UC Davis
IDAV Publications

Title
Simplification of Tetrahedral Meshes

Permalink
https://escholarship.org/uc/item/868157fr

Authors
Trotts, Isaac J.
Hamann, Bernd
Joy, Ken
et al.

Publication Date
1998

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Simplification of Tetrahedral Meshes

Issac J. Trotts*
Bernd Hamann
Kenneth I. Joy
David F. Wiley

Center for Image Processing and Integrated Computing
Department of Computer Science
University of California, Davis 95616-8562

Abstract

We present a method for the construction of multiple levels of tetrahedral meshes approximating a trivariate function at different levels of detail. Starting with an initial, high-resolution triangulation of a three-dimensional region, we construct coarser representation levels by collapsing tetrahedra. Each triangulation defines a linear spline function, where the function values associated with the vertices are the spline coefficients. Based on predicted errors, we collapse tetrahedron in the grid that do not cause the maximum error to exceed a use-specified threshold. Bounds are stored for individual tetrahedra and are updated as the mesh is simplified. We continue the simplification process until a certain error is reached. The result is a hierarchical data description suited for the efficient visualization of large data sets at varying levels of detail.

Keywords: Approximation; hierarchical representation; mesh generation; multiresolution method; scattered data; spline; triangulation; visualization.

*{trotts,hamann,joy,wiley}@cs.ucdavis.edu

0-8186-9176-x/98/$10.00 Copyright 1998 IEEE

1 INTRODUCTION

One of the most critical and fundamental research problems encountered in the analysis and visualization of massive data sets is the development of methods for storing, approximating, and rendering large volumes of data efficiently. The problem is to develop different representations of the data set, each of which can be substituted for the complete set depending on the requirements of the analysis or the visualization technique. The data set may be represented by a few points, or by several million if necessary, with each of the data sets capturing the features of the original data. A hierarchical representation (or *multiresolution* representation) allows the study of large-scale features by considering a small subset and the study of small-scale features by considering a large subset of a given scientific data set.

Most *scientific data sets* are multivalued, meaning that multiple dependent variables – *e.g.*, velocity, pressure, temperature, salinity, sound speed, chemical or nuclear contamination, or even entire "matrices" (tensors) – are associated with each grid point. The grids may represent a surface or a volume in space, and the underlying grid may belong to various grid types: it may be *structured*, where, in the volumetric case, the grid cell arrangement consists of hexahedral cells, or it may be *unstructured*, with a cell arrangement consisting of tetrahedra, hexahedral cells, or even combinations of various types of cells. Extremely large data sets cannot be analyzed or visualized in real time unless data reduction/compression methods are used, or "features," extracted from the given data sets in a preprocessing step, are rendered.

In this paper, we focus on 3-dimensional tetrahedral meshes. These meshes provide the greatest possible degree of flexibility and are less restrictive than all other mesh topologies, *e.g.*, Cartesian, rectilinear, and curvilinear. Furthermore, each mesh can be converted into a tetrahedral mesh. Data structures, data traversal, and data rendering for tetrahedral meshes are, in most cases, more involved than for more "structured" representations. Nevertheless, when visualizing very large data sets defined over complex three-dimensional regions it is more convenient to use tetrahedral meshes due to their ability to better adapt to local features. It is also important to investigate means for the representation of tetrahedral meshes at various *levels of detail* for efficient rendering and analysis.

Our method for the generation of a *hierarchy* of tetrahedral meshes is based on collapsing individual tetrahedra and removing them from the mesh. Considering a particular mesh, we weigh each tetrahedron based on a predicted increase in approximation error that would result after its collapse. The tetrahedra are ordered by these weights and collapsed one-by-one, with changes in the errors of the neighboring tetrahedra reflected in the new ordering.

The construction of multiple levels of tetrahedral meshes is a preprocessing step for subsequent data visualization. Speed is not the primary concern when constructing the levels; it is more important

that the resulting data format be compact and allow for simple and efficient access during the visualization process. Error estimates should be known for each level as well.

In Section 2, we review the algorithms related to mesh simplification that apply to our work. In Section 3, we illustrate our technique for triangle meshes in the plane. The main principles become very clear from the discussion of the planar case. In Section 4 we describe the needed extensions for tetrahedral meshes. Implementation issues are discussed in Section 5 and the results of our algorithm are illustrated on a set of complex examples in Section 6. Conclusions and future work are discussed in Section 7.

2 RELATED WORK

Three classes of algorithms exist that directly pertain to our work and that deal with triangle or tetrahedral meshes: Algorithms that simplify the mesh by removing vertices; algorithms that simplify the mesh by removing edges; and algorithms that simplify the mesh by removing higher-level simplices.

Schroeder *et al.*[17] and Renze and Oliver [16] have developed algorithms that simplify a mesh by removing vertices. Vertices to be removed are identified through a distance-to-simplex criterion. Removing a vertex creates a hole in the mesh that must be retriangulated, and several strategies may be used: Schroeder *et al.* use a recursive loop splitting procedure to generate a triangulation of the hole, while Renze and Oliver fill the hole by using an unconstrained Delaunay triangulation algorithm.

Hoppe [9, 10] and Hoppe and Popović [15] describe a progressive-mesh representation of a triangle mesh. This is a continuous-resolution representation based on an edge-collapse operation. The data reduction problem is formulated in terms of a global mesh optimization problem [11], ordering the edges according to an energy minimization function. Each edge is placed in a priority queue by the expected energy cost of its collapse. As edges are collapsed, the priorities of the edges in the neighborhood of the transformation are recomputed and reinserted into the queue. The result is an initial coarse representation of the mesh, and a linear list of edge-collapse operations, each of which can be regenerated to produce finer representations of the mesh. Other edge-collapse algorithms have been described by Xia and Varshney [18], who use the constructed hierarchy for view-dependent simplification and rendering of models, and Garland and Heckbert [3], who utilize quadratic error metrics for efficient calculation of the hierarchy.

