Simplification of Tetrahedral Meshes with Error Bounds

Issac J. Trotts, *Student Member, IEEE Computer Society*, Bernd Hamann, *Member, IEEE*, and Kenneth I. Joy, *Member, IEEE Computer Society*

Abstract—We present a method for the construction of multiple levels of tetrahedral meshes approximating a trivariate scalar-valued 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 edges of the mesh. Each triangulation defines a linear spline function, where the function values associated with the vertices are the spline coefficients. Error bounds are stored for individual tetrahedra and are updated as the mesh is simplified. Two algorithms are presented that simplify the mesh within prescribed error bounds. Each algorithm treats simplification on the mesh boundary. The result is a hierarchical data description suited for efficient visualization of large data sets at varying levels of detail.

Index Terms—Mesh simplification, hierarchical representation, multiresolution method, scattered data, spline, tetrahedral mesh, visualization.

1 Introduction

One of the most critical 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 multiple approximations of the data set, representing the data set a varying levels of accuracy. A data hierarchy may consist of levels characterized by only a few points, or by several million points, where each data set captures, as much as possible, the features of the original data. A hierarchical representation, or *multiresolution* representation, allows the study of large-scale features by considering a coarse data representation and the study of small-scale features by considering a high-resolution data representation.

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 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 extracted features are rendered.

In this paper, we focus on three-dimensional tetrahedral meshes. These meshes provide the greatest flexibility and are less restrictive than other mesh topologies. Cartesian, rectilinear, and curvilinear meshes can all 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 hierarchies of tetrahedral meshes is based on collapsing edges that cause a minimal increase of the error between the simplified mesh and the original one. We bound the error between a simplified mesh and the original mesh by calculating the deviations in the linear spline approximations to the scalar fields of the original mesh and the modified mesh. Error is accumulated as a result of continued edge collapses, giving us a bound on the maximal deviation of the simplified mesh from the original.

We present two algorithms to implement this methodology. The first algorithm orders the tetrahedra by the predicted error caused by an edge collapse. Edges are then collapsed one-by-one and the new ordering is updated to reflect error bounds for the neighboring tetrahedra. The second algorithm makes a pass through all tetrahedra and utilizes a greedy strategy to choose edges for collapse.

The construction of multiple levels of tetrahedral meshes is a preprocessing step for data visualization. In general, speed is not the primary concern when constructing the levels; it is more important that the resulting data format is compact and allows for simple and efficient access during the visualization process. Error bounds should be known for each level as well.

In Section 2, we review mesh simplification algorithms that relate to our work. In Section 3, we illustrate our

• *The authors are with the Department of Computer Science, University of California, Davis, CA 95616-8562.*
E-mail: {trotts, hamann, joy}@cs.ucdavis.edu..

For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 109335.

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 extensions for tetrahedral meshes. In Section 5, we describe two algorithms for the simplification of tetrahedral meshes using the error approximation techniques defined in Section 4. Implementation issues are discussed in Section 6, and the results of our algorithm are illustrated on a set of complex examples in Section 7. Conclusions and future work are discussed in Section 8.

2 Related Work

Three classes of algorithms have been developed that directly pertain to our work and that deal with triangle or tetrahedral meshes: Algorithms that simplify a mesh by removing vertices, algorithms that simplify a mesh by removing edges, and algorithms that simplify a mesh by removing higher-level simplices. All algorithms utilize an "error estimation" strategy to determine an order for the collapse operations.

Schroeder et al. [1] and Renze and Oliver [2] have developed algorithms that simplify a mesh by removing vertices. Vertices to be removed are identified through a distance-to-simplex measure. 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 [3], [4] and Hoppe and Popovic [5] 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 [6], 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 [7], who use the constructed hierarchy for view-dependent simplification and rendering of models, and by Garland and Heckbert [8], who utilize quadratic error metrics for efficient calculation of the hierarchy.

Staadt and Gross [9] have extended the progressive mesh algorithm of Hoppe [3] to tetrahedral meshes. They utilize a "cost" function to determine a priority for the collapse operations, which penalizes operations due to the shape of the resulting tetrahedra, the changes in the volume of the tetrahedra in the neighborhood of the collapse, and the differences in scalar values along an edge. Each of these quantities can be weighted separately to produce the final cost function. An edge-length criterion can also be used in the cost functions.

Cignoni et al. [10] treat the tetrahedral mesh problem by using 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 preserve the Delaunay property. This algorithm has been generalized to include a decimation strategy in [11]. Edges to be collapsed are selected by analyzing the field gradient at each vertex $\mathbf{p}$ of the grid. This value is calculated as a weighted average of the gradients of all tetrahedra adjacent to $\mathbf{p}$, where the weight of tetrahedron T is given by the solid angle subtended by T at $\mathbf{p}$. The edges are then ordered according to the difference between the gradients at the endpoints of the edge. Edges with smaller differences are eliminated first.

Hamann [12], Hamann and Cheng [13], and Gieng et al. [14], [15] have developed algorithms that simplify triangle meshes by removing triangles. These algorithms order the triangles using a weights based partially on the curvature of a surface approximation, partially on the changes in the topology of the mesh due to a triangle collapse, and partially due to a predicted error estimate of a 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 number of triangles from the front of the queue whose resulting collapse operations do not conflict, it is possible to "parallelize" triangle removal.

