Volume Decimation of Irregular Tetrahedral Grids

Allen Van Gelder, Vivek Verma, and Jane Wilhelms
Computer Science Department, 225 Applied Sciences Building
University of California Santa Cruz, Santa Cruz, CA 95064
Email: avg@cs.ucsc.edu, vivek@cs.ucsc.edu, wilhelms@cs.ucsc.edu

Abstract

Rendering highly complex models can be time and space prohibitive, and decimation is an important tool in providing simplifications. A decimated model may replace the original entirely or provide level-of-detail approximations. Here we present and evaluate, quantitatively and qualitatively, methods for rapidly decimating volumetric data defined on a tetrahedral grid. Results are compared using both direct volume rendering and isosurface rendering. A mass-based and a density-based decimation error metric are compared, and the mass-based metric is found to be superior. Grid surface vertices are decimated using a geometric error metric, as well as one of the data-based error metrics. Images produced using direct volume rendering and isosurface extraction on grids that are decimated approximately 80% are nearly indistinguishable from similar images using the non-decimated grids, and even at 95% decimation, the rendered images have few artifacts. Rendering speed-up depends upon the renderer used.

Keywords: *Computer Graphics, Scientific Visualization, Direct Volume Rendering, Decimation, Level-of-detail, Irregular Grids.*

1. Introduction

Although rendering algorithms and computer graphics hardware are rapidly improving our ability to visualize complex data, the increase in size and complexity of that data often outdistances those improvements. This is particularly true for scientific data defined over three dimensions, but is also an issue for computer graphical models, where volumetric models offer some advantages over more standard surface ones [9]. *Decimation* addresses the space and speed problems caused by data complexity by reducing the number of vertices. In some cases, decimation may produce a smaller data set with identical properties as the original, e.g., if a data set contains large regions of constant value. More often, decimation produces a simplified approximation using an error metric that compares the original to the decimated data. A series of approximations using different error metrics provide a level-of-detail (multi-resolution) hierarchy of data sets from which the user can select based on speed versus error considerations.

If the scientific data lies on a regular grid, or the volumetric model is defined in a regular pattern, the primitives in the decimated volume can be made to duplicate that simple pattern. Size and speed improvements can easily be seen in such cases. However, volumetric data is very often defined on irregular grids, and simply removing vertices results in increasingly complex polyhedral grid cells. Tetrahedral grids somewhat simplify this problem, because any complex polyhedral grid can be tetrahedralized, and many rendering algorithms exist for tetrahedral constructions.

Decimation of volumes has been less studied than surface simplification and the requirements are somewhat different. Usually, surface simplification is concerned mainly with the visual impression: if the simplified object "looks realistic" it is not important exactly how accurately it represents the original; shape impressions are of paramount importance. In scientific data, it is important that the simplified data convey the same scientific information as the original. One or more functions are numerically specified at the grid points. The goal is to reduce the number of vertices while maintaining an accurate representation of the functions. Thus, volume decimation is more akin to height-field decimation than to general surface decimation.

This paper presents a new technique in which the original, large data set is rapidly decimated for faster application of visualization techniques, such as direct volume rendering, isosurface extraction, stream lines, etc. Since large data sets can be hundreds of megabytes, it is important that the method introduces a minimal amount of new data. We estimate the memory overhead to be in the range of 20%. Initial experiments indicate that volumes decimated to about 80% are hardly distinguishable from undecimated volumes in terms of images produced using direct volume rendering and isosurface extraction, and that 95% decimation still produces images with few artifacts.

Our decimation technique repeatedly deletes a vertex of minimum penalty, adjusts the structure, and recomputes penalties in the affected region. The selected vertex is decimated by merging it into the most appropriate adjacent vertex, "dragging" its edges and faces. The geometric structure is little changed: new edges or faces are not created; degenerated tetrahedra are deleted. Data-based penalties are based on changes in density or mass caused by decimation. (For convenience, we call any scalar function "density", but it might represent any quantity. Following physical intuition, the integral of "density" over any volume is called "mass".) Successful decimation of surface vertices require that a geometric error, as well as data-based errors, are considered. The effect of decimation on rendering time depends on the rendering method and type of renderer used.

We discuss, in order: background in surface and volume decimation (Section 2); a new model for decimating tetrahedral grids (Section 3); experimental results (Section 4); and conclusions and future work (Section 5).

2. Background and Related Work

Mesh simplification algorithms can broadly be classified into *surface simplification*, and *volume simplification*.

Surface simplification techniques take a polygon mesh and attempt to build a new model with fewer polygons, whose surface deviates as little as possible from the original [2, 5, 6, 8]. Some methods require the surface to be resampled, so that the new model does not include vertices from the old. Most methods also require re-meshing, so the new connections are not a subset of the old. Here we are interested mainly in volume simplification, where much less work has been done.

