SCI INSTITUTE
TECHNICAL REPORT

Simplification of Unstructured Tetrahedral Meshes by Point-Sampling

Dirce Uesu, Louis Bavoil, Shachar Fleishman, Claudio T. Silva

UUSCI-2004-005

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

July 13, 2004

Abstract:

Tetrahedral meshes are widely used in scientific computing for representing three-dimensional scalar, vector, and tensor fields. Their size and complexity limit the performance of many visualization algorithms, making it hard to achieve interactive visualization. The use of simplified models is one way to enable the real-time exploration of these datasets. In this paper, we propose a novel technique for simplifying large unstructured meshes.

Most current techniques simplify the geometry of the mesh using edge collapses. Our technique simplifies the underlying scalar field directly. This is done by segmenting the original scalar field into two pieces: the boundary of the original domain and the interior samples of the scalar field. We then simplify each piece separately, taking into account proper error bounds. Finally, we combine the simplified domain boundary and scalar field into a complete, simplified mesh that can be visualized with standard unstructured data visualization tools.

Our technique tends to be faster than edge-collapse based approaches because we do not need to construct a complete representation of the mesh in memory. Also, our memory consumption is considerably lower when compared to edge-collapse approaches. We have been able to simplify large meshes with about 1.4 million cells in less than 8 minutes. The most expensive step in our technique is the computation of a conforming Delaunay tetrahedralization for the simplified data, which consumes over 75% of the time.

THE UNIVERSITY OF UTAH

Simplification of Unstructured Tetrahedral Meshes by Point-Sampling

Dirce Uesu * Louis Bavoil * Shachar Fleishman * Cláudio T. Silva *

Scientific Computing and Imaging Institute, University of Utah

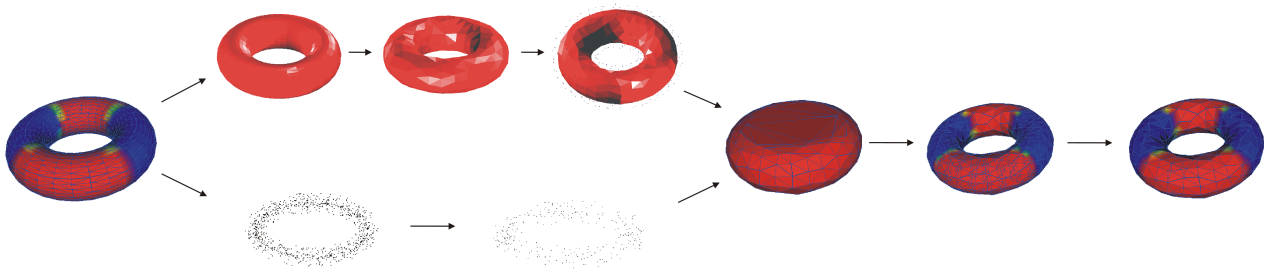

Figure 1: Graphical representation of our simplification pipeline. Our technique separates the original scalar field (shown on the left) into two pieces: the boundary of the original domain and the interior samples of the scalar field. We then simplify each piece separately, taking into account proper error bounds. Finally, we combine the simplified domain boundary and scalar field into a complete, simplified mesh (shown on the right) by computing a conforming Delaunay tetrahedralization and removing the external tetrahedra.

ABSTRACT

Tetrahedral meshes are widely used in scientific computing for representing three-dimensional scalar, vector, and tensor fields. Their size and complexity limit the performance of many visualization algorithms, making it hard to achieve interactive visualization. The use of *simplified models* is one way to enable the real-time exploration of these datasets. In this paper, we propose a novel technique for simplifying large unstructured meshes.

Most current techniques simplify the geometry of the mesh using edge collapses. Our technique simplifies the underlying scalar field directly. This is done by segmenting the original scalar field into two pieces: the boundary of the original domain and the interior samples of the scalar field. We then simplify each piece separately, taking into account proper error bounds. Finally, we combine the simplified domain boundary and scalar field into a complete, simplified mesh that can be visualized with standard unstructured data visualization tools.

Our technique tends to be faster than edge-collapse based approaches because we do not need to construct a complete representation of the mesh in memory. Also, our memory consumption is considerably lower when compared to edge-collapse approaches. We have been able to simplify large meshes with about 1.4 million cells in less than 8 minutes. The most expensive step in our technique is the computation of a conforming Delaunay tetrahedralization for the simplified data, which consumes over 75% of the time.

1 INTRODUCTION

In scientific computing, it is common to represent a scalar function $f : D \subseteq \mathbb{R}^3 \rightarrow \mathbb{R}$ as sampled data by defining it over a domain D, which is represented as a tetrahedral mesh. For visualization purposes, many choose to define the function f as linear inside each tetrahedron of the mesh. In this case, the function is completely defined by assigning values at each vertex $v_i(x,y,z)$, and is piecewise linear over the whole domain. The domain D becomes a 3-dimensional simplicial complex defined by a collection of simplices c_i. It is important to distinguish the domain D from the scalar field f. The purpose of visualization techniques, such as isosurface generation [17] and direct volume rendering [18] are to study intrinsic properties of the scalar field f. The time and space complexity of these techniques are heavily dependent on the size and shape of the domain D.

For large datasets, it is not possible to achieve interactive visualization. In these cases, it is often useful to generate a reduced-resolution scalar field $\bar{f} : \bar{D} \subseteq \mathbb{R}^3 \rightarrow \mathbb{R}$, such that:

- the new scalar field $\bar{f}$ approximates f in some natural way, *i.e.*, $|\bar{f} - f| \leq \varepsilon$;
- the new domain $\bar{D}$ is smaller than D.