Hamann [6, 7], and Gieng *et al.* [4, 5] have developed algorithms that simplify triangle meshes by removing triangles. These algorithms order the triangles according to a weight based partially on the curvature of a surface approximate, partially on the changes in the topology of the mesh due to a triangle collapse, and partially due to the predicted error of the collapse operation. Triangles are inserted into a priority queue and removed iteratively. Modified triangles receive new weights and are inserted back into the priority queue. By selecting a percentage of triangles to be collapsed it is possible to "parallelize" triangle removal.

Cohen *et al.* [2] proposed the idea of a *simplification envelope* of a surface. They produce hierarchical representations of an object, each of which is guaranteed to be within a user-specified distance from the original model. They generalize the concept of *offset surface* to a polygonal representation of an envelope that surrounds the surface within the specified tolerance.

Cignoni *et al.* [1] treat the tetrahedral mesh problem. They use a top-down Delaunay-based procedure to define a tetrahedral mesh that represents a three-dimensional set of points. The mesh is refined by selecting a data point whose associated function value is poorly approximated by an existing mesh and inserting this point into the mesh. The mesh is modified locally to keep the Delaunay property intact.

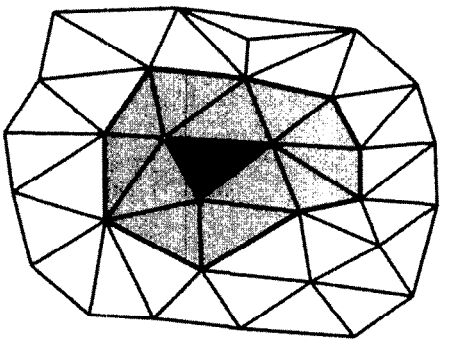

Figure 1: The stencil of a triangle T. The shaded triangles all share a vertex with T; A collapse of the triangle T impacts the triangles of the stencil.

This paper extends the results of Gieng *et al.* [5] to tetrahedral meshes. The general idea is to base the scheme on the predicted deviation from the original scalar field due to a tetrahedron collapse. If the deviation can be measured closely, the complicated weights of Hoppe [11] and Gieng *et al.*[5] should not be necessary. A maximum deviation bound is kept for each tetrahedron in the mesh. This value, together with the predicted increase in the error if the tetrahedron is collapsed, enables us to determine which tetrahedron to collapse and to insure that the maximum deviation over the surface remains less than a specified value. As the mesh is simplified the maximum deviation is updated for each tetrahedron affected by the collapse operation.

Our algorithm is a bottom-up approach that produces a hierarchy of tetrahedral meshes, each of which is guaranteed to be within a specific error distance from the original mesh. A tetrahedron is selected for collapse if it leads to a minimal increase in the overall error of the approximation, and does not increase the global error beyond a maximal error tolerance. The error calculations are local calculations, so the algorithm is fairly efficient even on large meshes. The algorithm collapses an individual tetrahedron by iteratively collapsing its edges. This strategy allows us to avoid many of the topological considerations of Gieng *et al.*'s work. Our algorithm utilizes only the original data points, which allows us to represent the resulting hierarchy very compactly.

3 TRIANGLE COLLAPSE IN THE PLANE

To understand the collapsing of tetrahedra in a three-dimensional mesh, it is useful to first study the collapsing of triangles in a planar mesh. Assume we have a collection of data points $\{v_0, v_1, v_2, ...v_n\}$ in the plane and a set of triangles $\{T_0, T_1, ..., T_m\}$ defining a triangulation of the data points. We assume that the generated triangulation is fair, *i.e.*, the mesh is connected, and each edge in the mesh is shared by at most two triangles. Meshes should not be self-intersecting, *i.e.*, no triangle of the mesh should have an intersection with the interior of another triangle.

We call a triangle T a *vertex neighbor* of a vertex v if v is a vertex of T, and T is an *edge neighbor* of an edge e if e is an edge of T. Each edge has two edge neighbors, while each vertex may have any number of vertex neighbors. The vertex neighbors of a triangle T consist of all triangles that share a vertex with T. The edge neighbors of T are the triangles that share an edge with T. The union of the vertex neighbors of a triangle T is called the *stencil* of T. The stencil contains those triangles that can be modified by collapsing T (see Figure 1).

It is possible to reduce the collapse of a triangle to a sequence

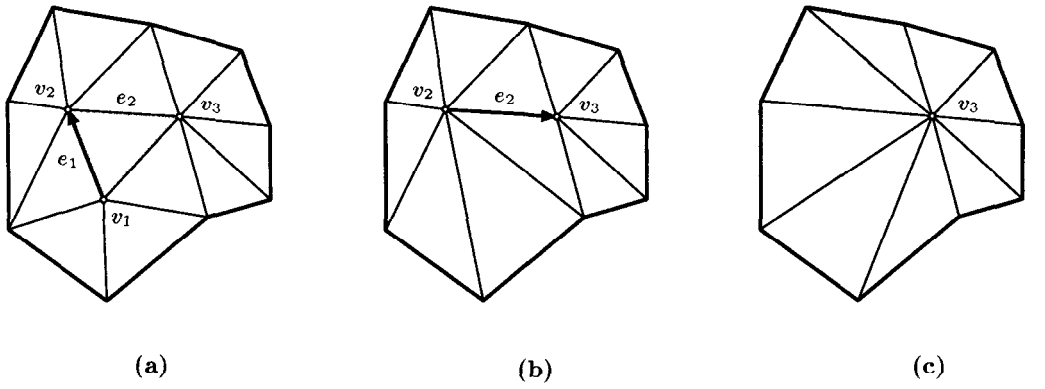

Figure 2: A two-edge collapse of a triangle T: (a) the triangle and its associated stencil; (b) the mesh after the collapse of edge e_1; (c) the mesh after the collapse of both e_1 and e_2.

of edge collapses. Given an edge e, with endpoints v_1 and v_2, we collapse the edge by removing the two triangles sharing the edge and by collapsing v_1 to v_2 (see Figure 2). This operation stretches the triangles that share v_1 as a common vertex to fill the hole. The edge-collapse operation does not commute, *i.e.*, collapsing v_2 to v_1 would produce a different result.