2.1. Volume Simplification

For volume rendering, geometry preservation may be less important, because the geometry is not readily perceived in the final image. However, we still wish to produce an image which is as close as possible to the original using fewer cells, and faster to render than images of undecimated volumes. Storage requirements can be extremely high for curvilinear and tetrahedral grids because the topology and/or geometry of the grid must be stored explicitly. Simplification algorithms should have minimum memory overhead to avoid accentuating the problem. Visualization of large curvilinear and tetrahedral grids is far from being interactive even with the most advanced graphics hardware.

The method of Cignoni *et al.* starts from an initial simplified tetrahedralization and iteratively adds new vertices until the desired precision is achieved [3]. Precision is defined as the maximum difference between the original value and the interpolated value, at original vertices. This heuristic may not measure the error accurately.

More recently, Cignoni *et al.* [4] have developed algorithms to simplify curvilinear grids by first converting the hexahedral cells into tetrahedra. For irregular tetrahedral grids the same authors propose to decimate vertices in the physical space by iteratively deleting them and patching the resulting holes using local triangulations, but no implementation has been reported.

Grosso *et al.* [7] use an L_2 norm to estimate the error between the original and the approximate mesh. The error metric is similar to one we have proposed (Equation 3). Their method starts with a coarse mesh and constructs successively better approximations. A disadvantage is that vertices of the original mesh need to be regularly spaced.

Trotts *et al.* [10] method for decimation of tetrahedral meshes is most similar to ours. Their algorithm computes a local error metric to assign priorities to tetrahedra. The error for each tetrahedron is the predicted deviation of a linear function over the vertices in the current mesh when that tetrahedron is collapsed. The tetrahedra are assigned priorities according to the error values computed for them. The simplification algorithm then iteratively chooses to collapse a tetrahedron until a user-specified error threshold is satisfied. After a tetrahedron is collapsed, the errors for each of its neighboring tetrahedra are recalculated and the priorities updated. They have presented results which compare images generated using the original mesh and decimated meshes, but have not discussed the savings in rendering time or the run time of their simplification algorithm. Our approach differs from theirs as follows: a different error metric (Section 3.1) using a different approximation to Equation 3; faster computation of error; the *mass-metric* which considers the size of tetrahedra; and our ability to use geometry to decimate surface vertices.

3 A New Model for Volume Decimation

A tetrahedral mesh is comprised of a set of vertices connected together such that the edges form a tetrahedralization of the vertices. There are data values associated with each vertex. For convenience, we refer to a scalar data value as "density", although it may represent any quantity in the application. A typical interpolation scheme assumes that the density values defined inside each tetrahedron vary linearly; this is a basic assumption made in this work.

3.1 Error Metrics

The vertices of the tetrahedral mesh can be divided into *surface vertices* and *interior vertices*. Surface vertices lie on the exterior of the grid volume (itself a grid of surface triangles). Interior vertices do not lie on the exterior. Decimation of the interior vertices only involves error metrics based on volume data. We present two data-based error metrics called *density-metric* and *mass-metric* (Section 3.1.2). Decimation of surface vertices involves both data-based error metrics and geometric error metrics (Section 3.1.3).

3.1.1 Decimation Basics

Vertices are connected to form a set of space-filling, non-intersecting tetrahedra. Density values are associated with vertices. The set of all the tetrahedra incident on an internal vertex v form a closed region R that encloses v and whose surface is a triangular mesh. These tetrahedra form a tetrahedralization of the region R. A new tetrahedralization of this region can be obtained when the vertex v is merged into one of its neighbors, dragging along with it all the edges incident on it. A 2-D equivalent is shown in Figure 1.

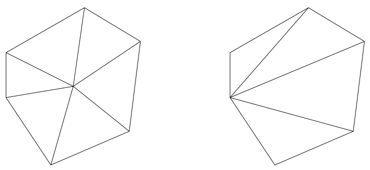

Figure 1. Retriangulation of a region.

Let us now make the following definitions (Figure 2):

$\{v_1, v_2, \ldots, v_n\}$:	the neighboring vertices of v.
$T_v = \{t_1, t_2, \ldots, t_m\}$:	the tetrahedra incident on v.
$D(v_i)$:	the density value at vertex v_i.
v_{j1}, v_{j2}, v_{j3} :	vertices other than v of tetrahedron t_j.
$\mathbf{a} \cdot \mathbf{b}, \mathbf{a} \times \mathbf{b}$:	vector dot product and cross product, respectively.