There are many possible ways to compute $\bar{f}$ from f. Recently, many techniques have been proposed that simplify tetrahedral meshes by the use of edge (1-simplex) and tetrahedron (3-simplex) collapses (see, *e.g.*, [4, 20, 25]) on the domain D. These techniques work similarly to triangle-based simplification techniques [14, 11] and use connectivity information to incrementally cull simplices c_i from the domain (*i.e.*, when a 1-complex is collapsed, several 2- and 3-simplices become degenerate and can be removed from the tetrahedralization). Most techniques order the collapses using some type of error criteria, stopping when the size of the domain $|\bar{D}|$ reaches a user-defined target number of simplices n (*i.e.*, the simplification stops once $|\bar{D}| < n$) or when the function $\bar{f}$ reaches a maximum user-defined error bound $\varepsilon > 0$ (*i.e.*, $|\bar{f} - f| \leq \varepsilon$).

With surfaces embedded in three-dimensions, it is quite natural to try to maintain the shape of the overall mesh during simplification. For scalar fields, the overall geometry of the domain D is not nearly as important. The domain is used to represent the subset of $\mathbb{R}^3$ where the scalar field f is defined. For this, we only need to maintain the shape and topology of the boundary ∂D. In our work, instead of slowly building $\bar{D}$ from D using a series of collapses, we build the boundary of $\bar{D}$, $\bar{B}$, by simplifying ∂D, while completely ignoring the connectivity of the interior. For the interior, we use a

*{uesu,bavoil,shachar,csilva}@sci.utah.edu

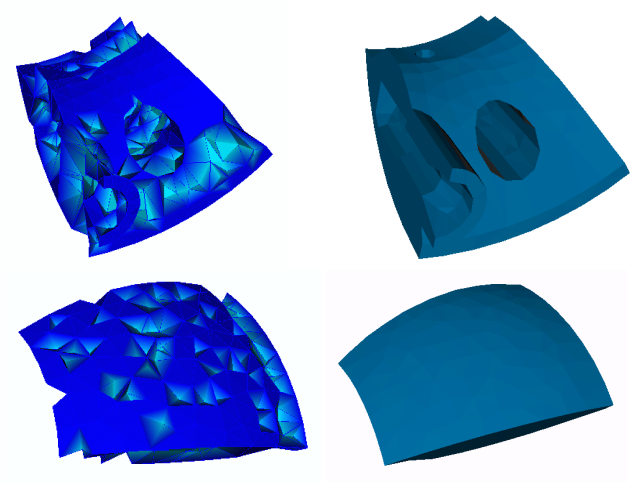

Figure 2: Comparison with [10]: The result of Farias *et al.* for the SPX dataset is shown on the left; observe the damaged boundary. The result of the algorithm proposed in this paper is shown of the right.

point-sampling approach to build the 0-simplices $\bar{V} = \{\bar{v_i}\}$ that are used to define the final domain of $\bar{f}$. Then we use a tetrahedralization algorithm to create the simplicial complex $\bar{D}$ by combining $\bar{B}$ and the set $\bar{V}$.

To summarize, our technique works directly on the underlying scalar field. This is done by segmenting the original scalar field into two pieces: the boundary of the original domain and the interior samples of the scalar field. We simplify each piece separately, taking into account proper error bounds. Finally, we combine the simplified domain boundary and scalar field into a complete simplified mesh. Our technique tends to be faster than edge-collapse based approaches because we do not need to construct a complete representation of the mesh in memory. Also, note that our memory consumption is considerably lower when compared to edge-collapse approaches.

Our new algorithm builds on our previous work presented in [10]. Although based on related ideas, our previous method was quite rudimentary, and was meant to be used primarily for low-quality renderings. In particular, it did not provide error bounds on either the interior or boundary of the simplified mesh.

The remainder of this paper is organized as follows. We summarize related work in Section 2. In Section 3, we describe our algorithm. Section 4 presents our results. In Section 5, we discuss different trade-offs of our approach. Finally, in Section 6, we provide final remarks and directions for future work.

2 Related work

Most of tetrahedral mesh simplification to date uses the edge-collapse approach. These algorithms employ various edge-cost functions to preserve the boundary. Trotts *et al.* [25] extend the technique of Gieng *et al.* [13] for the simplification of triangle meshes to tetrahedral meshes. They collapse individual tetrahedron by collapsing three edges and evaluating the resulting error. Their technique attempts to preserve the boundary of the tetrahedral mesh, as much as possible, by the use of boundary constraints. In their followup work, Trotts *et al.* [24] improve upon their previous methods by avoiding the topological problems created from collapsing a tetrahedron. Their strategy was to use only one edge collapse instead of a sequence of three edge collapses. They presented two algorithms: the first uses a priority queue which sorts the tetrahedron by error cost; the second uses a greedy strategy which computes the local error resulting from the edge collapses such that the mesh remains within a tolerance of the original.

Staadt and Gross [23] extend the work of Hoppe [14] for progressive tetrahedralization. They discuss intersections, inversions, and degenerations of tetrahedra for a robust implementation of edge collapsing. They also redefine the cost function by considering the volume preservation and gradient.

Van Gelder *et al.* [12] evaluate the effect of decimation by comparing two data-based error metrics: the mass-metric and the density-metric.

Chopra and Meyer [4] propose a fast algorithm to progressive simplification: *TetFusion*. The idea of this algorithm is to use a tetrahedral collapse operation in which one tetrahedron is collapsed onto its barycenter. In their work they preserve the boundary surface by avoiding a simplification of any tetrahedron on the boundary.

Chiang and Lu [3] construct multiple levels of the tetrahedral volume, preserving the topology of all isosurfaces. The algorithm simplifies the tetrahedral mesh in two phases. In the segmentation phase, it classifies each vertex into critical or non-critical points and uses the contour tree algorithm of Carr *et al.* [2] to compute the fully augmented contour tree. The augmented contour tree is used to identify topologically equivalent regions. In the simplification phase, the algorithm uses edge collapse operations in which each topological equivalent region is simplified independently. Note that any change to the transfer-function requires a recomputation of the simplification algorithm.

