Volume Graphics (2005)
E. Gröller, I. Fujishiro (Editors)

Simplification of Unstructured Tetrahedral Meshes by Point Sampling

Dirce Uesu Louis Bavoil Shachar Fleishman Jason Shepherd Cláudio T. Silva †

Scientific Computing and Imaging Institute, University of Utah

Abstract

Tetrahedral meshes are widely used in scientific computing for representing three-dimensional scalar, vector, and tensor fields. The size and complexity of some of these meshes can 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 an underlying scalar field directly 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 is much faster than edge-collapse-based simplification approaches. Furthermore, it is particularly suitable for aggressive simplification. Experiments show that isosurfaces and volume renderings of meshes produced by our technique have few noticeable visual artifacts.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - surface and object representations, geometric algorithms

Keywords: mesh simplification, unstructured meshes, level-of-detail, point sampling

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. It is important to distinguish the domain D from the scalar field f. The purpose of visualization techniques, such as isosurface generation [LC87] and direct volume rendering [MHC90] is 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.*, $|\bar{f} - f| \leq \varepsilon$, and 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.*, [CM02, NE04, THJW98]) on the domain D. These techniques are similar to triangle-based simplification techniques [Hop96, GH97] and use connectivity information to incrementally cull simplices c_i from the domain. (*i.e.*, when a 1-simplex 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 $|\bar{D}| \leq n$) or when the function $\bar{f}$ reaches a maximum user-defined error bound ε.

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

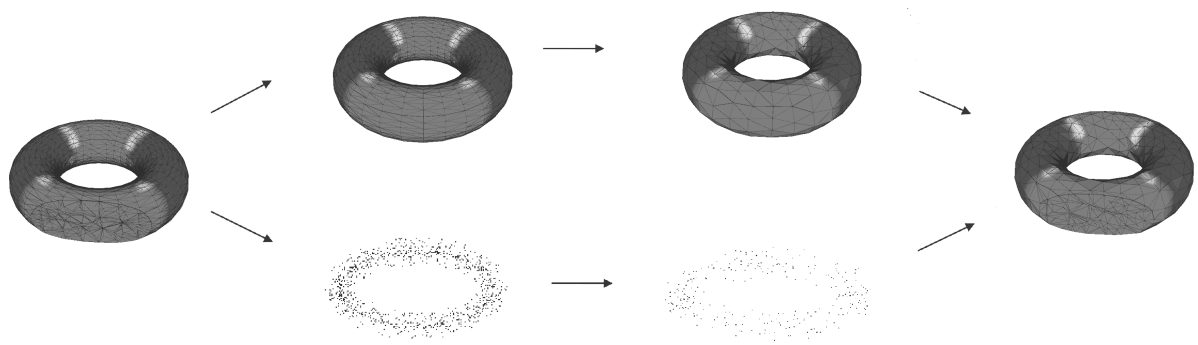

Figure 1: *Graphical representation of our simplification pipeline. Our technique separates the original scalar field (shown on the left) into two pieces: the boundary of the original domain and the interior samples of the scalar field. We then simplify each piece separately, according to a target number of boundary edges, and a local error bound on the interior samples. 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.*

With surfaces embedded in three-dimensions, it is 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 as important. The domain is used to represent the subset of $\mathbb{R}^3$ where the scalar field f is defined. For this reason, we only need to maintain the shape and topology of the boundary surface ∂D. In our work, instead of slowly building $\bar{D}$ from D using a series of collapses, we build $\bar{B}$, the boundary of $\bar{D}$, by simplifying ∂D, while completely ignoring the connectivity of the interior. For the interior, we use a 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 (re)tetrahedralization approach 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.

Edge-collapse-based approaches work well but they are intrinsically limited in speed by their top-down approach. For example, to simplify a mesh to 10% of its original number of edges, it is necessary to collapse 90% of the original edges. Our method is much faster in this case because it uses point sampling instead of edge collapses.

Our new algorithm builds on our previous work [FMSW00]. 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 the details of our simplification algorithm. Section 4 presents our experimental 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

2.1. Triangle mesh simplification

Most existing triangle mesh simplification algorithms use edge collapses. Garland and Heckbert [GH97] introduced the quadric error metric which can be used to optimize the position of the vertices resulting from edge collapses such that the surface is locally preserved. Their method uses iterative contractions on vertex pairs and calculates the error approximations using quadric matrices. This metric measures the squared (geometric and field) distances from points to hyperplanes spanned by triangles. The boundaries are preserved using a similar metric on the boundary edges and by weighting boundary and interior errors appropriately. The quadric error of collapsing an edge to a single point is expressed as the sum of squared distances to all accumulated incident hyperplanes, and can be encoded efficiently for n-D vertices as a symmetric $n \times n$ matrix $\mathbf{A}$, an n-vector $\mathbf{b}$, and a scalar c. Lindstrom and Turk [LT98] used a different formulation of the volume-preservation error metric, and added a constraint on the shape of the triangles. Their algorithm is implemented by the GNU Triangulated Surface (GTS) [Pop03] open-source library.