We collapse a triangle T by successively collapsing two of its edges (see Figure 2). In this case, the edge neighbors of T are eliminated from the mesh and the vertex neighbors of two of the vertices of T are stretched to include the third vertex of the triangle. There are nine ways to collapse a triangle, but the results can modify the mesh in only three ways.

3.1 Error Bounds

To calculate a bound for the error due to a triangle collapse, we assume that the triangle mesh represents a scalar field defined by a piecewise-linear spline with individual spline segments $s = F(u, v, w)$, where (u, v, w) are the barycentric coordinates of a point in the triangle. The spline coefficients are the function values at the mesh vertices.

Each triangle T has an associated a "maximal deviation" ϵ_T, which represents a bound on the deviation between the linear spline segment defined by T and the linear spline of the original triangle mesh in the area of T. Original triangles have $\epsilon_T = 0$, and this value is updated whenever a collapse is performed.

Suppose we have selected a triangle T for collapse, and let e be the edge $\overline{v_1 v_2}$. Suppose that v_1 has k vertex neighbors $T_1, T_2, ..., T_k$ and suppose that T_{k-1} and T_k are edge neighbors of e. Then, as v_1 is collapsed to v_2, the triangles $T_1, T_2, ..., T_{k-2}$ are stretched to have v_2 as a common vertex, and v_1 is eliminated. This collapse operation (see Figure 3) creates a new set of triangles $T_1^C, T_2^C, ..., T_{k-2}^C$, which define a new piecewise-linear function F^C. We can calculate the maximal deviation between the two linear splines over each stretched triangle T^C by considering the points where T^C and the original triangles of the stencil of T intersect (see Figure 4).

Let $c_1, c_2, ..., c_j$ be the points where the stretched triangles intersect the original triangles, and let c_0 be the eliminated vertex v_1 (see Figure 5). For each of these points, we know that the induced linear spline cannot deviate more than

$$|F^C(c_i) - F(c_i)| + \max\{\epsilon_T\}$$

from the linear spline defined by the original triangles. This means that, for each point c_i the deviation is bounded by the difference between the two linear splines at c_i plus the maximum of the errors ϵ_T for the triangles that contain c_i as an edge point or vertex point.

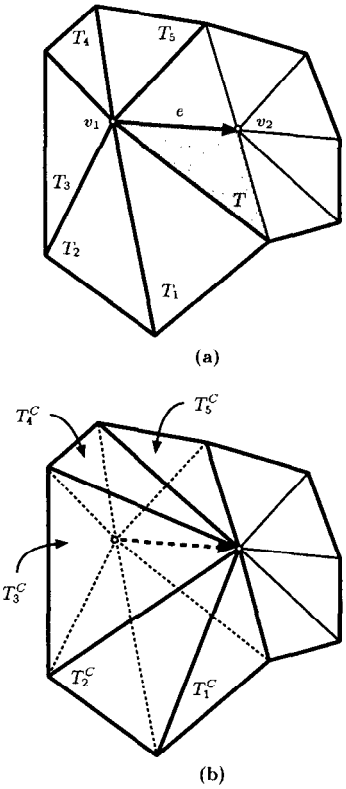

Figure 3: Calculation of the error introduced by collapsing an edge: (a) the triangle T and original mesh; (b) the triangulation after the collapse of edge e.

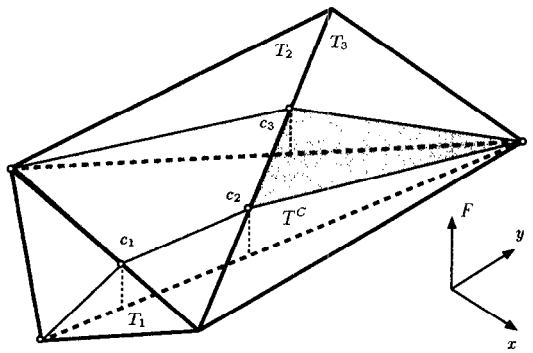

Figure 4: Calculation of the error for a stretched triangle T^C. A bound on the increase in the deviation from the original triangles T_1, T_2 and T_3 is the maximum of the deviations measured at c_1, c_2, and c_3.

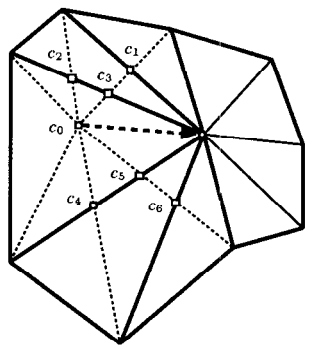

Figure 5: Calculation of the error bound: c_0 is the location of the eliminated vertex v_1; the points c_i, $i = 1, ..., k$, are the intersection points of the stretched triangles and the triangles of the previous mesh. A bound on the error for each stretched triangle can be calculated by finding the deviations in the linear splines at the c_i and adding these values to the maximum of the errors over the original triangles at these points.