In the tetrahedral mesh simplification of Natarajan and Edelsbrunner [20], they use edge contractions with the modified quadratic error metric of Garland and Heckbert [11]. The quadratic cost function preserves the density map, improves the mesh quality in terms of angles, and preserves the global topological type of the mesh.

All of the papers mentioned use mesh connectivity to simplify the tetrahedral mesh. Our work uses a different approach, we simplify the boundary surface and simplify the interior points independently, then rebuild the connectivity using Delaunay tetrahedralization.

Farias *et al.* [10] present an algorithm to improve the speed of volume rendering for unstructured grids using an approach similar to the one proposed in this paper. This technique uses the following steps. First, a set of sample points are generated by sampling the interior points. Second, the boundary is simplified separately, leading to a collection of simplified points. From the simplified boundary, a set of external "ghost" vertices are generated. They then discard the geometry of the boundary (unlike our technique). The final pass is a Delaunay tetrahedralization of all the points, where any cell that contains a "ghost" vertex is discarded. A key difference between this algorithm and ours is that we preserve the boundary surface of the mesh, using a conforming Delaunay tetrahedralization algorithm [7, 19]. Because the geometry of the boundary is discarded, the algorithm employed by Farias *et al.* [10] could suffer from severe boundary anomalies. See Figure 2.

Evaluating the quality of the simplification is as important as the simplification algorithm. We adopt the approach of Cignoni *et al.* [5] who presented a methodology to evaluate the approximation error of the simplification. Their multi-variate error function uses both the domain error (geometry of the boundary) and the field error components.

3 Our Simplification Algorithm

In this work, we consider the tetrahedral mesh as a function $f : D \subseteq \mathbb{R}^3 \rightarrow \mathbb{R}$, therefore, we simplify the function rather than the

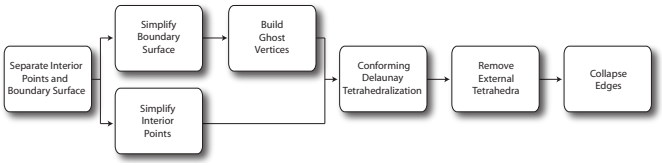

Figure 3: Main steps of our simplification algorithm described in detail in Section 3.

geometry of the tetrahedral mesh. We present a multi-stage algorithm (see Figure 3). First, we separate the interior points from the boundary. We simplify the boundary using a modified surface simplification algorithm that takes into account the scalar field defined at the vertices, and we use the output of the simplification process to generate some extra ghost vertices (our use of ghost vertices is a bit different than in the Farias *et al.* [10] algorithm, here they are used mostly for robustness). We simplify the interior points using a *k-d* tree partition of the points of the mesh. Then, we reconstruct a simplified tetrahedral mesh by using conforming Delaunay tetrahedralization (CDT) on the interior points while taking into consideration the simplified boundary. At this point, we have a convex mesh that approximates D in a subset. Next, we remove all the tetrahedra that lie outside of the boundary of D (to be precise, outside of a simplified version of the boundary of D). A side effect of the CDT is a collection of unwanted Steiner points. Our final step is to remove as many of these points as possible. All these steps are further detailed below.

3.1 Interior simplification

Our goal is to build a simplification of the volumetric scalar field in such a way that *features* (*e.g.*, isosurfaces) of the volume are well preserved. We know that the function reconstructed from the simplified model is a linear interpolation of tetrahedra, and thus, we do not consider higher order interpolation techniques such as *radial basis function* interpolation. Our approach is to sub-sample the input vertices using a space-partitioning data structure, where the scalar values of each node have a bounded variation. We build a hierarchy of the scalar function using a *k-d* tree, trying to group points with similar scalar values together. The final level of detail is obtained by sampling each leaf of the tree by the point that has a value that is closest to the mean scalar value of the cell. (See Figure 5.)

The method for splitting a node of the tree determines the quality of the simplification of the interior, *i.e.* the error in the reconstructed scalar-function is a function of the number of sampled points. A node is split if the variation of the values in the node is larger than a user-defined threshold. To subdivide the nodes, we have to determine an axis and a position on this axis. To determine the axis among the three possible, we find the points $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ of minimum and maximum scalar values, and we take the axis e_i for which the norm of the projection of the vector $\mathbf{g} = \mathbf{x}_{max} - \mathbf{x}_{min}$ is the greatest. This direction is the direction along which the scalar value varies the most. The problem is then to determine where to cut the bounding box of the node along the chosen axis. Assuming that the sampling of the subdivided nodes is perfect, the best position to cut is where the sum of the maximum variations of the scalar values on both sides is minimal. As shown on Figure 4, only picking one point per subdivided cell can introduce a significant error if the size of the cell is big. This can be solved by limiting the geometric size of the cells, but this would introduce another parameter, which we prefer to avoid. We have tested the exact computation using an exhaustive search over all of the points in the node, but have decided to use a heuristic that works well in practice. We cut the selected

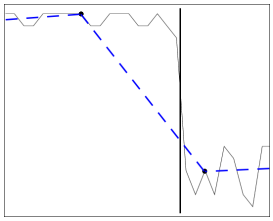

Figure 4: An example of splitting a cell that only contains coplanar points, with the x-axis as the splitting direction. The plain line is the original scalar field, the dash line is the approximation from sampling one point on each side of the split.

axis at the middle of the vector $\mathbf{g}$.

3.2 Simplification of the boundary meshes

The boundary surface is defined by the faces of the tetrahedral mesh that belong to a single tetrahedron. It comprises the geometric and topological aspects of the shape, therefore, we apply geometric constraints for the boundary simplification to preserve these qualities. A boundary simplification algorithm should meet the following criteria:

- preserve the shape of the object, that is, the Hausdorff distance between the simplified model and the input model should be as small as possible,
- preserve the topology of the object (some of the most important features in tetrahedral volumes lie near holes and cavities),
- preserve the scalar function on the boundary. Therefore, geometric simplification algorithms like the *Simplification Envelopes* technique [6] which creates large triangles on flat surfaces are not appropriate.

