by detecting critical vertices in the original dataset and preserve them during simplification. Our method guarantees that no new critical vertices are generated during the simplification. Thus, all the simplified datasets have the same topology in 3D scalar field as that in the original dataset. A gradient magnitude based error metric is used to enhance the material boundaries in the volumetric datasets.

In the future, we would take element shape measures into account as error metrics to generate high quality tetrahedral meshes. We also wish to exploit choosing the metrics based on a view-dependent factor. We treat all critical points in the same way now, but some of them may be discarded without loss. The "degree of criticality" of critical points will be considered in our future work.

Acknowledgement

The Spx dataset is courtesy of Lawrence Livermore National Laboratory. The Blunt dataset is courtesy of NASA Ames Research Center. The Car and Plane datasets are courtesy of CVC at Stony Brook University. This work has been supported by a grant from ONR N000130110034.

7. REFERENCES

[1] C. L. Bajaj and D. R. Schikore. Topology preserving data simplification with error bounds. *Computers & Graphics*, 22(1):3–12, Feb. 1998.

[2] P. Chopra and J. Meyer. Tetfusion: An algorithm for rapid tetrahedral mesh simplification. In *Proceedings Visualization 2002*, pages 133–140, Oct. 2002.

[3] P. Cignoni, D. Constanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. In *Proceedings Visualization 2000*, pages 85–92, Oct. 2000.

[4] P. Cignoni, E. Puppo, and R. Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, Oct. 1997.

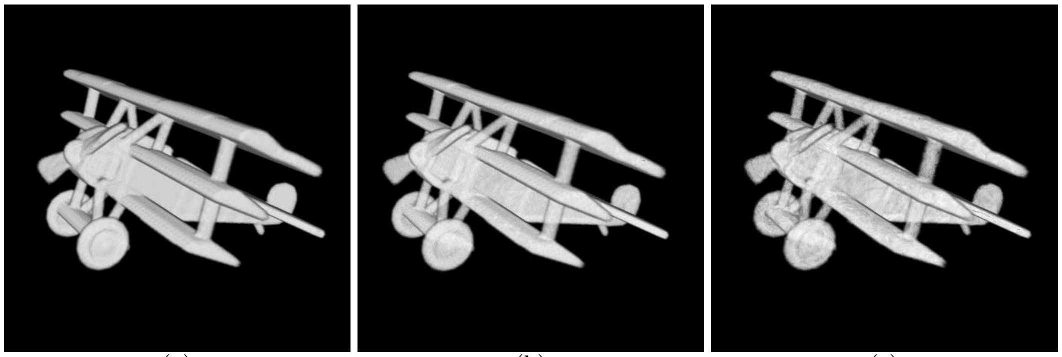
(a) (b) (c)

Figure 5: Volume rendering of the Plane dataset: (a) the original dataset with 93,389 vertices and 394,480 tetrahedra; (b) the 50% decimated dataset; (c) the 80% decimated dataset.

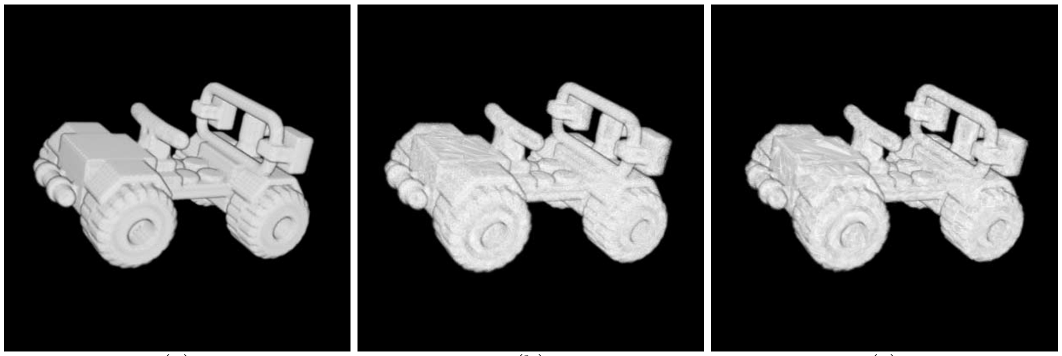
(a) (b) (c)

Figure 6: Volume rendering of the Car dataset: (a) the original dataset with 132,124 vertices and 597,902 tetrahedra; (b) the 50% decimated dataset; (c) the 80% decimated dataset.

[5] T. Gerstner and R. Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *Proceedings Visualization 2000*, pages 259–266, Oct. 2000.

[6] T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schlussmann, and I. J. Trotts. Smooth hierarchical surface triangulations. In *IEEE Visualization '97*, pages 379–386, Oct. 1997.

[7] T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schussman, and I. J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, Apr. 1998.

[8] S. Guthe, S. Röttger, A. Schieber, W. Straßer, and T. Ertl. High-quality unstructured volume rendering on the PC platform. In *Proceedings of the 17th Eurographics*, pages 119–126, Sept. 2002.

[9] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

[10] B. Hamann and J. Chen. Data point selection for piecewise linear curve approximation. *Computer Aided Geometric Design*, 11(3):289–301, 1994.

[11] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[12] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *BIT*, 34(2):268–287, 1994.

[13] S. Roettger and T. Ertl. A two-step approach for interactive pre-integrated volume rendering of unstructured grids. In *Proceedings Volume Visualization 2002*, pages 23–28, Oct. 2002.

[14] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings Visualization 2000*, pages 109–116, Oct. 2000.

[15] W. J. Schroeder. A topology modifying progressive decimation algorithm. In *IEEE Visualization '97*, pages 205–212, Oct. 1997.

[16] O. G. Staadt and M. H. Gross. Progressive tetrahedralizations. In *Proceedings IEEE Visualization '98*, pages 397–402, Oct. 1998.

[17] I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of tetrahedral meshes with error bounds. In *IEEE Transactions on Visualization and Computer Graphics*, volume 5 (3), pages 224–237. 1999.

[18] A. Van Gelder, V. Verma, and J. Wilhelms. Volume decimation of irregular tetrahedral grids. In *Proceedings of the Conference on Computer Graphics International 1999*, pages 222–230, June 1999.

[19] G. H. Weber, G. Scheuermann, H. Gagen, and B. Hamann. Exploring scalar fields using critical isovalues. In *Proceedings Visualization 2002*, pages 171–178, Oct. 2002.