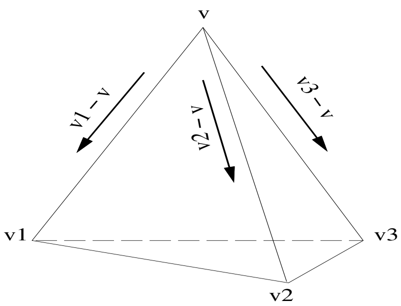

Figure 2. Computing the volume of a tetrahedron.

The volume V_j of tetrahedron t_j is then given by the triple scalar product:

$$V_j = \frac{1}{6}(v - v_{j1}) \cdot ((v_{j2} - v_{j1}) \times (v_{j3} - v_{j1})). \qquad (1)$$

The volume V_R of the region R can then be obtained by

$$V_R = \sum_{t_j \in T} V_j.$$

3.1.2 Data-Based Error Metrics

We will first describe the *density-metric*, based on the changes in density inside regions of the volume during decimation. The *mass-metric* is based on the changes in "mass" (the integral of "density" over a volume) during decimation.

The average density of tetrahedron t_j is given by the average of the density values at its corners. Hence, the mass of t_j is given by $\frac{1}{4}(D(v_{j1}) + D(v_{j2}) + D(v_{j3}) + D(v))V_j$, and the average density of region R is given by

$$\begin{aligned} \rho &= \frac{1}{V_R} \sum_{t_j \in T} \frac{V_j(D(v_{j1}) + D(v_{j2}) + D(v_{j3}) + D(v))}{4} \\ &= \frac{D(v)}{4} + \frac{1}{V_R} \sum_{t_j \in T} \frac{(D(v_{j1}) + D(v_{j2}) + D(v_{j3}))\, V_j}{4} \end{aligned}$$

We can now compute the error, ϵ_i, associated with the process of merging v into v_i, in terms of the change in average density of the material in region R, as follows:

$$\begin{aligned} \epsilon_i &= \frac{1}{4}(D(v_i) - D(v)) + \qquad (2) \\ & \frac{1}{4\,V_R} \sum_{t_j \in T} (D(v_{j1}) + D(v_{j2}) + D(v_{j3}))\, \Delta V_j \end{aligned}$$

where, ΔV_j is the change in volume of t_j (Equation 1).

A more rigorous definition of the error is given by the following expression:

$$\epsilon_i' = \sqrt{\frac{\int_R (D_{old}(p) - D_{new}(p))^2 \, dp}{V_R}} \tag{3}$$

In the above expression, the integral is a volume integral over the region R and $D_{old}(p)$ and $D_{new}(p)$ are the old and new density values, respectively, at a point p in the region R. Calculating ϵ_i' will be prohibitively expensive because it involves computing the intersections of the old tetrahedra with the new tetrahedra, therefore we settle for a less accurate value for the error. The quantity ϵ_i gives us an estimate of how the density changes, on average, in the region that has been re-tetrahedralized.

We also have to take care that, when we merge v into v_i, the new tetrahedralization is valid; i.e., no tetrahedra intersect each other. If the region R is convex, then we will always get a valid tetrahedralization. However, when R is non-convex, it is not always possible to obtain a valid tetrahedralization by merging v into one of its neighbors. Hence, we calculate ϵ_i only for those vertices v_i which result in a valid tetrahedralization when v is merged into v_i. To determine validity, we compute the volumes of the new tetrahedra. If they are all positive, the resulting tetrahedralization will be valid (we can do this because we are using the scalar triple product definition of the volume of a tetrahedron, see equation 1).

If the merger is valid, we compute ϵ_i using Equation 2, else we assign a value of infinity to ϵ_i. The decimation error for v is the minimum of ϵ_i over all v_i adjacent to v. The vertex chosen for the actual merge operation is the one for which the error is minimum and is called the *destination vertex*. There are cases where a new valid tetrahedralization is not possible, and for such cases v cannot be merged with any of its neighbors. When the error for a vertex is calculated, its destination vertex is stored in an array called the `destination_table`.

Using the *mass-metric*, the error associated with merging v into its neighbor v_i is calculated to be the change in the "mass" of the material in region R surrounding the vertex v. The equivalent of Equation 2 is as follows:

$$\begin{aligned}\epsilon_i \;&=\; \frac{1}{4}(D(v_i) - D(v))\, V_R + \\ &\quad \frac{1}{4}\sum_{t_j \in T} (D(v_{j1}) + D(v_{j2}) + D(v_{j3}))\, \Delta V_j\end{aligned} \tag{4}$$

The *mass-metric* can be viewed as the error calculated for a vertex v using the *density-metric*, which is weighted by the mass of the region R surrounding v.