In addition, the algorithm of the conforming Delaunay tetrahedralization works better with well-shaped triangulations. Skinny triangles make this process more time consuming and less efficient because of the addition of Steiner points on sharp corners [7]. Therefore, we also try to generate simplified boundary meshes with regular triangles.

We use the program `coarsen` provided by the GTS library [21], which is based on edge collapses. The potential edges to be collapsed are inserted into a priority queue. Each edge has a cost function, which is computed by simulating the edge collapse and estimating the quality of the result. The edge with the lowest cost is collapsed until the target number of edges is reached.

GTS implements the edge-collapse algorithm described in [16]. As with any edge-collapse algorithm, it is characterized by the placement algorithm of the new vertices resulting from a collapse, and by the cost function $f_C(e,v)$.

The position of a new vertex can be computed by using the different methods described in [16]: volume preservation, boundary preservation, weighted average of volume and boundary optimization, and triangle shape. Moreover, GTS has code for a weighted average for volume and shape optimization. We use the volume preservation constraint, followed by the coupled volume and shape optimization. Finally, the shape optimization constraint is applied if necessary.

The weight of the volume optimization is fixed to 1, and the weight of the shape optimization for an edge e is

$$w_s(e) = \lambda \cdot L(e)^2$$

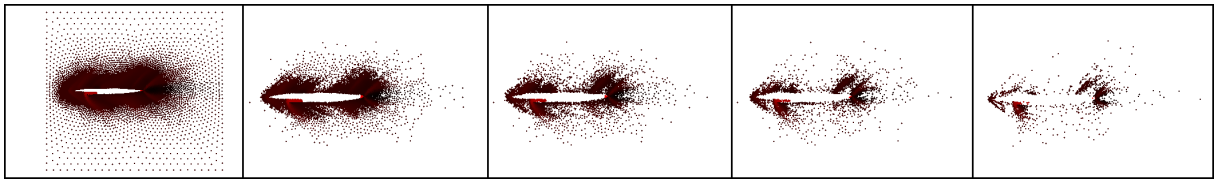

Figure 5: One slice of sampled points for "fighter" LODs highlighting our adaptive sampling technique.

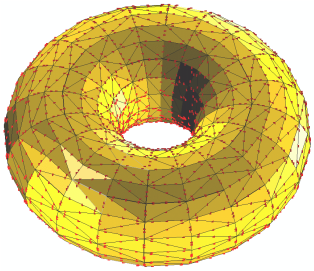

Figure 6: A torus with Steiner points in red.

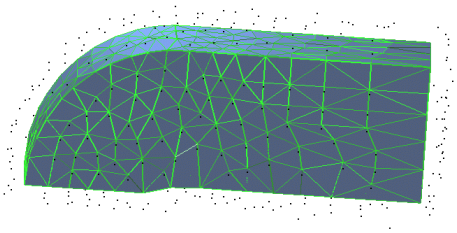

Figure 7: The simplified boundary of the "blunt fin" dataset with ghost vertices.

where λ is a parameter which we will refer to later as the *shape optimization weight*. For the cost function we use:

$$f_C(e,v) = f_S(e,v) \cdot L(e)^2$$

where $f_S(e,v)$ is the measure of the triangle shape quality introduced by [16].

The scalar value of a new vertex resulting from an edge collapse is computed by a linear interpolation of the two adjacent vertices.

3.3 Tetrahedral mesh reconstruction

We wish to remain in the same domain that we started with, allowing further processing and visualization of the simplified model; thus we reconstruct a tetrahedral mesh from the simplified boundary mesh and the simplified scalar function of the interior.

The reconstruction algorithm is based on Delaunay tetrahedralization of the boundary faces and the simplified interior points. Delaunay tetrahedralizations (DT) are very well known and studied geometric entities (see, e.g., [9, Chapter 5]). A basic property that characterizes this geometric structure is that a tetrahedron belongs to the DT of a point set if the circumsphere passing through the four vertices is empty, meaning no other point lies inside the circumsphere. Under some non-degeneracy conditions (no 5 points co-spherical), this property completely characterizes DTs and the DT is unique.

Since the shape formed by Delaunay tetrahedralization is the convex hull of the input set of points and we wish to be able to handle non-convex shapes, we use conforming Delaunay tetrahedralization (CDT) (see below) to make sure that the boundary is indeed part of the tetrahedralization. In a post-processing step, we remove tetrahedra that are outside the object as well as Steiner points inserted on the boundary by conforming Delaunay tetrahedralization. For removing the outside tetrahedra, we add ghost vertices prior to the tetrahedralization of the points. Following is a detailed description of each step.

Conforming Delaunay Tetrahedralization In order to preserve the simplified boundary S in the building of the Delaunay tetrahedralization, we use the conforming Delaunay tetrahedralization algorithm introduced by Cohen-Steiner *et al.* [7]. Given a set of faces $\{f_i\}$ that need to be included in a DT, the idea behind *conforming* Delaunay tetrahedralizations is to add points to the original input set so that the DT of the new point set (consisting of the original points *plus* the newly added points) is such that each face f_i can be expressed as the union of a collection of faces of the DT. The newly added points are often called *Steiner* points.

A challenge in computing a conforming DT is minimizing the number of Steiner points and avoiding the generation of very small tetrahedra. While techniques for computing the traditional DT of point sites are well known, and reliable code exists, conforming DT algorithms are still in active development [19, 7]. This particular technique for adding Steiner points affects the termination of the algorithm and also the quantity and quality of added geometry.

Consider the dataset with the simplified boundary surface S, the set of simplified interior vertices and ghost vertices. The conforming Delaunay tetrahedralization finds a new set of points whose Delaunay tetrahedralization conforms to S. This set of points includes the vertices of S and a number of additional Steiner points, shown in Figure 6.

The added Steiner points are not part of the simplified boundary that we wish to preserve, thus we remove them as well as the tetrahedra that lie outside of the shape.