A different approach was taken by Cohen *et al.* with Simplification Envelopes [CVM*96], which guarantees a maximum distance between the simplified surface and the original. However, edge-collapse codes are faster.

2.2. Edge-collapse-based tetrahedra mesh simplification

Edge-collapse algorithms employ various edge-cost functions, which may be different functions for boundary edges and interior edges. In their most recent work Garland and

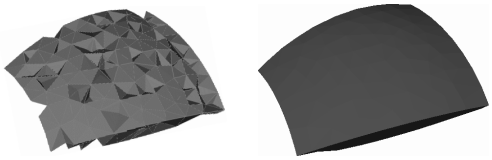

Figure 2: *Comparison with [FMSW00]: The result of Farias et al. for the SPX dataset is shown on the left; observe the damaged boundary. The result of the algorithm proposed in this paper is shown on the right.*

Zhou [GZ05] present an iterative edge contraction for simplicial complex of any dimension. They propose a generalized error metric using tangent vectors instead of perpendicular normals [GH97]. They add a boundary penalty factor to preserve the geometry of the boundary. Their system is very general, and it supports the simplification of data of any topological type, including non-manifold complexes and mixed complexes.

The tetrahedral mesh simplification of Natarajan and Edelsbrunner [NE04] is based on edge contractions with the modified quadratic error metric of Garland and Heckbert [GH97]. 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.

Trotts *et al.* [THJW98] extend the technique of Gieng *et al.* [GHJ*98] for the simplification of triangle meshes to tetrahedral meshes. They collapse a given tetrahedron by successively collapsing three of its edges. Their technique attempts to preserve the boundary of the tetrahedral mesh by using boundary constraints. In their followup work, Trotts *et al.* [THJ99] improve upon their previous methods by avoiding the topological problems created from collapsing a tetrahedron. Their strategy is to use only one edge collapse instead of a sequence of three edge collapses.

Gelder *et al.* [GVW99] evaluate the effect of decimation by comparing two data-based error metrics: a mass metric and a density metric.

Staadt and Gross [SG98] extend Hoppe's work [Hop96] for progressive tetrahedralization. They discuss intersections, inversions, and degenerations of tetrahedra for a robust edge-collapse implementation. They redefine the cost function by considering the volume preservation and gradient.

Chiang and Lu [CL03] construct multiple levels of detail of a tetrahedral volume, preserving the topology of all isosurfaces. Their algorithm simplifies the tetrahedral mesh in two phases. In the segmentation phase, it classifies each vertex into critical and non-critical points and identifies topologically-equivalent regions. In the simplification phase, the algorithm uses edge-collapse operations in which each topologically-equivalent region is simplified independently. Notice that any change to the transfer function requires a fresh run of the simplification algorithm.

2.3. Other tetrahedra mesh simplification techniques

Chopra and Meyer [CM02] propose a fast algorithm for progressive simplification named *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 keeping all the tetrahedra on the boundary.

Farias *et al.* [FMSW00] 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 subset of the interior points is generated. Second, the boundary is simplified separately, leading to a collection of simplified points. From the simplified boundary, a set of external *ghost* vertices is generated. Then the geometry of the boundary is discarded (unlike in 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. Because the geometry of the boundary is discarded, the algorithm employed by Farias *et al.* could suffer from severe boundary anomalies. (See Figure 2.)

2.4. Tetrahedral mesh generation

The most widely used tetrahedral meshing algorithms are based on the Delaunay criterion [Owe98]. Delaunay triangulations and Delaunay tetrahedralizations (DT) are very well known and studied mesh entities (see, e.g., [Ede01, Chapter 5]). A basic property that characterizes this geometric structure is the fact 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 within the tetrahedralization lies inside the circumsphere. Under some non-degeneracy conditions (no 5 points co-spherical), this property completely characterizes DTs and the DT is unique. Part of the appeal of Delaunay tetrahedralizations is the relative ease of computing the tetrahedralizations. As a well-studied structure, often used in mesh generation, standard codes are readily available that compute the DT. The practical need of forcing certain faces to be part of the tetrahedralizations led to the development of *conforming* Delaunay tetrahedralizations (CDT) [She99].

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 added points are 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 sets are well known and reliable code exists,

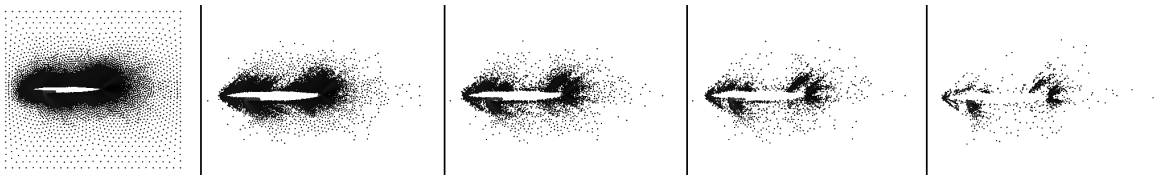

Figure 3: *One slice of sampled points for* `fighter` *LODs highlighting our adaptive sampling technique.*

conforming DT algorithms are still in active development [MMG00, CSdVY02]. The particular technique for adding Steiner points affects the termination of the algorithm, and also the quantity and quality of the added geometry.