Therefore, an error bound for the triangle T^C can be calculated by taking the maximum of this deviation over all points c_i that are contained in the triangle, *i.e.*,

$$B(T^C) = \max_i \left\{ |F^C(c_i) - F(c_i)| + \max \epsilon_T \right\}. \qquad (1)$$

We note that one of the stretched triangles will contain the (eliminated) vertex c_0, and in this case the maximum must also include the deviation between the two linear splines at this point. Using this approximation, we can calculate a new error bound for each triangle T^C.

We define the "cost" of collapsing triangle T to be

$$\delta_T = \min \left[\max_j B(T_j^C) \right] - \epsilon_T, \qquad (2)$$

where the minimum is taken over all six possible collapse strategies for T, and the maximum is taken over the stretched triangles formed by collapsing two edges of T. This is the difference between a predicted error bound for the region and the current error bound at T.

3.2 Outline of an Algorithm

Suppose we are given a set of vertices $\{v_0, v_1, v_2, ...v_n\}$ in the plane and a triangulation defined by the set of triangles $\{T_0, T_1, ..., T_m\}$.

Each triangle T is assigned an error measure ϵ_T, initially set to zero. The value ϵ_T represents a bound on the difference between the linear spline defined by T, and the linear spline segment defined by the initial mesh.

For each triangle T in the mesh, calculate a weight δ_T which reflects the "predicted error," *i.e.*, the maximal deviation that would result when collapsing the triangle, as defined by equation (2). Place the triangles in a priority queue ordered by increasing values of $\epsilon_T + \delta_T$. Thus, the first triangle removed from the queue should have the least effect on the change in the linear spline after the collapse operation.

Next, select a maximum error ϵ_0 for which you wish a mesh to be generated and iteratively perform the following steps:

Remove a triangle T from the queue;

- if $\epsilon_T + \delta_T > \epsilon_0$, then the triangles in the queue represent the simplified mesh;
- if $\epsilon_T + \delta_T \leq e_0$, then
  - **collapse** the triangle T and remove the edge neighbors of T from the queue;
  - **recalculate** ϵ_{TC} and δ_{TC} for each triangle T^C that is stretched as a result of the collapse, and reposition it in the queue;
  - **recalculate** δ_T for each triangle T in the stencil of a stretched triangle T^C.

The last step is necessary to keep the queue in the correct order. Once a triangle is stretched, the cost of collapsing a neighboring triangle changes.

4 TETRAHEDRAL MESHES

We now generalize these principles to simplify a tetrahedral mesh. We assume that we have a set of vertices $\{v_0, v_1, v_2, ...v_n\}$ in three-dimensional space and a set of tetrahedra $\{T_0, T_1, ..., T_m\}$ defining a "triangulation" of this data set. In a similar way to the planar case, we define the vertex neighbors of a point v to be the set of tetrahedra that share v as a vertex, and define the edge neighbors of an edge e to be the set of tetrahedra in the mesh that share e as an edge. In addition, we define the face neighbors of a face f to be the (at most) two tetrahedra in the mesh that share the face f. We also define, in an analogous fashion, the vertex neighbors, edge neighbors and face neighbors of a tetrahedron. A tetrahedron has at most four face neighbors, but can have any number of edge and vertex neighbors.

We collapse a tetrahedron T by successively collapsing three of its edges. One can collapse to any of a tetrahedron's four vertices, and there are ten ways to collapse individual edges to achieve one of the four final states. A collapse removes the edge neighbors of T, and the vertex neighbors of three of the vertices of T are stretched to include the fourth vertex, the vertex to which we collapse.

We insure that the collapse operation does not produce intersecting tetrahedra by comparing the sign on the volume of the original and stretched tetrahedra. If by collapsing an edge of the tetrahedron T, the sign of the volume of a stretched tetrahedron T^C flips, we label the edge as "not-collapsible."

4.1 Error Bounds

To estimate the error, we assume that the tetrahedral mesh represents a linear spline defined by individual spline segments $s = F(u, v, w, t)$, where (u, v, w, t) are the barycentric coordinates of a point inside a tetrahedron. The spline coefficients of F are the function values at the mesh vertices.

Each tetrahedron T will have associated a "maximal deviation" ϵ_T, which will represent a bound on the deviation between the linear spline segment defined by T and the linear spline of the original tetrahedral mesh in the area of T. An original tetrahedron T has $\epsilon_T = 0$, and this value is updated whenever a collapse is performed.

If F^C is the piecewise linear function induced by a collapse operation we can bound the error over a stretched tetrahedron T^C similarly to equation (1), *i.e.*,

$$B(T^C) = \max_i \left\{ |F^C(c_i) - F(c_i)| + \max\{\epsilon_T\} \right\} \qquad (3)$$

where the c_i are the intersection points of the edges of the stretched tetrahedron T^C with the faces of the tetrahedra in the previous mesh (and possibly one of the eliminated vertices of the collapsed tetrahedron) and the maximum is taken over all tetrahedra that contain c_i as an edge point or a vertex. We define the cost of collapsing a tetrahedron T as

$$\delta_T = \min \left[\max_j B(T_j^C) \right] - \epsilon_T, \qquad (4)$$

where the minimum is taken over all possible collapse strategies for the tetrahedron T, and the maximum is taken over the tetrahedra stretched by collapsing three edges of T. This value is the difference between a predicted error bound for the region and the error bound at T.