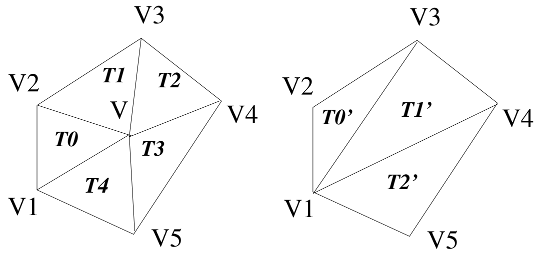

Figure 3. After decimating vertex V to vertex $V1$, we find the geometric differences between triangles $(T0', T0), (T0', T1), (T1', T2), (T2', T3), (T2', T4)$.

We can thus use the following algorithm to compute the error, E_v, of a vertex v:

1. For each neighboring vertex v_i of v compute ϵ_i using equation 2 for *density-metric* or equation 4 for *mass-metric*, (as described in the above paragraphs).

2. $E_v = \begin{cases} \infty & \text{if } v \text{ can't be merged to} \\ & \text{any neighbor,} \\ \min_i\{|\epsilon_i|\} & \text{otherwise} \end{cases}$

3. If $E_v < \infty$, store the vertex v_i for which ϵ_i is minimum in `destination_table[`v`]`.

3.1.3 Surface Vertex Decimation

If v is a surface vertex, region R consists of all tetrahedra incident on v. A surface vertex can be merged only to another surface vertex; otherwise major surface changes will result. Merging a surface vertex into an adjacent surface vertex may still change the shape of the surface of the grid. In order to consider this shape change, the error of a surface vertex is a weighted combination of the data-based error calculated as described in the previous section, and a geometry error, described here (Figure 3).

The *geometric difference* (G_{diff}) between two triangles is the magnitude of the vector difference between the old triangle surface normal (N_{old}) and the new triangle surface normal (N_{new}) scaled by the old triangle area (A_{old}) divided by the total area of the surface (A_{total}) (creating a dimensionless number).

$$G_{diff} = ||N_{old} - N_{new}|| \frac{A_{old}}{A_{total}} \tag{5}$$

When merging a vertex v with vertex v_i, the n triangles T_{old} that surrounded v before the merge are replaced with

$n-2$ triangles T_{new}. Each new triangle shares one or two edges with one or two old triangles (Figure 3). For each new triangle, we find the geometric difference(s) between it and any old triangles with which it shares an edge. We sum these geometric differences to create the error ϵ_{gi} for the decimation of vertex v by merging it with vertex v_i, and repeat for all v_i surround v. If the merger creates a tetrahedron with a negative volume, ϵ_{gi} is set to infinity.

The final error E_v of a surface vertex v is a weighted combination of the geometric error ϵ_{gi} described here and the data-based error ϵ_i described in the previous section. W_g and W_d are user-defined weights for the geometry-based and data-based errors, respectively. The vertex v_i for which the error is minimum is stored in the `destination_table[`v`]`.

$$E_v = \epsilon_{gi} \times W_g + \epsilon_i \times W_d \tag{6}$$

3.2. Decimation Algorithm

Our decimation algorithm iteratively selects a vertex based on its error and deletes it by merging it to its destination vertex. When a vertex is merged to its neighbor, it drags along with it all the edges that were incident on it. The destination vertex is chosen from the neighbors of the vertex being deleted such that merging the selected vertex to it will cause minimum change in the average density or the mass of the region surrounding the deleted vertex. The input to the decimation algorithm is a tetrahedral grid and an error threshold. The decimation algorithm computes the error for all the vertices and inserts them in a priority queue. It also stores the destination vertex for each vertex in the grid. A priority queue allows us to quickly select the vertex with the minimum error.

Following is our decimation algorithm:

1. Compute initial error for all vertices and insert them in a priority queue.

2. Choose an error threshold ϵ.

3. While there exist vertices with error less than ϵ:

   (a) select and remove vertex v with minimum error from the priority queue

   (b) find the neighbor d of v for merger (from the `destination_table`), and replace vertex v with d for all tetrahedra which are incident on v.

   (c) remove degenerate tetrahedra from the grid.

   (d) recompute the error of neighboring vertices of v (and new *destination vertices* and reinsert them in the priority queue.

This algorithm deletes vertices until a vertex is encountered that has error greater than the user-specified threshold. The number of vertices deleted can also be used as the termination condition. We can also allow only surface or interior vertices to be decimated, or a combination.

4. Experiments and Results

We have examined the decimation of the *lftr* dataset (the Langley fighter from at NASA Langley Research Center). This tetrahedral grid is wrapped around half of the fighter's body and contains 13,832 vertices (3,633 of the surface) and 70,125 tetrahedra. The grid is partitioned into 143,881 polygons with 7,262 polygons on the surface of the grid. We chose three representative views of the volume for presentation (results from these views are consistent with other views of the volume). The three views are shown in Figure 4: one full volume view and two restricted views. The arrows indicate the viewing direction for each view.