Borouchaki *et al.* [BHSG95] introduced a robust mesh generation algorithm that tetrahedralizes a set of points with face constraints. This algorithm allows the user to specify a bound on the shape of the tetrahedra, and may insert additional points inside the domain to meet the shape constraints specified by the user. Their method is incremental. It takes as input a boundary mesh and a set of interior points. It first creates a DT based on the points of the boundary and the interior samples. Then mesh transformations are applied to try to make the boundary mesh conform to the input boundary mesh. When these transformations fail, Steiner points are inserted in the interior of the mesh. A solid implementation of this method (GHS-3D 3.1, described in [GHS91]) is included with the CAMAL library [San04].

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 geometry of the tetrahedral mesh. We present a multi-stage algorithm (see Figure 1). 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. We sample the interior points using a *k-d* tree partition of the interior points of the mesh and we remove the samples that are outside the boundary, or closer than a certain minimum distance to the boundary. Then, we reconstruct a simplified tetrahedral mesh by using a conforming Delaunay tetrahedralization (CDT) on the interior points while taking into consideration the simplified boundary. At this point, we have a mesh with no associated scalar values, which approximates D in a subset and has a boundary surface which conforms to the simplified boundary surface. The CDT introduces new points for which we do not have scalar values in the original mesh. Our final step is to compute values at these points using the function f. These steps are further detailed below.

3.1. Sampling the interior points

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. Since we know that a function reconstructed from the simplified model is a linear interpolation, we will not consider higher-order interpolation techniques such as a 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, grouping points with similar scalar values together. The final level of detail is obtained by sampling each leaf of the tree by the point with the value closest to the mean scalar value of the cell. (See Figure 3.)

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. 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 axis at the middle of the vector $\mathbf{g}$ (defined above).

As shown in Figure 4, picking a single point in a subdivided cell can introduce a significant error for large cells. However, this usually does not introduce significant visualization artifacts as demonstrated by our results.

3.2. Simplifying the boundary mesh

The boundary surface is defined by the faces of the tetrahedral mesh that belong to a single tetrahedron. The algorithm to compute the boundary is quite simple. We traverse all the faces of the mesh. For each face, we order its vertices consistently so that if two tetrahedra share a face, the two instances of that faces have the same vertices in the same order. The boundary surface is the set of faces that are inserted only

once, and the interior points are the vertices that do not belong to the boundary.

The boundary 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 behavior of the scalar function on the boundary. Therefore, pure geometric simplification algorithms like the *Simplification Envelopes* technique [CVM*96], which creates large triangles on flat surfaces, are not appropriate.

In addition, the conforming Delaunay tetrahedralization algorithm 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 [CSdVY02]. Therefore, we also try to generate simplified boundary meshes with equilateral triangles.

Instead of developing a mesh simplication code from the ground up, we decided to modify the `coarsen` program provided by the GTS library [Pop03], which is based on edge collapses. The potential edges to be collapsed are inserted into a priority queue. Each edge is associated with a cost, 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 [LT98]. As with other edge-collapse based simplification algorithms, `coarsen` is characterized by the placement algorithm of the new vertices resulting from a collapse, and by the cost function $f_c(e,v)$, where e is an edge and v is a possible result of the collapse of e.

The position of a new vertex can be computed by using the different methods as described in [LT98]: 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. In our work, we use the volume preservation constraint, composed with the coupled volume and shape optimization. Finally, the shape optimization constraint is applied if necessary. The details follow.

The weight of the volume optimization is fixed to one, and the weight of the shape optimization for an edge e, $w_s(e)$, is

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

where $L(e)$ is the length of edge e, and λ is a parameter

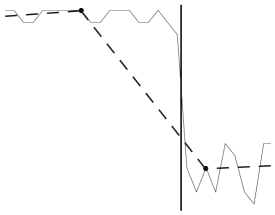

Figure 4: *An example of splitting a cell that only contains coplanar points, with the x-axis as the splitting direction. The plain line is the original scalar field, the dash line is the approximation from sampling one point on each side of the split.*

which we will refer to later as the *shape optimization weight*. The purpose of λ is to weight the shape optimization relative to the volumetric optimization when placing new vertices. Increasing λ tends to create triangles of better quality, but with a larger geometric distance (Hausdorff) between the simplified and original surfaces.

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 [LT98]. 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. Removing sample points

One consequence of boundary simplification is that points which were inside the boundary of the original mesh may be outside the boundary of the simplified mesh. These points have to be removed from the set of sample points before reconstructing a mesh. To do this efficiently, we build a *k-d* tree of the triangles of the boundary mesh with a fixed maximum number of triangles per leaf. For each point, we find the leaf which contains the point and compute the distance to the nearest triangle in this leaf. Based on this distance, the points that lie outside the boundary are discarded.