4.2 Boundary Preservation

The boundary surface of a tetrahedral mesh is given by the set of all faces belonging to exactly one tetrahedron. It is desirable to preserve this boundary surface as much as possible. Some data sets have rectangular boxes as their boundaries while most are much more complex. Given a tetrahedron T, selected for collapse, we check the following:

- If T has a single vertex v on the boundary, the three edges that contain v can only be collapsed to v. The other three edges of T can be collapsed in either direction.
- If T has a two vertices v_1 and v_2 on the boundary, then the four edges of the tetrahedron containing v_1 and v_2 can only be collapsed to these points. The other two edges of the tetrahedron, which includes $\overline{v_1 v_2}$, can be collapsed in either direction.
- If T has three vertices v_1, v_2 and v_3 on the boundary, then the three edges containing the fourth point and one of v_1, v_2 or v_3 can only be collapsed to the boundary points. The three edges on the boundary can be collapsed in either direction.
- if T has a four vertices on the boundary, there are several cases to consider:
  - T is at the corner of mesh. In this case T has only one face neighbor. These tetrahedra can only be collapsed to the corner.
  - T is on a boundary edge. In this case T has two face neighbors. If v_1 and v_2 are on the edge, then the four edges containing v_1 and v_2 can only be collapsed to the edge. The edge $\overline{v_1 v_2}$ can be collapsed in either direction.
  - T has one vertex on an boundary edge. In this case T has three face neighbors. If v_1 is the vertex on the boundary edge, then the three edges incident to v_1 can only be collapsed to v_1.

Each of these cases restricts the number of possible edge collapse operations on T. In some cases, only one or two edges of T may be collapsible, and T may be collapsed to a face or an edge respectively. In general, if T has an edge with a vertex v on the boundary, then the edge must be collapsed to v. If we collapse to the other vertex, the boundary would be compromised. Figure 6 illustrates this for the two-dimensional case.

In general these rules allow the simplification algorithm to work with data sets having convex polyhedral boundaries.

4.3 Algorithms for Mesh Simplification

Two different algorithmic strategies can be used to simplify the mesh. The first, similar to that presented in Section 3.2, utilizes a priority queue and the error prediction mechanism. The second algorithm makes a pass through the complete tetrahedral structure and attempts collapse operations, evaluating each collapse against the error threshold. Each algorithm works with a set of vertices $\{v_0, v_1, v_2, ...v_n\}$, and a set of tetrahedra $\{T_0, T_1, ..., T_m\}$ forming a triangulation of the vertices. Each tetrahedron T in the mesh, carries an "accumulated error" ϵ_T, initially zero.

The first strategy is based upon a priority queue, where for each tetrahedron T, we calculate a weight δ_T which reflects the predicted error increase resulting from collapsing the tetrahedron as defined by equation (4). We place the tetrahedra in the priority queue ordered by increasing $\epsilon_T + \delta_T$.

Select a maximum error ϵ_0 used to terminate the collapse algorithm, and iteratively perform the following steps:

Remove a tetrahedron T from the queue;

- if $\epsilon_T + \delta_T > \epsilon_0$, then the set of tetrahedra in the queue represent the simplified mesh.
- if $\epsilon_T + \delta_T \leq \epsilon_0$,
  - **collapse** the tetrahedron T and eliminate the face neighbors of T from the priority queue;
  - **recalculate** ϵ_{T^C} and δ_{T^C} for each tetrahedron T^C modified by the collapse operation;
  - **recalculate** δ_T for each tetrahedron T in the stencil of a tetrahedron T^C stretched by the collapse operation.

Three algorithms can be implemented that use the collapse operation to generate sets of meshes $\mathcal{M}_0, \mathcal{M}_1, ..., \mathcal{M}_n$ at different levels of detail approximating the original mesh.

(1) Choose a sequence of error bounds $\epsilon_0 < \epsilon_1 < ... < \epsilon_n$. Select tetrahedra from the queue, collapse the tetrahedra, and reinsert the stretched tetrahedra into the queue until the tetrahedron T at the front of the queue satisfies $\epsilon_T + \delta_T > \epsilon_0$. The set of tetrahedra in the priority queue define the mesh $\mathcal{M}_0$.

Using the mesh $\mathcal{M}_0$, collapse the tetrahedra in the queue until the tetrahedron T at the front of the queue satisfies $\epsilon_T + \delta_T > \epsilon_1$. The set of tetrahedra in the queue defines the mesh $\mathcal{M}_1$.

The algorithm continues in this way until mesh $\mathcal{M}_n$ is generated.

This strategy generates a sequence of meshes $\mathcal{M}_0, \mathcal{M}_1, ..., \mathcal{M}_n$ with specified error bounds $\epsilon_0, \epsilon_1, ..., \epsilon_n$ respectively.

(2) Choose a specified number (or percentage) of the original tetrahedra to be collapsed at each step. The intermediate meshes are defined by those tetrahedra remaining in the queue after each step. Here, the error for each intermediate mesh can be reported.