We have used direct volume rendering and isosurface methods to evaluate the two data-based error metrics using our decimation algorithm. We have also evaluated the effect of using data-based and geometry-based error metrics on decimating surface vertices.

In addition to measuring decimation using an error metric, we have also measured the quality of the images rendered (for direct volume rendering) using the decimated volume. For this purpose we have compared an image generated using the decimated volume with the image generated (with the same viewing parameters) using the undecimated volume. The sum-of-squares of these differences is accumulated and averaged. The square root of this value is the error in our images generated using the decimated volume. To compute the average, we discard those pixels that are black in both images of the pair. The root-mean-square error (*rms-image-error*) computed in this fashion gives us a measure of the average deviation of a pixel value from the original and is first computed separately for each color channel. We then sum the differences and average them to find a single error value for a particular image.

Experiments were done on an SGI Onyx-2 with 256 Mbytes of main memory using one 195 MHz IP27 processor. All images were 500×500 pixels.

Table 1 reports decimation and speed results of our decimation method using the *mass-metric* and the *density-*

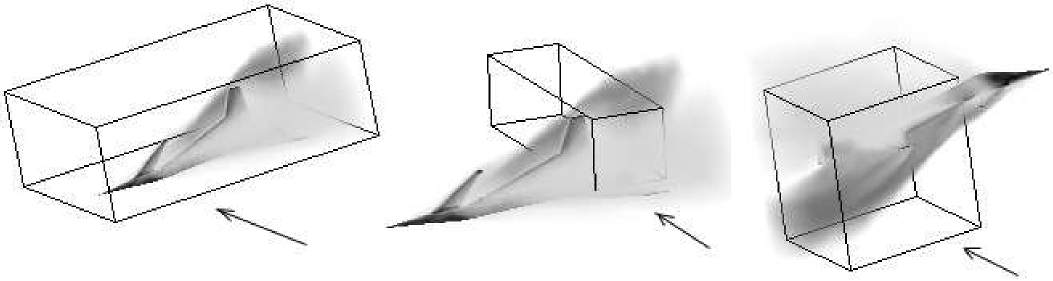

Figure 4. Viewpoints.

				Decimate time	Rendering time		
Error	Polygons kept	Vertices kept	% Deleted		*view-1*	*view-2*	*view-3*
Mass Metric							
0.0	143881	13832	0%	0	7.12	27.12	20.40
4.5×10^{-5}	31965	2831	80%	106.19	5.24	24.18	17.42
2.5×10^{-2}	9027	831	93 %	125.66	3.89	17.71	13.39
Density Metric							
0.0	143881	13832	0%	0	7.12	27.12	20.40
6.8×10^{-5}	29888	2831	80%	121.43	4.22	20.38	14.71
2.5×10^{-4}	8416	831	95%	143.00	2.96	13.97	9.93

Table 1. Decimation using the *mass-metric* and the *density-metric*. Times are in seconds.

metric. We compare 80% and 95% vertex decimation to the original volume. It takes under two minutes to decimate the original volume by these amounts.

4.1. Effect of Decimation on Direct Volume Rendering

We have used a direct volume rendering system based on a generalized software scan conversion of polygons [11]. The scan conversion algorithm generalizes traditional polygon scan-line methods in that it renders semi-transparent regions of space between polygons, as well as opaque surfaces. This method requires no graphics hardware, and produces excellent quality images.

We have found that a volume decimated by as much as 80% is able to generate images without much noticeable difference in image quality, and even 95% decimation produces quite good images though artifacts do appear. The space improvement is quite large. The images rendered from the decimated volumes using the *mass-metric* can be seen in Figure 8. The images show no decimation, 80% decimation, and 95% decimation.

The *mass-metric* fares better than the *density-metric*. A comparison of the images rendered from the decimated volumes compared to those of the undecimated volume using the two error metrics is shown in the plots in Figure 5. For the *mass-metric*, at 80% decimation the average pixel difference between the decimated image and the original image is about 0.8% and even at 95% decimation the average pixel difference is about 1.5%. This measured difference is also supported visually.

Table 1 also reports results on rendering times. The plots in Figure 6 show that for both the *mass-metric* and the *density-metric*, the rendering time does not decrease significantly. For a volume decimated using the *mass-metric* by 80%, the rendering time decreases by about 27%. A volume decimated by 95% can generate images with few artifacts and the rendering time decreases by about 55%. This is due to our direct volume renderer [11], for which rendering time is proportional to the area of polygons rendered. While our decimation greatly reduces the number of geometrical objects, it does not reduce the area of the retained triangles proportionately. The plots in Figure 7 show that the area of the retained triangles does not reduce linearly with the number of vertices. We found that the rendering time varies approximately linearly with the area of the retained triangles. The total triangular area is likely to be less important for direct volume rendering via hardware-assisted projection of Gouraud-shaded triangles.