The CDT implementation that we are using only works when interior points are far enough from the boundary, otherwise the CDT fails to produce a mesh conforming to the faces of the boundary. Therefore, we also need to discard points that are closer to the boundary than a minimum distance defined as a percentage of the diameter of the bounding box of the simplified boundary.

3.4. Tetrahedral mesh reconstruction

At this point, we have shown how to simplify the interior points and the domain boundary. Now, we need to construct a simplified, unstructured tetrahedral grid for further processing and visualization of the model, which incorporates the simplifed interior points and domain boundary.

Since this problem is essentially equivalent to normal unstructured tetrahedral mesh generation [Owe98], many options are available to accomplish this task. Ideally, our chosen technique should preserve the boundary and generate as few additional (i.e. Steiner) points to our sample points in the domain, as possible. If any additional points are introduced during this mesh generation phase, we will compute scalar values for these points, as a post-processing step, using the interpolation scheme described in the next section.

As discussed in Section 2.4, the most widely used tetrahedral meshing algorithms are based on Delaunay triangulations. They are conceptually simple, and existing robust codes are widely available. Also, incremental DT algorithms are an efficient way of creating a DT by inserting points one-at-a-time. We have thus decided to use a conforming DT as our meshing scheme.

Our implementation uses the CDT implemented in GHS-3D [BHSG95, GHS91] through the CAMAL interface [San04], which is a robust and efficient implementation of incremental DT, taking as input a boundary mesh and a set of interior points. It first creates a DT based on the points of the boundary and the interior samples. Then mesh transformations are applied to try to make the boundary mesh conform to the input boundary mesh. When these transformations fail, Steiner points are inserted in the interior of the mesh. Since the Delaunay tetrahedralization of non-convex objects is accomplished on the interior of the convex hull of the domain points, it is possible for tetrahedra to be generated outside the boundary of the original domain. These additional tetrahedra should be removed. This is performed intrinsically within the GHS-3D mesh generator.

3.5. Field reconstruction

We now have a mesh that conforms to the interior samples and to the boundary mesh, we need to set scalar values at each vertex of the mesh. All the original points have scalar values. However, for the Steiner points, we need to compute a value from the function of the original field. The value at a Steiner point $\mathbf{p}$ is computed by a linear interpolation of the scalar values at the vertices of the tetrahedron of the original mesh in which it was picked, using barycentric coordinates.

To find the value of $\mathbf{p}$ on the original mesh, it is first necessary to find a tetrahedron that contains $\mathbf{p}$ in this mesh. To do this efficiently, we build a *k-d* tree of the original mesh such that if $\mathbf{p}$ is in a tetrahedron, then this tetrahedron is in the cell of the *k-d* tree containing $\mathbf{p}$. We use a fixed maximum number of tetrahedra per leaf to build the tree.

Then, to get the value of the scalar field, we find the cell containing $\mathbf{p}$ and for every tetrahedron t of this cell, we check if $\mathbf{p}$ 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.

3.6. Summary

Our multi-stage simplification algorithm consists of the following steps: (1) separate the interior points from the boundary; (2) interior *point* simplification; (3) boundary *mesh* simplification; (4) unstructured mesh reconstruction; (5) reconstruction of scalar field on Steiner vertices. It is possible to implement our technique in different ways, but depending on the underlying algorithms used for steps (2), (3), and (4), further steps might be necessary. In fact, our earlier implementation of the algorithm, described in [UBFS04], was more involved, slower, and less reliable, due to a less robust mesh reconstruction in step (4).

Our algorithm has a set of parameters that roughly corresponds to the different steps. We need an error bound for the point sampling ρ; the target number of edges for the surface simplification σ; the shape optimization weight for the surface simplification; and minimum distance between sample points and the simplified boundary.

In theory, the only parameters required from the user are ρ and σ. It is possible to eliminate the other parameters by making improvements to our underlying algorithms used for steps (2) and (4), and by automating the choice of maximimum leaf size in the k-d tree to optimize for efficiency and/or memory consumption.

4. Results

We have implemented the algorithms described in Section 3 in C++. For surface simplification, we use the one available in GTS [Pop03].

The mesh reconstruction algorithm is performed using CAMAL 2.0.2, which in turn utilizes the GHS-3D 3.1 tetrahedral mesh generator. The CAMAL interface takes as input a list of boundary faces and interior nodes. The output is a tetrahedral mesh conforming to the original boundary, with all elements lying interior to the boundary. It does not maintain the scalar field, but we use the fact that the indices of the input and output vertices match to avoid performing an expensive matching phase.

To extract the boundary surface and the interior vertices, we use the class `Indexed_tetra_set` provided by the GTB library [CBF*04]. Our implementation of point-triangle distance needed in the closest triangle computation (Section 3.3) is based on code adapted from the Eberly's *Wild Magic* library [Ebe04]. Despite using multiple packages to implement our algorithm, everything is integrated and easy to use. In particular, it is possible to reuse the simplification of the original boundary mesh to quickly generate low-resolution levels of detail of an input mesh.