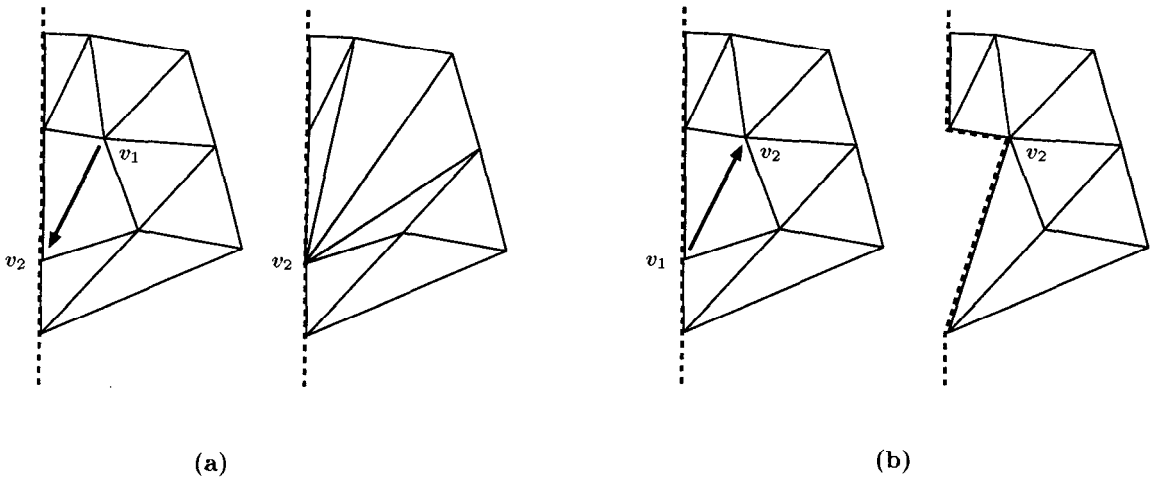

Figure 6: Collapsing edges on the boundary: (a) an edge is collapsed *to* a boundary vertex; (b) the boundary is destroyed when the collapse goes the other way.

(3) Similarly to Gieng *et al.*[5], remove a set of tetrahedra from the queue and collapse them in parallel. The only restriction is that the stencils of the tetrahedra to be collapsed must not intersect. The resulting sequence of meshes can be used for "smooth" transitions between mesh levels.

The second strategy is based upon a sweep through the grid, examining each tetrahedron individually.

> Choose a sequence of error tolerances $\epsilon_0 < \epsilon_1 < \ldots < \epsilon_n$. For each ϵ_i,
>
> - Make a sweep through the grid, examining each tetrahedron T.
> - Attempt to collapse each edge of the selected tetrahedron until either (1) one of these collapses induces an error threshold below ϵ_i, or (2) all six edges of T have been tried.
> - A successful collapse modifies several tetrahedra, which are "stretched" versions of their counterparts before the collapse. For each modified tetrahedron T_C, calculate the error change δ_{T_C} and add this to ϵ_{T_C}.
> - Continue until the simplified mesh cannot be collapsed further without increasing the error above ϵ_i.

The first strategy selects that tetrahedron that will cause the minimal increase in the error at each step. The second strategy utilizes a "greedy" method that selects an arbitrary tetrahedron and attempts to collapse it. If the collapse results in a mesh below the error bound, it is accepted and the process continues. This results in a less structured algorithm, but avoids many of the complex δ_T calculations for tetrahedra that are removed in the collapsing process.

Using the second strategy also enables a simplified test for boundary preservation. Consider a vertex v that is removed by collapsing an edge. If v is on the boundary then if the resulting tetrahedra, modified by the collapse, does not contain v, the boundary has been compromised. Therefore, if v is not contained within the modified tetrahedra, then the collapse is rejected.

5 IMPLEMENTATION ISSUES

We have implemented this algorithm using a simple data structure for tetrahedral meshes. We store a list of vertices and a list of tetrahedra. Each tetrahedron contains links that reference the four vertices of the tetrahedron and the four face neighbors. Calculating the vertex neighbors and edge neighbors of a tetrahedron is straightforward using this data structure.

Each tetrahedron T carries an error estimate ϵ_T and a predicted deviation δ_T. The value δ_T is determined by finding the sequence of edge collapses in T that generates the minimum error increase. We store this sequence of collapses in the tetrahedron data structure. When the tetrahedron is to be collapsed, we need not recalculate this sequence.

Our algorithm is based on an "edge collapse" paradigm and can be implemented using only edge-collapse strategies. However, we felt that the overhead of implementing and maintaining an edge data structure would be overwhelming for the large data sets that we use.

6 RESULTS

Results of our work are shown in Figures 7-15. We utilize voxelized data sets, where each voxel of the original data set is initially split into six tetrahedra (see [14]). We utilized the "greedy algorithm of Section 4.3 in each case, and specified a maximum error for each approximating mesh.

Our first example, shown in Figures 7-9, collapses tetrahedra until a specified error tolerance is reached. Figure 7 illustrates a piecewise-linear scalar field over a unit cube containing 41,154 tetrahedra, shown in Figure 8. The function is defined by

$$f(x,y,z) = \begin{cases} 0 & \text{if } x < 0; \\ x & \text{if } 0 \le x \le 1; \\ 1 & \text{if } x > 1. \end{cases}$$

Simplifying the mesh with any small user-specified error threshold yields the mesh shown in Figure 9, which contains 25 tetrahedra and still represents the scalar field exactly. (We note that 15 tetrahedra would be optimal.) One would expect that the simplification scheme works well with simple piecewise-linear functions – and it does.

Our second example is an MRI scan of a human skull and is represented by an array of 64x64x109 density values. Splitting each

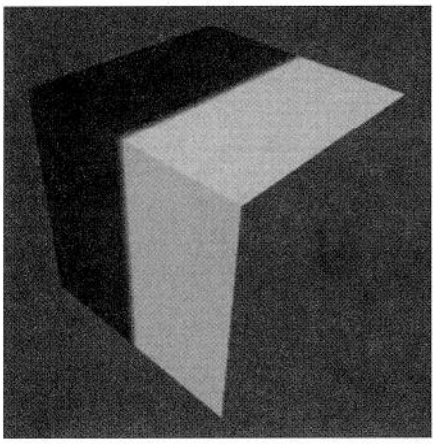

Figure 7: A piecewise-linear example: The values of the scalar field are only rendered on the boundary of the grid.