Error approximation between meshes is addressed by several researchers. Cohen et al. [16] utilize an edge collapse-strategy to simplify polygonal models. They order the edges by using the distances between corresponding points of the mapping. Bajaj and Schikore [17] measure the error of the scalar fields across the surface. Their method is similar to that discussed in this paper, but it is surface based. A similar technique is also by Hoppe [18] for the error calculations in level-of-detail rendering of terrain modeling. Klein et al. [19] calculate the Hausdorff distance between two meshes.

This paper is an expansion of our work discussed in [20]. The general idea is to base tetrahedral collapse operations on the deviation of the simplified scalar field from the original field. If the deviation can be measured closely, the complex weights of Hoppe et al. [6], Staadt and Gross [9], and Gieng et al. [15] 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 as a result of collapsing a tetrahedron, enables us to determine which tetrahedron to collapse and to ensure that the maximum deviation over the scalar field remains less than a specified value. As the mesh is simplified the maximum deviation is updated for each tetrahedron affected by the collapse operations.

In our initial work [20], we collapsed individual tetrahedra, collapsing three edges of the tetrahedron and measuring the error induced by the collapse. In this paper, we discuss an approach based on an edge-collapse strategy. With this approach, we can provide precise error bounds on the simplified mesh and we can simplify the discussion of the decimation process by avoiding the complex topological problems that result from the collapse of several edges

constituting the collapse of a tetrahedron. This approach allows us to extend our strategy to unstructured meshes with complex boundaries. We also discuss a greedy simplification strategy, which allows the edge-collapse process to become much more efficient.

Our basic algorithm is a bottom-up approach that produces a hierarchy of tetrahedral meshes, each of which is guaranteed to be within a specific error from the original mesh. The error calculations are local calculations, so the algorithms are fairly efficient even on large meshes. The algorithm can be configured to utilize only the original data points, which allows us to represent the resulting hierarchy compactly.

3 Triangle Mesh Simplification 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, each edge in the mesh is shared by at most two triangles, and no triangle of the mesh intersects 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 at most 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, and the union of the vertex neighbors of an edge e is called the *stencil* of e. The stencil of an edge e contains those triangles that are modified when e is collapsed, see Fig. 1. We also define the *extended stencil* of an edge e to be the union of the stencils of triangles T that are in the stencil of e, see Fig. 2. This stencil defines the set of edges whose collapse affects the stencil of e.

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 and v_2 to a new point v, see Fig. 3. The points v_1 and v_2 are commonly called the parents of v in the mesh hierarchy, while v is called the child of v_1 and v_2. This operation removes two triangles from the stencil of e and stretches the remaining triangles of the stencil to contain the new vertex v. As a special case, we can collapse an edge to one of its end points. (The collapse operation does not commute, i.e., collapsing v_2 to v_1 is different from collapsing v_1 to v_2, see Fig. 4.)

3.1 Error Bounds

To calculate a bound for the error due to an edge collapse, we assume that the triangle mesh approximates 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 a triangle. The spline coefficients are the function values at the mesh vertices.

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

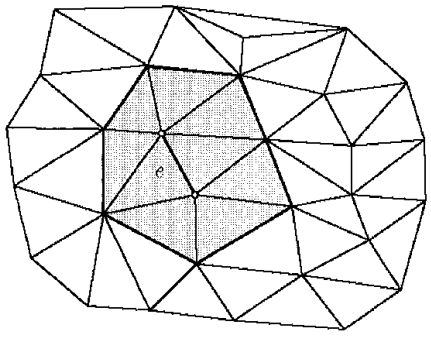


Fig. 1. The stencil of an edge e; the shaded triangles all share a vertex with e; a collapse of e impacts the triangles of the stencil e.

Suppose we have selected an edge e for collapse and v_1 and v_2 are the end points of e. Let v be the new point to which the edge will be collapsed. Suppose that the stencil of e consists of k triangles $T_1, T_2, ..., T_k$, where T_{k-1} and T_k are the edge neighbors of e. Then, as the edge is collapsed to v, the triangles $T_1, T_2, ..., T_{k-2}$ are stretched to share v as a common vertex, and T_{k-1} and T_k are eliminated. This collapse operation creates a new set of triangles, $T_1^C, T_2^C, ..., T_{k-2}^C$, and defines a new linear spline F^C over the modified mesh, see Fig. 5.

For each stretched triangle T_C, we calculate a bound on the deviation between the two linear splines over T^C by considering the points where T^C and the triangles of the stencil of T intersect, see Fig. 6.