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. In testing, we

utilized a Pentium 4 at 3.20 GHz with 2 GB of RAM and 1 MB of CPU cache to generate our results.

Comparison with an edge-collapse-based technique. We compare our technique to the implementation of Garland and Zhou [GZ05] described in [VCL*05]. As shown by Table 1, our method is considerably faster than their technique for aggressive simplifications. Besides, low-resolution simplifications keep the features of the original dataset as demonstrated by Figures 6 and 9. Using histograms, shown in Figure 5, we can see that our technique preserves the global structure of the scalar field. The RMS errors of our technique are worse than those for the [GZ05] technique. Figure 7 shows the relative impact of a larger RMS error in our technique compared to Garland and Zhou's method [GZ05].

The `delta` dataset has some degenerate edges on its boundary. This led us to implement an optional edge-cleanup step between the boundary extraction and the boundary simplification steps. The `delta` has been the only dataset that we needed to use this extra step.

Our technique has another advantage, which is that it does not have to check for volume flips. The volume flip check of Garland and Zhou's [GZ05] is unstable when checking skinny tetrahedra with volume almost equal to zero. Therefore, [VCL*05] cannot reduce the `delta` dataset to 6%. In order to reach this value, we needed to disable the volume-flip checks, which introduced visible artifacts, see Figure 8.

Analysis of the pipeline. As shown in Table 2, we can simplify the `fighter` dataset, which has over 1.4 million tetrahedra, to about 12% of the original number of tetrahedra in about 36s, using a maximum of 400 MB. We also notice that the complexity of the CDT increases with the number of interior point samples. In addition, we see that more than half of the sampled vertices are discarded because they are either outside the simplified boundary, or closer than the specified uniform minimum distance. The CDT step adds Steiner points to be able to conform to the boundary faces, which are the difference between the selected samples and the final vertices in Table 2. We set the minimum distances to the boundary surface to 1% of the diameter of the bounding box of the simplified boundary mesh. For datasets such as the `fighter` dataset which have a large density of points close to the surface, this can imply discarding about 70% of sample points but it apparently does not affect the quality of the simplifications.

Field error estimation. To estimate the field error in Table 1 of our results, we use a similar approach of Cignoni *et al.* [CCM*00]. However, we do not only sample error at vertices but we also take a uniform number of samples inside each tetrahedron. Another difference is that we ignore the samples that are outside the original boundary. Given a reference tetrahedral mesh 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.

Dataset (LOD)	Edge collapses		Our technique	
	Time	RMS	Time	RMS
`fighter` (6%)	72.09s	0.26%	26.74s	0.55%
`fighter` (22%)	62.06s	0.11%	32.46s	0.37%
`fighter` (48%)	46.33s	0.03%	38.43s	0.29%
`f117` (14%)	6.49s	0.15%	3.98s	0.79%
`bluntfin` (9%)	6.28s	0.52%	3.71s	1.86%
`delta` (4%)	33.37s	1.69%	20.53s	2.12%

Table 1: *Comparison of the simplification time of a fast edge-collapse implementation of Garland and Zhou's approach [GZ05] with the overall time of our approach, for the `fighter` (1.4M tets), `delta` (1M tets), `f117` (240K tets), and `blunt fin` (187K tets). The times do not include reading and writing to files, nor building face connectivities. The levels of detail (LOD) are percentages of the original number of tetrahedra. The edge-collapse code uses Cholesky's method with substituted solution for singular matrices to solve linear systems. Both implementations were tested on the same Pentium 4 computer at 3.2 Ghz. The RMS error is the Root-Mean-Square error relative to the range of scalar values.*

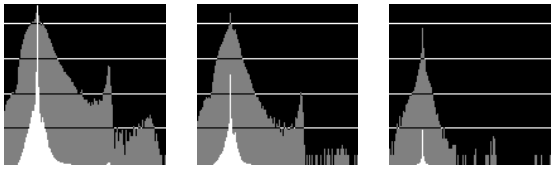

Figure 5: *Histograms of the scalar values of LODs of the `fighter` dataset. 1.4M tets (100%); 669K (48%); 90K tets (6%). The white graph is linearly scaled histogram, the gray graph is log-scaled histogram, and the horizontal lines are the decades on the log scale.*

Contrary to Cignoni's approach [CCM*00], when a sample taken in the domain of M_1 is outside of the domain of M_2, we ignore that point, since the associated value is undefined in M_2. In practice, this usually leads to ignore less than 10% of the samples.

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 methods. From the 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 advantage of our approach is the overall improved speed for aggressive simplifications. 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 [CL03]. 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.

The use of a conforming Delaunay tetrahedralization provides a powerful method allowing us to preserve the boundary of the domain, and greatly simplifies the implementation of our technique. However, it should be noted that there is an interplay between the quality of the simplified surface and the time to generate the final CDT. 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, or to return an error. There is work to be done to develop a better surface simplification technique fitting our needs. Part of the problem is that surface simplification codes are developed with a goal toward maintaining visual acuity instead of true geometric and topological quality and strict error bounds. Also, currently-available 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.