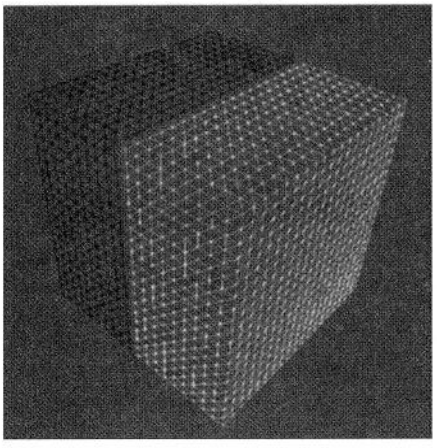

Figure 8: The underlying tetrahedral mesh for the function shown in Figure 7. The field contains 41,154 tetrahedra. The edges are colored according to the scalar function.

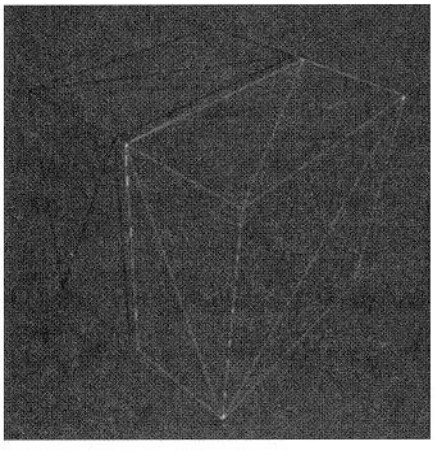

Figure 9: The reduced tetrahedral mesh for the function shown in Figure 7. This mesh contains 25 tetrahedra and represents the scalar field exactly.

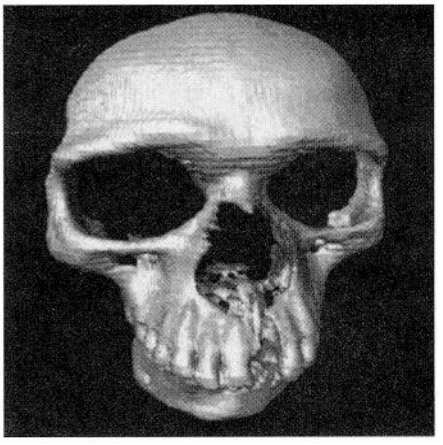

Figure 10: An isosurface of a skull data set containing 2.7 million tetrahedra in the original mesh.

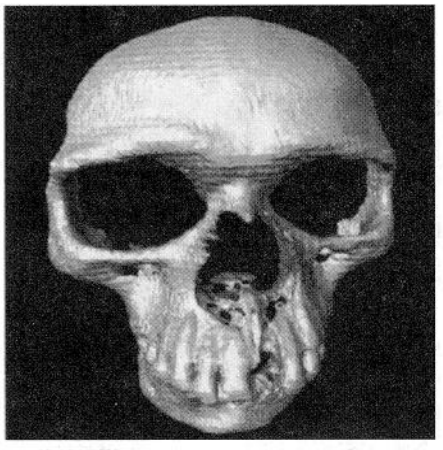

Figure 11: Simplified skull data set: An error bound was chosen that reduces the data set to approximately 910,000 tetrahedra. The same isosurface value was used as in Figure 10.

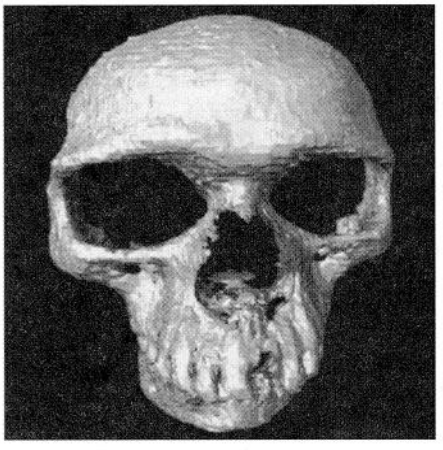

Figure 12: Skull data simplification containing approximately 209,000 tetrahedra: The isosurface is shown using the same value as in Figure 10.

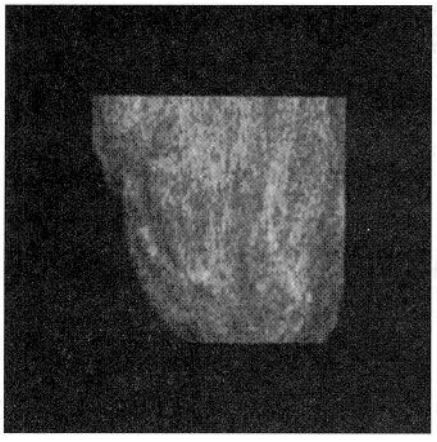

Figure 13: A ray-traced image of a section of the brain of a Macaque monkey. The original data set contains approximately 1.3 million tetrahedra.

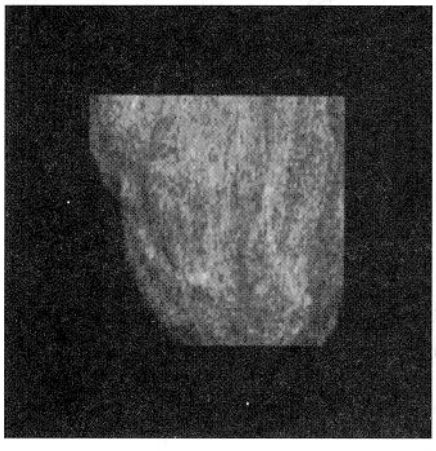

Figure 14: A ray-traced image of the simplified brain data set. An error bound was chosen that reduces the data set to approximately 700,000 tetrahedra.

voxel into 6 tetrahedra, yields approximately 2,700,000 tetrahedra for the original mesh. Figure 10 shows an isosurface generated for the original data set. Figure 11 shows an isosurface for a simplified mesh containing approximately 910,000 tetrahedra, and Figure 12 shows the same isosurface on a simplified mesh containing approximately 209,000 tetrahedra – roughly 8% of the original size.