Removing external tetrahedra The Delaunay tetrahedralization of non-convex objects tetrahedralizes the convex hull of the points. Thus, we need to remove the additional tetrahedra generated in the nonconvex regions.

Ghost vertices provide one additional accurate criterion for identifying external tetrahedra, since any tetrahedron with at least one ghost vertex is an external tetrahedron.

Given a vertex $\mathbf{v}$ of the boundary surface S and the set of faces F adjacent to $\mathbf{v}$. For each vertex $\mathbf{v}$, we define a *ghost vertex* $\mathbf{g_v}$ of $\mathbf{v}$ as a point outside of S in the direction of the average outward normal to the faces in F. The distance between $\mathbf{g_v}$ and $\mathbf{v}$ is set to be less than the distance between $\mathbf{v}$ and the closest vertex to it in S. Figure 7 shows the ghost vertices for the "blunt fin" dataset.

First, we remove every tetrahedron with at least one ghost vertex. Because the Delaunay tetrahedralization is conforming, all tetrahedra with one ghost vertex are external tetrahedra and can be directly removed. Simultaneously, we mark every tetrahedron that has at least one interior vertex as *inside* the volumetric mesh.

Next, we classify each tetrahedron as *inside*, *outside* or *undecided*. Each interior tetrahedron t is marked as inside, every adja-

Dataset	fighter				delta				f117			
Parameters:												
Boundary LOD (% of edges)	8%	8%	8%	8%	6%	6%	6%	6%	10%	10%	10%	10%
Clustering bound (% of range)	0.5%	1.2%	2.3%	4.7%	0.5%	2.0%	5.5%	10%	0.3%	1.4%	2.9%	7.1%
Level of detail:												
Num. of tetrahedra	598K	326K	172K	78K	474K	228K	109K	71K	162K	77K	42K	14K
Percentage of the original tetrahedra	42%	23%	12%	6%	47%	22%	11%	7%	72%	34%	19%	7%
Field error:												
Maximum error (%)	51.1	51.3	53.0	53.0	64.8	81.2	64.5	78.6	66.9	66.9	64.0	62.6
Mean error (%)	0.42	0.50	0.65	0.91	0.72	1.00	1.21	1.86	0.49	0.65	0.80	1.21
Standard deviation (%)	1.63	1.67	1.77	1.90	3.76	4.64	3.85	4.65	1.50	1.54	1.61	1.87
RMS error (%)	1.68	1.74	1.88	2.11	3.83	4.74	4.04	5.01	1.58	1.67	1.80	2.23

Table 1: Field error of the LODs presented in Figure 10. The field errors are in percentage of the range of the scalar values of the reference mesh, and the geometric error in percentage of the bounding-box diagonal of the reference mesh. We use a shape optimization weight of 0.1 for the simplifications of the boundary surfaces. The clustering bound is the maximal variation of the scalar values in each cell of the *k-d* tree of the sample points. The RMS error is the Root-Mean-Square error.

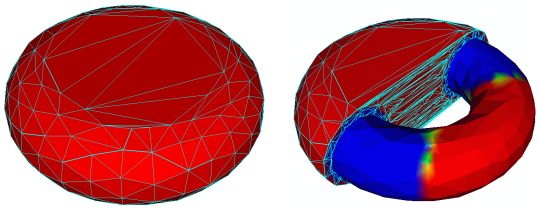

Figure 8: The convex hull of the "torus" (left). The torus with external tetrahedra partially removed (right).

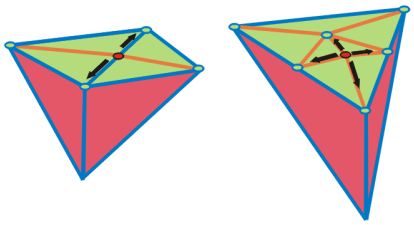

Figure 9: Collapsing Steiner points: a Steiner point on an edge is shown on the left, this point must collapse to a vertex of the original mesh. The right figures illustrates a Steiner point in the middle of a face; this point can collapse to any of its neighbors.

cent tetrahedron to the faces of t are marked as inside if the adjacent face is not a boundary face. For each outside tetrahedron, we repeat the same process marking its adjacent tetrahedra as outside. Figure 8 shows the convex hull of the "torus" dataset with ghost vertices and a partial result from the process.

Removing Steiner points The conforming Delaunay tetrahedralization adds some Steiner points, as described above. Our final step is to remove these points. We use an edge collapse approach to simplify the boundary with Steiner points S^* to the simplified boundary S. Each edge e that has at least one Steiner vertex is collapsed.

An edge collapse can cause self-intersections of tetrahedra that can be identified if the sign of the volume of the tetrahedron is inverted [23, 25]. An edge collapse is performed only if it does not cause self-intersection. In this case we have a *valid edge collapse*.

We have to consider two cases. First, Steiner vertices that are on an edge of S are collapsed only along an edge of S. Second, Steiner vertices that are inside a face of S use the first valid edge collapse that we find. See Figure 9. We continue this process until there are no additional valid edge collapses.

4 Results

We have implemented the algorithms described in Section 3 in C++ with the exception of the boundary simplification code, which is based on the simplification code available in GTS [21], and the conforming Delaunay tetrahedralization code, which is a CGAL-based implementation of the Cohen-Steiner *et al.* [7] algorithm (graciously provided to us by David Cohen-Steiner). Everything is integrated to allow for simple use. In order to assess the speed and quality of our technique, we ran a large number of tests using diverse datasets. We briefly summarize some of our results below.

We used a Pentium 4 at 3.20 GHz with 2 GB of RAM to generate our results. Our simplification times are quite fast. For instance, as Table 2 shows, we can simplify the "fighter" dataset, which has over 1.4 million tetrahedra, to 23% of the original number of tetrahedra in less than 2 minutes, using a maximum of 335 MB.