6. Conclusions and future work

In this paper we introduced a new technique for simplifying large unstructured meshes. It is an example of the emerging impact of point-based techniques on visualization [LM99, CPJ04]. 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 scalar field is computed as a piecewise linear extension of the simplified sampled data inside the simplified domain. One of the nice features of our technique is that it builds on existing techniques for which robust solutions (and libraries) are available. Given these libraries, 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 faster than existing edge-collapse simplification codes for aggressive simplifications.

There are many avenues for future work. An interesting one is the development of a geometry and topology accurate surface simplification technique. Current techniques sometimes generate non-manifold triangulation with bad triangles. We are also interested in developing an out-of-core version of our approach in order to simplify extremely large datasets. Finally, we are very interested in exploring better feature-aware sampling techniques, including the possibility of preserving major topological features of the scalar field.

Dataset	fighter	delta	delta	f117
Parameters:				
Boundary LOD (% of edges)	10%	5%	5%	10%
Clustering (% of range)	1%	1%	10%	0.5%
Level of detail:				
Num. of original tetrahedra	1.4M	1.0M	1.0M	0.24M
Percentage of the original tets	12%	17%	4%	14%
Vertices:				
Original vertices	215K	191K	191K	44K
Selected samples	18K	26K	4K	5K
Final vertices	32K	29K	7K	6K
Steiner points	82%	9%	56%	18%
Memory:				
Total size (MB)	399	277	265	72
Times: (seconds)				
Reading mesh	7.14	4.50	4.64	1.22
Extr. boundary	0.64	1.98	2.09	0.07
Simp. boundary	16.59	7.55	7.60	2.00
Sampling points	0.65	0.37	0.23	0.10
Rem. samples	1.33	0.72	0.58	0.60
CDT	2.05	28.66	3.63	0.95
Steiner scalar field	7.11	8.63	12.94	0.53
Other	0.52	0.69	0.72	0.06
Total (real)	36.03s	53.10s	32.43s	5.53s

Table 2: *Times of the steps of the simplification pipeline. For all the datasets, we use a shape optimization weight of 0.1 (Section 3.2), and a minimum distance to boundary of 1% of the diameter of the bounding box of the simplified boundary (Section 3.3). We use a maximum of 200 triangles per leaf for the boundary k-d tree used to remove sample points, and a maximum of 500 tetrahedra per leaf for the k-d tree used to compute scalar values of Steiner points. The times were taken on a Linux 2.4.20 system with empty disk cache.*

7. Acknowledgements

We thank Steven Callahan, Joe Kniss, and John Schreiner for help with the paper; Sandia National Laboratories for the CAMAL library; the authors of the GNU Triangulated Surface Library; Steven Callahan and Milan Ikits for their volume renderer; Wagner Corrêa for help with the GTB library; and Huy T. Vo for the implementation of the fast edge-collapse based simplification code. We used SCIRun for computing and rendering isosurfaces, and the Teem tookit to generate our histograms. The authors also acknowledge Bruno Notrosso (Electricite de France) for the spx dataset, Hung and Buning (NASA) for the blunt fin dataset, Neely and Batina (NASA) for the fighter dataset, and NASA Ames Research Center for the Delta Wing dataset. Louis Bavoil is supported by the Department of Energy (DOE) under the VIEWS program. Cláudio T. Silva is partially supported by the DOE under the VIEWS program and the MICS office, the National Science Foundation under grants CCF-0401498, EIA-0323604, and OISE-0405402, and a University of Utah Seed Grant.

References

[BHSG95] BOROUCHAKI H., HECHT F., SALTEL E., GEORGE P. L.: Reasonably efficient Delaunay based mesh generator in three dimensions. In *the 4th International Meshing Roundtable* (1995), pp. 3–14. 4, 6

[CBF*04] CORRÊA W., BAVOIL L., FLEISHMAN S., JIMENEZ W., KLOSOWSKI J., MARTINS G., PESCO S., ROCHA L., SCHEIDEGGER C., SILVA C., UESU D.: Graphics Tool Box (GTB), 2004. http://sourceforge.net/projects/gtb. 6

[CCM*00] CIGNONI P., CONSTANZA D., MONTANI C., ROCCHINI C., SCOPIGNO R.: Simplification of tetrahedral meshes with accurate error evaluation. In *IEEE Visualization 2000* (2000), pp. 85–92. 7

[CL03] CHIANG Y.-J., LU X.: Progressive simplification of tetrahedral meshes preserving all isosurface topologies. *Computer Graphics Forum 22*, 3 (2003), 493–504. 3, 7

[CM02] CHOPRA P., MEYER J.: Tetfusion: an algorithm for rapid tetrahedral mesh simplification. In *IEEE Visualization 2002* (2002), pp. 133–140. 1, 3

[CPJ04] CO C., PORUMBESCU S., JOY K. I.: Meshless isosurface generation from multiblock data. In *VisSym 2004* (2004), pp. 273–281. 8