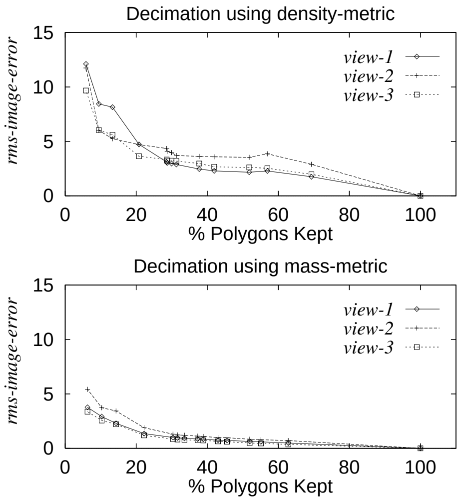

Figure 5. *rms-image-error* for decimation using the two error metrics.

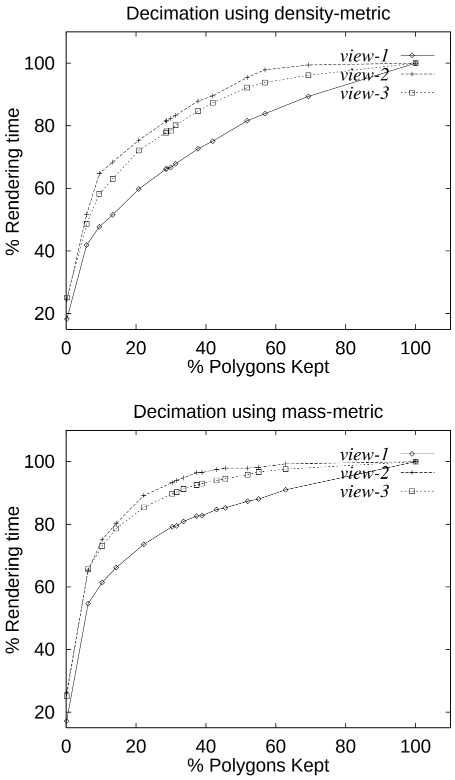

Figure 6. Effect of decimation on rendering times using the two error metrics.

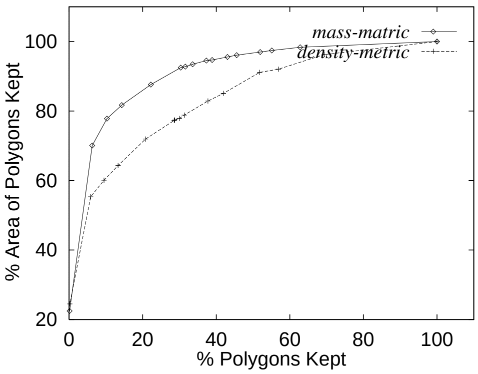

Figure 7. Effect of decimation on area of triangles retained using the two error metrics.

We also notice from the plots in Figure 6 that the rendering times for *density-metric* are lower than those for *mass-metric*. This behavior is easy to explain in terms of the area of the retained triangles. The *mass-metric* prefers to delete vertices which have smaller tetrahedra incident on them because it takes into account the volume of the tetrahedra incident on the vertices. The *density-metric* on the other hand does not discriminate between vertices which have large tetrahedra surrounding them and those that have small tetrahedra surrounding them. As consequence, *mass-metric* tends to retain larger tetrahedra than the *density-metric*.

4.2. Effect of Decimation on Isosurface Rendering

The decimated grid can be used for other visualization techniques such as isosurface generation. We decimated our tetrahedral grid by 80% and generated isosurfaces using FAST, a visualization program developed at NASA [1]. Figure 9 shows two isosurfaces around the body of the aircraft (green: density 1.3, blue: density 0.27). The blue surface represents a very low density vortex core. The left image was generated using an undecimated volume and the right using the same parameters but using an 80% decimated volume. The decimated volume image retained the vortex core and other features.

4.3. Effect of Geometric Versus Data-Based Decimation

Geometric surface variations on the exterior grid faces can have visual impacts beyond that seen by geometric changes in the interior of the structure. We allow the user to weight the impact of geometry-based and data-based error metrics in decimating surfaces vertices. Figure 10 shows the most important of the exterior grid surfaces (that which represents the airplane) rendered as a flat shaded surface. We compare the undecimated surface (in red at the left), to an 80% decimated surface using only data-based decimation (green in center) and an 80% decimated surface using only geometry decimation (red at right). Note the obvious surface degradation in the center image using only data-based decimation, especially over the wing. Figure 11 shows the exterior grid faces representing the plane and a connected face as line drawings. The original grid is red and the decimated grid is green. Using compositing, lines that are identical in both grids appear yellow. This techniques makes it easy to identity grid differences.