We found that global error estimation numbers (see next section) do not necessarily give a good measure of the simplification results, making it difficult to understand the similarities and discrepancies between the different simplified models. Others (e.g., [3, 20]) have used one isosurface of the data to compare the different LODs. Here, we use semi-transparent volume rendering of the models to present holistic views of the overall simplification quality. The images shown in Figure 10 have been generated with a high-quality volume rendering approach [1]. Using a histogram, shown in Figure 11, we can see that our technique preserves the global structure of the scalar field.

4.1 Error estimation

The error of the simplification of a tetrahedra mesh is estimated by a field error, which is the variation of the scalar values in the interior of a reference mesh and a simplified mesh, and a domain error, which is the variation of the geometry of the boundary surface [5]. We report the maximum error, the absolute mean error, the standard deviation of the error, and the mean squared error, in percent of the range of scalar values of M_1. See Table 1.

To measure the domain error, we use the tool *MeshDev* [22]. It measures the geometric deviation between two surfaces. It uses the vertices of the reference mesh as samples to compute the error. In addition, we use its subsampling option with a density of 0.1 to add samples inside the triangles. For the field error, we use a similar approach to [5]. Given a reference tetrahedral mesh

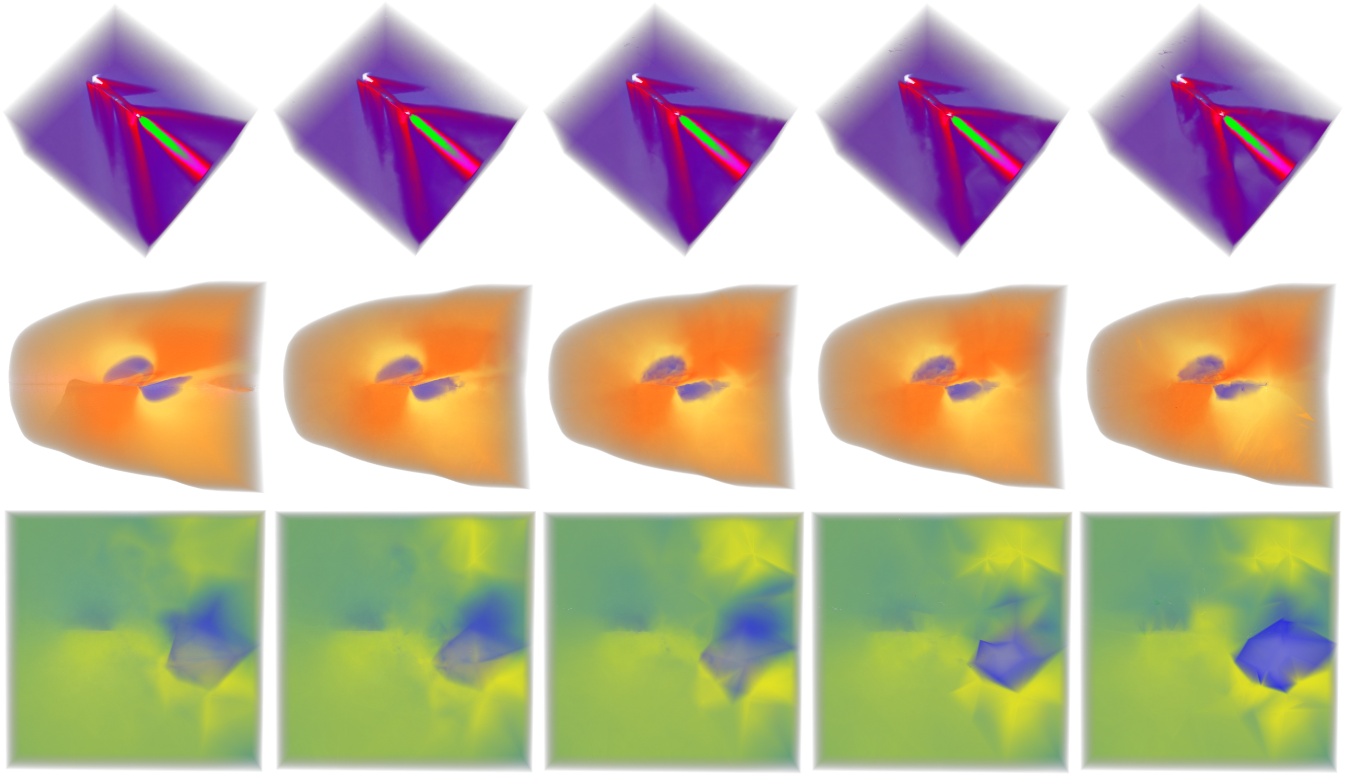

Figure 10: Volume rendering of LODs of different datasets. First row: "fighter" – 1.4M tets (100%), 598K tets (42%), 326K tets (23%), 172K tets (12%), 78K tets (6%). Second row: "delta" – 1M tets (100%), 474M tets (47%), 228K tets (22%), 109K tets (11%) 71K tets (7%). Third row: "f117" – 224K tets (100%), 162K tets (72%), 77K tets (34%), 42K tets, (19%), 15K tets (7%).

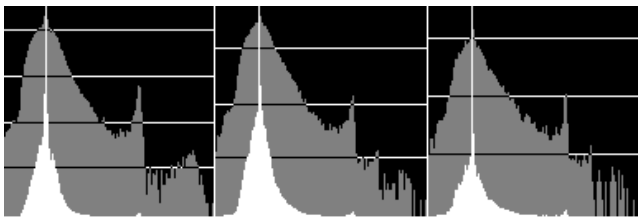

Figure 11: Histograms of the scalar values of LODs of the "fighter" dataset. 1.4M tets (100%); 354K tets (25%); 99K tets (7%). The white graph is linearly scaled histogram, the gray graph is log-scaled histogram, and the horizontal lines are the decades on the log scale.