[CSdVY02] COHEN-STEINER D., DE VERDIERE E. C., YVINEC M.: Conforming Delaunay triangulations in 3D. In *the 18th Annual Symposium on Computational Geometry* (2002), pp. 199–208. 4, 5

[CVM*96] COHEN J., VARSHNEY A., MANOCHA D., TURK G., WEBER H., AGARWAL P., BROOKS F., WRIGHT W.: Simplification envelopes. In *SIGGRAPH 1996* (1996), pp. 119–128. 2, 5

[Ebe04] EBERLY D. H.: *3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic*. Morgan Kaufmann Publishers, 2004. 6

[Ede01] EDELSBRUNNER H.: *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001. 3

[FMSW00] FARIAS R., MITCHELL J., SILVA C., WYLIE B.: Time-Critical Rendering of Irregular Grids. In *the Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI) 2000* (2000), pp. 243–250. 2, 3

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *SIGGRAPH 1997* (1997), pp. 209–216. 1, 2, 3

[GHJ*98] GIENG T. S., HAMANN B., JOY K. I., SCHUSSMAN G. L., TROTTS I. J.: Constructing hierarchies for triangle meshes. In *IEEE Visualization 1998* (1998), pp. 145–161. 3

[GHS91] GEORGE P. L., HECHT F., SALTEL E.: Automatic mesh generator with specified boundary. *Comput. Methods Appl. Mech. Eng. 92*, 3 (1991), 269–288. 4, 6

[GVW99] GELDER A. V., VERNA V., WILHELMS J.: Volume decimation of irregular tetrahedral grids. *Computer Graphics International 1999* (1999), 222–230. 3

[GZ05] GARLAND M., ZHOU Y.: Quadric-based simplification in any dimension. *ACM Transactions on Graphics 24*, 2 (Apr. 2005). 3, 7

[Hop96] HOPPE H.: Progressive meshes. In *SIGGRAPH 1996* (1996), pp. 99–108. 1, 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH 1987* (1987), pp. 163–169. 1

[LM99] LEUTENEGGER S., MA K.-L.: R-tree retrieval of unstructured volume data for visualization. In *DIMACS Series of Discrete Masthematics and Theoretical Computer Science* (May 1999), vol. 50, pp. 279–291. 8

[LT98] LINDSTROM P., TURK G.: Fast and memory efficient polygonal simplification. In *IEEE Visualization 1998* (1998), pp. 279–286. 2, 5

[MHC90] MAX N., HANRAHAN P., CRAWFIS R.: Area and volume coherence for efficient visualization of 3D scalar functions. *ACM SIGGRAPH Comput. Graph. 24*, 5 (1990), 27–33. 1

[MMG00] MURPHY M., MOUNT D. M., GABLE. C. W.: A point-placement strategy for conforming Delaunay tetrahedralization. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms* (2000), pp. 67–74. 4

[NE04] NATARAJAN V., EDELSBRUNNER H.: Simplification of three-dimensional density maps. *IEEE Transactions on Visualization and Computer Graphics 10*, 5 (2004), 587–597. 1, 3

[Owe98] OWEN S. J.: A Survey of Unstructured Mesh Generation Technology. In *7th International Meshing Roundtable* (1998). 3, 6

[Pop03] POPINET S.: GNU Triangulated Surface (GTS) Library, Release 0.7.1, 2003. http://gts.sourceforge.net. 2, 5, 6

[San04] SANDIA NATIONAL LABORATORIES: The CUBIT adaptive meshing algorithm library (CAMAL), release 2.0.2, 2004. http://cubit.sandia.gov. 4, 6

[SG98] STAADT O. G., GROSS M. H.: Progressive tetrahedralizations. In *IEEE Visualization 1998* (1998), pp. 397–402. 3

[She99] SHEWCHUK J. R.: *Lecture Notes on Delaunay Mesh Generation*. Tech. rep., University of California at Berkeley, 1999. 3

[THJ99] TROTTS I. J., HAMANN B., JOY K. I.: Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 224–237. 3

[THJW98] TROTTS I. J., HAMANN B., JOY K. I., WILEY D. F.: Simplification of tetrahedral meshes. In *IEEE Visualization 1998* (1998), pp. 287–295. 1, 3

[UBFS04] UESU D., BAVOIL L., FLEISHMAN S., SILVA C. T.: *Simplification of Unstructured Tetrahedral Meshes by Point-Sampling*. SCI Institute Technical Report UUSCI-2004-005, University of Utah, 2004. 6

[VCL*05] VO H. T., CALLAHAN S. P., LINDSTROM P., PASCUCCI V., SILVA C. T.: *Streaming Simplification of Tetrahedral Meshes*. Tech. rep., University of Utah, 2005. 7