The third example is a section of a brain taken from a Macaque monkey. Figure 13 shows a ray-traced image of the brain using the original data set. This contains approximately 1.3 million tetrahedra. Figure 14 shows a simplified brain data set with approximately 700,000 tetrahedra. This image is almost identical to that shown in Figure 13. Figure 15 shows the image of a simplified brain data set with approximately 158,000 tetrahedra – approximately 12% of the size of the original. Despite some "feathering" in the lower-left corner of this image, the image is quite good.'

'We note that a 32x32x32 data set with each voxel split into 6 tetrahedra would contain 196,608 tetrahedra.

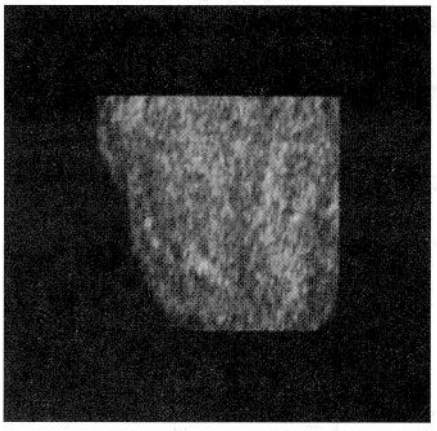

Figure 15: A ray-traced image of a simplified brain data set. An error bound was chosen that reduced the set to approximately 158,000 tetrahedra.

7 CONCLUSIONS

We have presented a method for the simplification of tetrahedral meshes approximating a trivariate function. The simplification of the mesh is based upon a tetrahedral collapse algorithm and local error calculations that insure that the mesh remains within a user-specified tolerance of the original. Several methods can be applied to generate various mesh hierarchies to be used in level-of-detail applications. We have found this a useful tool to treat massive three-dimensional data sets defined by arbitrary grid structures.

We plan to generalize our approach to allow more flexible placement of vertices when collapsing tetrahedra and to compute the linear spline coefficients in an optimal way for each triangulation level (see [8]). Furthermore, we intend to extend and apply our algorithm to vector fields and time-varying fields.

8 ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under contract ASC 9624034, the Office of Naval Research under contract N00014-97-1-0222, the NASA Ames Research Center under the NRA2-36832(TLL) program, the Army Research Office under contract AR036598-MA-RIP, and Lawrence Livermore National Laboratory under contract W-7405-ENG-48 (B335358). We would like to thank the members of the Visualization Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, for their support.

References

[1] CIGNONI, P., DE FLORIANI, L., MONTONI, C., PUPPO, E., AND SCOPIGNO, R. Multiresolution modeling and visualization of volume data based on simplicial complexes. In *1994 Symposium on Volume Visualization* (Oct. 1994), A. Kaufman and W. Krueger, Eds., ACM SIGGRAPH, pp. 19–26.

[2] COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, JR., F. P., AND WRIGHT, W. Simplification envelopes. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 119–128.

[3] GARLAND, M., AND HECKBERT, P. S. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), T. Whitted, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 209–216.

[4] GIENG, T. S., HAMANN, B., JOY, K. I., SCHUSSMAN, G. L., AND TROTTS, I. J. Smooth hierarchical surface triangulations. In *Proceedings of Visualization 97* (Oct. 1997), H. Hagen and R. Yagel, Eds., IEEE Computer Society Press, Los Alamitos, California, pp. 379–386.

[5] GIENG, T. S., HAMANN, B., JOY, K. I., SCHUSSMAN, G. L., AND TROTTS, I. J. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (1998), (to appear).

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

[7] HAMANN, B., AND CHEN, J.-L. Data point selection for piecewise linear curve approximation. *Computer-Aided Geometric Design 11*, 3 (June 1994), 289–301.

[8] HAMANN, B., AND JORDAN, B. Triangulations from repeated bisection. In *Mathematical Methods for Curves and Surfaces II*, M. Dæhlen, T. Lyche, and L. Schumacker, Eds. Vanderbilt University Press, Nashville, Tennessee, 1998, pp. 229–236.

[9] HOPPE, H. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 99–108.

[10] HOPPE, H. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), T. Whitted, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 189–198.

[11] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), J. T. Kajiya, Ed., vol. 27, pp. 19–26.

[12] KAUFMAN, A. E. Volume visualization. *ACM Computing Surveys 28*, 1 (Mar. 1996), 165–167.

[13] KLEIN, R., LIEBICH, G., AND STRASSER, W. Mesh reduction with error control. In *Proceedings of IEEE Visualization '96* (Oct. 1996), IEEE Computer Society Press, Los Alamitos, California, pp. 311–318.

[14] NIELSON, G., MÜLLER, H., AND HAGEN, H., Eds. *Scientific Visualization: Overviews, Methodologies, and Techniques*. Academic Press, 1997.

[15] POPOVIĆ, J., AND HOPPE, H. Progressive simplicial complexes. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), T. Whitted, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 217–224.

[16] RENZE, K. J., AND OLIVER, J. H. Generalized unstructured decimation. *IEEE Computer Graphics & Applications 16*, 6 (Nov. 1996), 24–32.

[17] SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. Decimation of triangle meshes. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 65–70.

[18] XIA, J. C., AND VARSHNEY, A. Dynamic view-dependent simplification for polygonal models. In *Proceedings of IEEE Visualization '96* (Oct. 1996), IEEE Computer Society Press, Los Alamitos, California, pp. 327–334.