M_1 and a simplified version of it M_2, we sample the domain of M_1 at the vertices of M_1. At each sampled point (x,y,z), an error value $e(x,y,z) = |M_1(x,y,z) - M_2(x,y,z)|$ is computed. Contrary to Cignoni's approach [5], when a sample taken in the domain of M_1 is outside of the domain of M_2, we ignore that point, since the point is undefined in M_2 and the domain error accounts for this error. This leads us to ignore 6.8% of the samples on our simplification of the "fighter" dataset with 10,000 boundary edges.

Error-value computation The value at a sample point **p** in the volume of M_1 is computed by a linear interpolation of the scalar values at the vertices of the tetrahedron in which it was picked, using barycentric coordinates. To find the value of **p** on the second mesh, it is first necessary to find a tetrahedron that contains **p** in this mesh. To do this efficiently, we build a *k-d* tree of M_2 in a way that if **p** is in a tetrahedron, then this tetrahedron is in the cell of the *k-d* tree containing **p**. Then, to get the value of the scalar field, we find the cell containing **p** and for every tetrahedron t of this cell, we check if **p** is inside t by computing the signed distances to the oriented planes defined by the faces. These planes are oriented using the fourth vertex of each tetrahedron. To determine if **p** is in t, we could also compute the barycentric coordinates of **p** inside of t and check if all the barycentric coordinates are positive. However, using the distances to the planes is faster by a factor of ten.

5 Discussion

Our main motivation in doing this work is to search for an accurate simplification technique that is faster than existing edge-collapse based approaches. Those have shown to be quite slow, taking between one to two orders of magnitute longer to compute simplifications than our approach. From our results presented in this paper, it appears that the quality of a point-sampling approach like the one outlined here can be quite competitive with the quality of the previous techniques, while our technique is substantially cheaper to compute. In a nutshell, the main advantages of our approach are the lower memory consumption and the overall improved speed. Still, there are many nice features of other techniques that our current technique does not possess. In particular, we do not have fine control of topological features, such as the work of Chiang and Lu [3]. One way to potentially address this shortcoming of our technique is to explore domain segmentation along user-defined features, and to take these extra "surfaces" into consideration when performing the sampling and the reconstruction of the final mesh.

An important issue is the use of the conforming Delaunay tetrahedralization. This is a powerful geometric operation that we use in our algorithm to be able to preserve the boundary of the domain. It greatly simplifies the implementation of our technique.

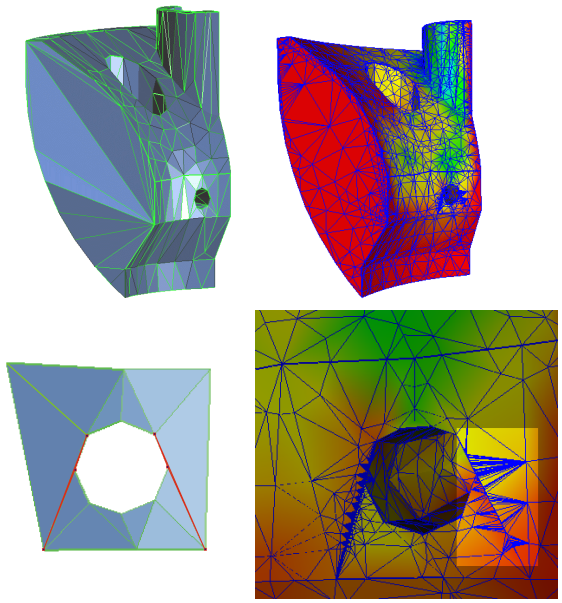

Figure 12: Top-left: Simplified boundary of spx; Bottom-left: detail of simplified boundary (two denegerated triangles are shown in red); Top-right: Steiner points added by CDT; Bottom-right: too many Steiner points added on degenerated triangles.

We need to stress that there is an interplay between the quality of the simplified surface that we get back from GTS and the speed and quality of the final CDT. We noticed that artifacts in the simplified surface, including degenerate triangles (skinny, small, or otherwise malformed triangles), cause the CDT to misbehave and sometimes produce unexpected results. See Figure 12. The shape of the triangles greatly affects the number of added Steiner points, which in turn causes it to take longer to compute the CDT. Because of this dependency on the quality of the approximation, we actually spend a considerable amount of time evaluating the different surface simplification codes available, and in the end, we chose to modify GTS to improve the quality of the simplified models. Still, we believe there is work to be done to develop a definite solution to this problem. Part of the problem is that surface simplification codes are developed with a goal towards maintaining visual acuity instead of true geometric and topological quality and strict error bounds. Also, current simplification codes do not allow us to take into account the scalar field defined at vertices, forcing us into using indirect parameters (e.g., number of edges) when trying to achieve a given scalar field error bound.

The CDT is the most expensive part of the overall algorithm. One option to further improve simplification times would be to develop custom tetrahedralization code that maintains the boundary surface. Another potential use for this type of work is maintaining certain features of the original tetrahedralization such as directionality and/or shape of original cells. Our sampling procedure is quite fast, one of the reasons that it is not even faster is that as a result of boundary simplification some of the sampled interior points can be outside. In the same way, some of the ghost vertices can be inside. We used a *k-d* tree with 200 tetrahedra per node to speed-up the removal of external sample points and interior ghost vertices. Moreover, we used a minimum distance to avoid sample points to be on the surface. The threshold is set to 1/1000 the diagonal of the simplified boundary.

6 Conclusions and Future Work

In this paper we introduced a new technique for simplifying large unstructured meshes. The basic idea is to focus on the scalar field, and to simplify its domain and sample points separately, taking into account proper error bounds in each case. The simplified function is computed as a piecewise linear extension of the simplified sampled data inside the simplified domain. Given certain high-level pieces, our algorithm is relatively simple to implement. Our experimental results show the effectiveness of our technique both in terms of simplification quality and speed. In particular, our technique is one to two orders of magnitude faster than existing edge-collapse simplification codes.