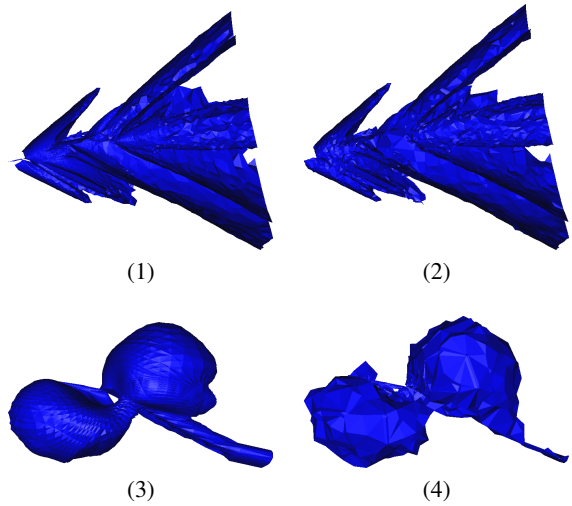

Figure 6: *First row: Isosurface of LODs of the* `fighter` *dataset. (1) 1.4M tets (100%); (2) 90K tets (6%). Second row: Isosurface of LODs of the* `delta` *dataset. (3) 1M tets (100%); (4) 59K tets (6%).*

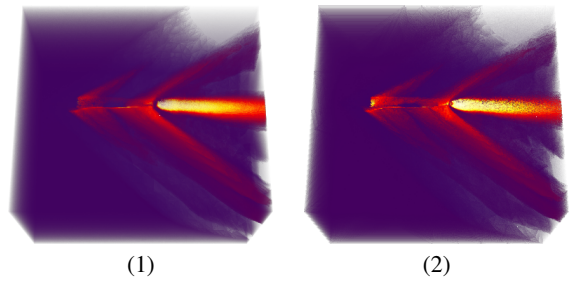

Figure 7: *Volume rendering of the* `fighter` *dataset reduced to 6%. (1) with our technique; (2) with Garland and Zhou's method.*

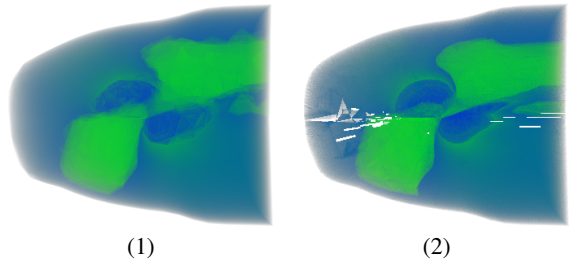

Figure 8: *Volume rendering of the* `delta` *dataset reduced to 6%. (1) with our technique; (2) with Garland and Zhou's method without volume-flip checks.*

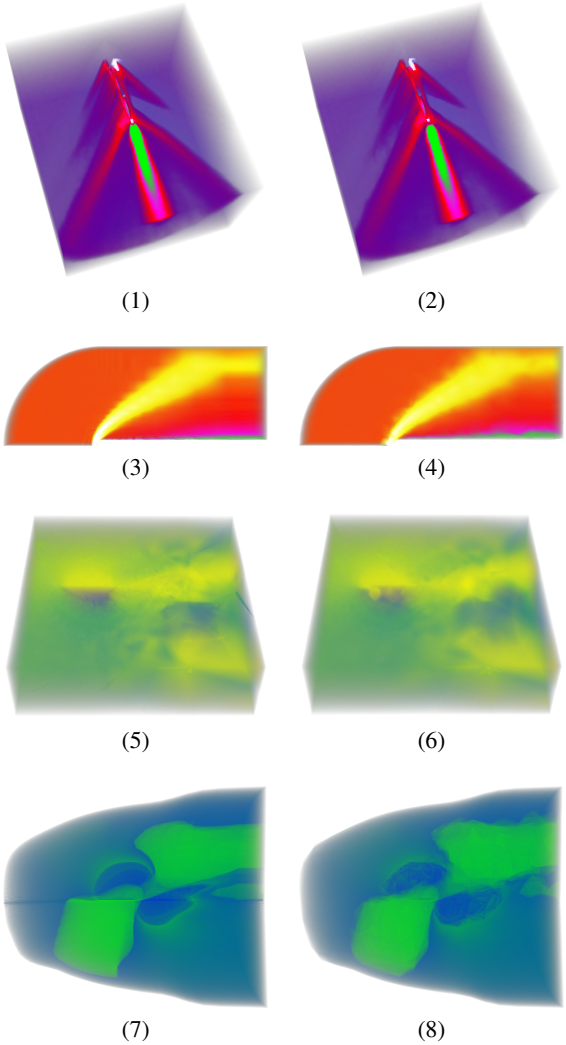

Figure 9: ***First row:*** *Volume rendering of the* `fighter` *dataset. (1) 1.4M tets (100%), (2) 90K tets (6%);* ***Second row:*** *Volume rendering of the* `blunt fin` *dataset. (3) 187K tets (100%), (4) 18K tets (10%);* ***Third row:*** *Volume rendering of the* `f117` *dataset. (5) 240K tets (100%), (6) 7K tets (3%).* ***Fourth row:*** *Volume rendering of the* `delta` *dataset. (7) 1M tets (100%), (8) 59K tets (6%).*