5. Conclusion and Future Work

We have presented a new model for tetrahedral grid decimation. Our model includes two data-based error metrics and one geometric error metric, as well as a decimation algorithm that can decimate large tetrahedral grids to generate smaller grids, while preserving the important features of the original volume. Our experiments indicate that we can decimate a grid by as much as 80% with excellent quality of the volume rendered images. Even at 95% decimation, the images show few visual artifacts. Our *rms-image-error* image-based error metric agrees well with the visual comparison and gives an objective measure of image quality. We have found that the *mass-metric* produces a better quality decimation than the *density-metric*. We find that geometry-based surface decimation is necessary to avoid artifacts due to geometric errors on exterior mesh surface. Rendering times using our direct volume renderer did not decrease proportionately with decimation, but this effect is dependent upon the type of visualization approach and algorithm being used.

Our error metrics take into account the change in data values inside the volume. We would like the error metrics to also take into account the shape of the tetrahedra. We believe that if the tetrahedra in the decimated volume are not long and skinny then the total triangular area will reduce proportionately with the number of vertices. We are currently investigating metrics that associate a penalty for creating long skinny tetrahedra. Also, our error metrics are local in nature because they are based on the local changes in the grid. In the future, we would like to investigate metrics that give us an estimate of the global error in a decimated volume. We have investigated the effect of decimation on volume rendering methods. In the future we would like to study the effect of decimation on techniques like stream-line visualization and other vector field visualizations.

Acknowledgments

Funds for the support of this study have been allocated by NAS/NASA Ames Research Center, Moffett Field, California, Grants Numbers NAG 2-991 and NAG 2-1239, and by the National Science Foundation, Grants Number CCR 9503829 and CDA 9724237. We wish to thank Jiannhwa Sun and Tom Raffill for programming assistance.

References

[1] G. V. Bancroft, F. J. Merritt, T. C. Plessel, P. G. Kelaita, R. K. McCabe, and A. Globus. FAST: A multi-processed environment for visualization of computational fluid dynamics. In *Visualization '90*, pages 14–27, San Francisco, CA., October 1990. IEEE.

[2] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, June 1997.

[3] P. Cignoni, L. D. Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplical complexes. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 19–26, October 1994.

[4] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4), December 1997. (In press).

[5] M. Garland and P. S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, School of Computer Science, 1995.

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

[7] R. Grosso, C. Lürig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *Proceedings of Visualization '97*, pages 387–394, October 1997.

[8] W. J. Schröder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Proceedings of SIGGRAPH'92*, volume 26, pages 65–60, July 1992.

[9] M. Sramek and A. Kaufman. Object voxelization by filtering. In *ACM Symposium on Volume Visualization*, pages 111–118, 1998.

[10] I. J. Trotts, B. Hamann, K. I. Joy, and D. F. Wiley. Simplification of tetrahedral meshes. In *Proceedings of Visualization '98*, pages 287–295, October 1998.

[11] J. Wilhelms, A. Van Gelder, P. Tarantino, and J. Gibbs. Hierarchical and parallelizable direct volume rendering for irregular and multiple grids. In *Proceedings of Visualization'96*, pages 57–64, October 1996.

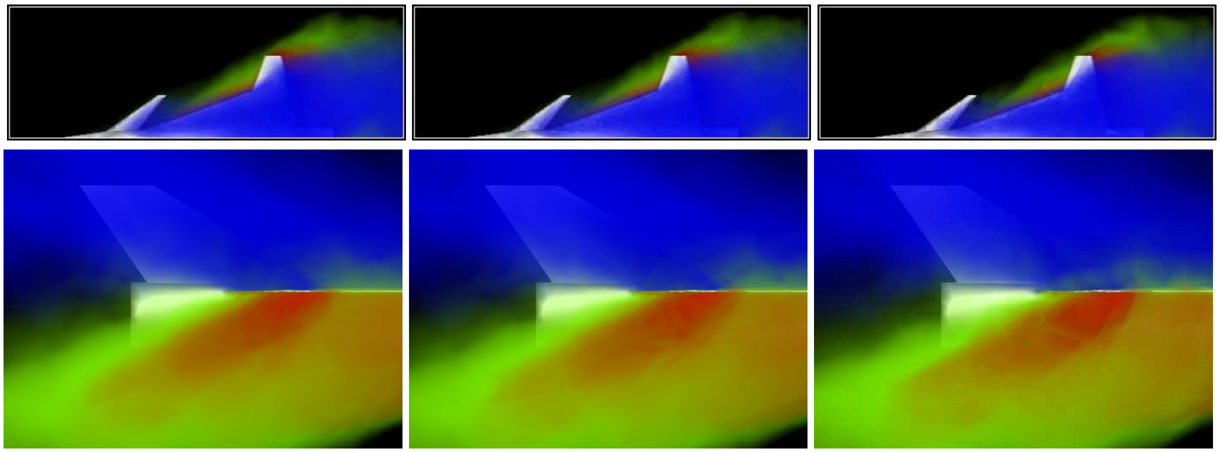

Figure 8. Decimation using *mass-metric*. Left: no decimation; Middle: 80%; Right: 95%.