There are many avenues for future work. A particularly important one for practical use of our approach (discussed in Section 5) is to develop a geometry and topology accurate surface simplification technique. We are also interested in developing an out-of-core version of our approach in order to simplify extremely large datasets along the lines of the work of [8]. Finally, we are very interested in exploring better feature-aware sampling techniques, including the possibility of preserving major topological features of the scalar field.

7 Acknowledgements

We thank Steven Callahan for useful criticisms on this work that helped to improve its content and presentation. We thank David Cohen-Steiner for the conforming Delaunay tetrahedralization code, the CGAL consortion, the authors of the GNU Triangulated Surface Library, Steven Callahan and Milan Ikits for the volume renderer used in our experiments, and Wagner Corrêa for help with the GTB library. The Teem toolkit [15] proved very useful for processing our datasets and results. The authors also acknowledge Bruno Notrosso (Electricite de France) for the spx dataset, Hung and Buning for the blunt fin dataset, and Neely and Batina for the fighter dataset. This work has been supported by DOE under the VIEWS program and the MICS office, and the National Science Foundation under grants CCF-0401498 and EIA-0323604.

References

[1] S. Callahan, M. Ikits, J. Comba, and C. Silva. Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering. In *SCI Institute Technical report UUSCI-2004-003*, 2004.

[2] H. Carr, J. Snoeyink, and U. Axen. Computing countour trees in all dimensions. In *Proceedings ACM-SIAM Symposium Discrete Algorithms*, pages 981–926, 2000.

[3] Y.-J. Chiang and X. Lu. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. In *Computer Graphics Forum, volume 22, number 3*, pages 493–504, 2003.

[4] P. Chopra and J. Meyer. TetFusion: an algorithm for rapid tetrahedral mesh simplification. In *Proceedings of the conference on Visualization '02*, pages 133–140. IEEE Computer Society, 2002.

[5] P. Cignoni, D. Constanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of Tetrahedral meshes with accurate error evaluation. In *Proceedings of the conference on Visualization '00*, pages 85–92. IEEE Computer Society Press, 2000.

[6] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification Envelopes. In *Proceedings of SIGGRAPH 96, In Computer Graphics Proceedings, Annual Conference Series*, pages 119–128. ACM SIGGRAPH, 1996.

[7] D. Cohen-Steiner, E. C. de Verdiere, and M. Yvinec. Conforming Delaunay triangulations in 3D. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 199–208. ACM Press, 2002.

[8] W. T. Corrêa. *New Techniques for Out-Of-Core Visualization of Large Datasets*. PhD thesis, Princeton University, 2004.

Dataset	Fighter			
Parameters:				
Boundary LOD (% of edges)	24%		8%	
Clustering variation (% of range)	1.2%		1.2%	
Level of detail:				
Num. of original tetrahedra	1,403K		1,403K	
Percentage of the original tetrahedra	25%		23%	
Times: (seconds)				
Extracting boundary	3.7	0.8%	3.6	3.4%
Simplifying boundary	17.4	3.6%	20.4	18.9%
Sampling points	19.5	4.1%	6.4	5.9%
Adding ghost vertices	4.9	1.0%	3.2	3.0%
Conforming Delaunay tetrahedralization	385.2	80.6%	39.0	36.2%
Removing external tetrahedron	18.5	3.9%	12.3	11.4%
Removing Steiner points with edge-collapses	4.6	1.0%	3.2	2.9%
Computing values of remaining Steiner points	17.4	3.6%	13.8	12.8%
Other	7.2	1.5%	5.9	5.4%
Total	7m58s		1m47s	

Table 2: Times of the steps of the simplification pipeline. The point sampling step includes the *k-d* tree clustering followed by the removal of the points that are outside the simplified boundary surface. The conforming Delaunay tetrahedralization is the bottleneck of the pipeline.

[9] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001.

[10] R. Farias, J. Mitchell, C. Silva, and B. Wylie. Time-Critical Rendering of Irregular Grids. In *Proceedings of SIBGRAPI*, pages 243–250, 2000.

[11] M. Garland and P. S. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 209–216, Aug. 1997.

[12] A. V. Gelder, V. Verna, and J. Wilhelms. Volume decimation of irregular tetrahedral grids. *Computer Graphics International*, pages 222–230, 1999.

[13] T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schussman, and I. J. Trotts. Constructing hierarchies for triangle meshes. In *IEEE Transactions on Visualization and Computer Graphics 4*, pages 145–161, 1998.

[14] H. Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 99–108, Aug. 1996.

[15] G. L. Kindlmann. Teem, 2003. `http://teem.sourceforge.net`.

[16] P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *Proceedings of the conference on Visualization '98*, pages 279–286. IEEE Visualization, 1998.

[17] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21 (4), pages 163–169, jul 1987.

[18] N. Max, P. Hanrahan, and R. Crawfis. Area and volume coherence for efficient visualization of 3D scalar functions. In *Proceedings of the 1990 workshop on Volume visualization*, pages 27–33. ACM Press, 1990.

[19] M. Murphy, D. M. Mount, and C. W. Gable. A Point-placement strategy for Conforming Delaunay tetrahedralization. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 67–74, 2000.

[20] V. Natarajan and H. Edelsbrunner. Simplification of Three-Dimensional Density Maps. In D. S. Ebert, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 10 (5), pages 587–597. IEEE Computer Society, 2004.

[21] S. Popinet. GNU Triangulated Surface Library - Release 0.7.1, 2003. `http://gts.sourceforge.net`.

[22] M. ROY. Mesh Comparison Software, 2002. `http://meshdev.sourceforge.net`.

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

[24] I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of Tetrahedral Meshes with Error Bounds. In H. Hagen, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 5 (3), pages 224–237. IEEE Computer Society, 1999.

[25] I. J. Trotts, B. Hamann, K. I. Joy, and D. F. Wiley. Simplification of tetrahedral meshes. In *IEEE Visualization '98 (VIS '98)*, pages 287–295, Washington - Brussels - Tokyo, Oct. 1998. IEEE.