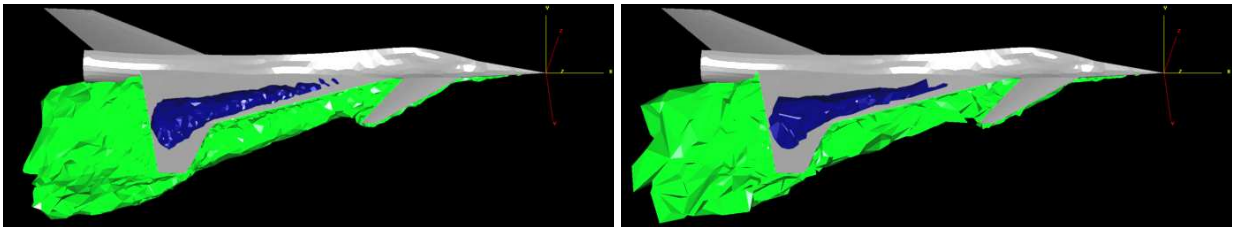

Figure 9. Comparison of isosurfaces. Left: no decimation; Right: 80% decimation.

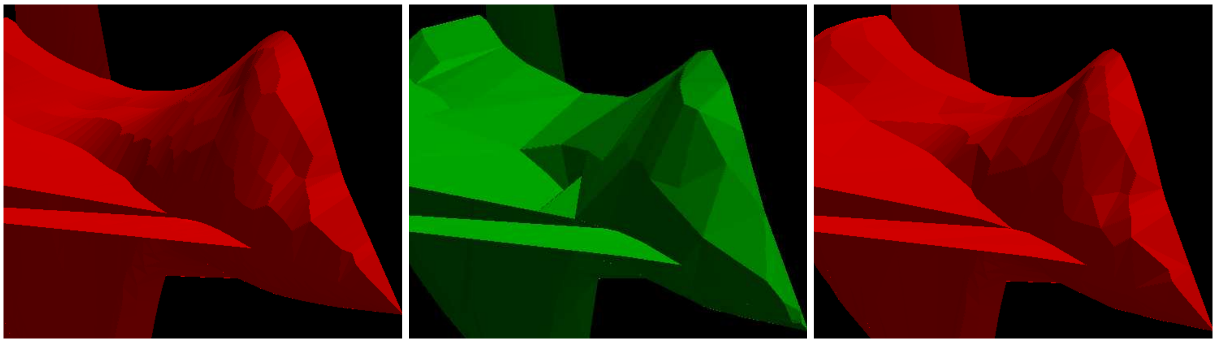

Figure 10. Comparing the surface of the grid. Left: no decimation; Middle: 80% decimation with data-based decimation only; Right: 80% decimation with geometry decimation only.

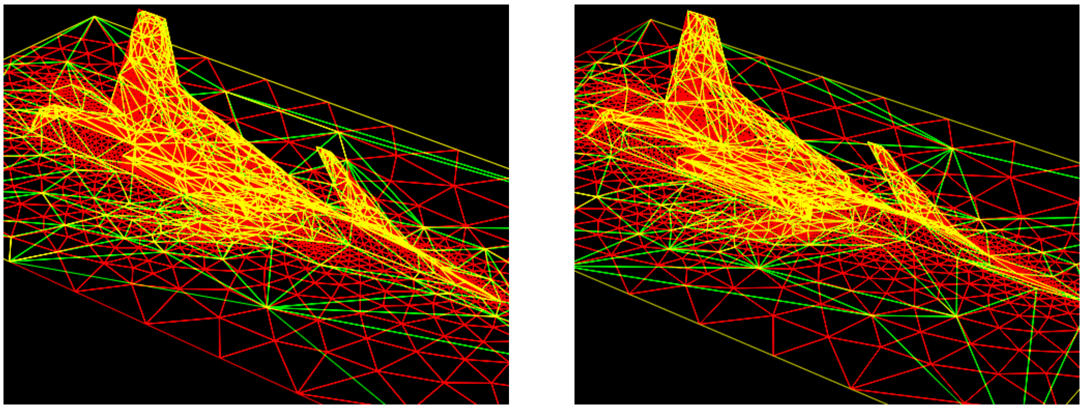

Figure 11. Left: 80% data-based error; Right: 80% geometry-based error.