Let c_1 and c_2 denote the vertices v_1 and v_2 and let $c_3, c_4, ..., c_k$ be the points where the stretched triangles intersect the original triangles, see Fig. 7. For each of the c_i, we know that the induced linear spline F^C cannot deviate more than

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

from the linear spline defined by the original triangles. Here, the maximum is taken over the two triangles that are the edge neighbors for the edge e that contains c_i. 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

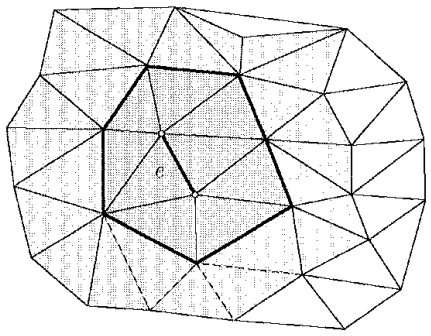


Fig. 2. The extended stencil of an edge e; the shaded triangles all contain edges whose collapse would affect the stencil of e.

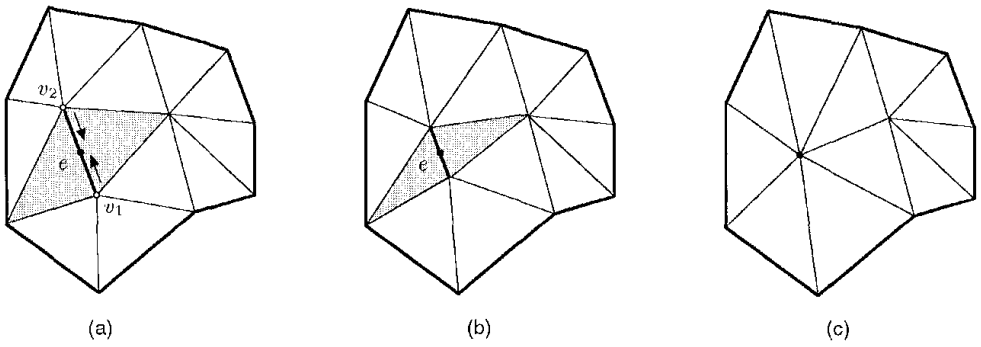


Fig. 3. Collapsing an edge in a mesh: The edge e is collapsed by moving v_1 and v_2 to a new vertex v; the two shaded triangles are eliminated and the remaining triangles of the stencil of e are stretched.

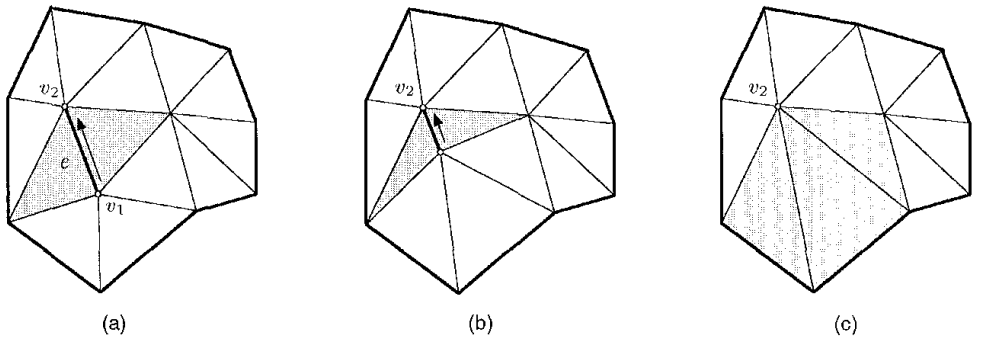


Fig. 4. The special case of collapsing an edge to one of its end points: The edge e is collapsed by moving v_1 to v_2; the two shaded triangles are eliminated, and the remaining triangles of the vertex neighborhood of v_1 are stretched.

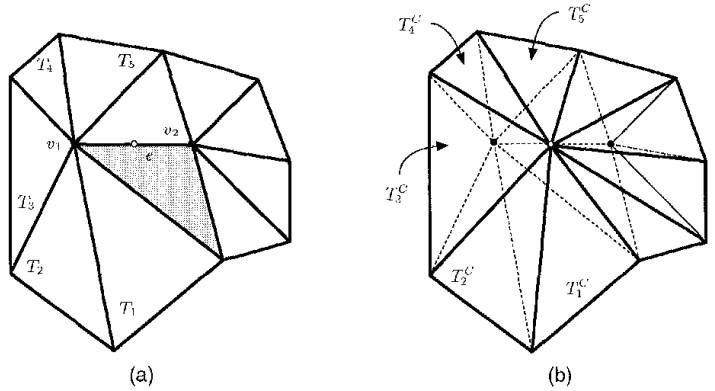


Fig. 5. Effect on the stencil of an edge e of a triangle T when e is collapsed: (a) the triangle T and original mesh; (b) the triangulation after the collapse of edge e.

maximum of the errors ϵ_T for the triangles that contain c_i as an edge point or vertex point.

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

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

We note that two of the stretched triangles will contain the (eliminated) vertices c_1 or c_2 and, in these cases, the maximum must also include the deviation between the two linear splines at these points. Using this approximation, we can calculate a new error bound for each triangle T^C.

We define the "cost" of collapsing triangle T as

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

where the minimum is taken over the three possible edge collapses for T and the maximum is taken over the stretched triangles formed by collapsing two edges of T. The cost of collapsing T is the difference between a predicted error bound for the region and the current error bound of T.

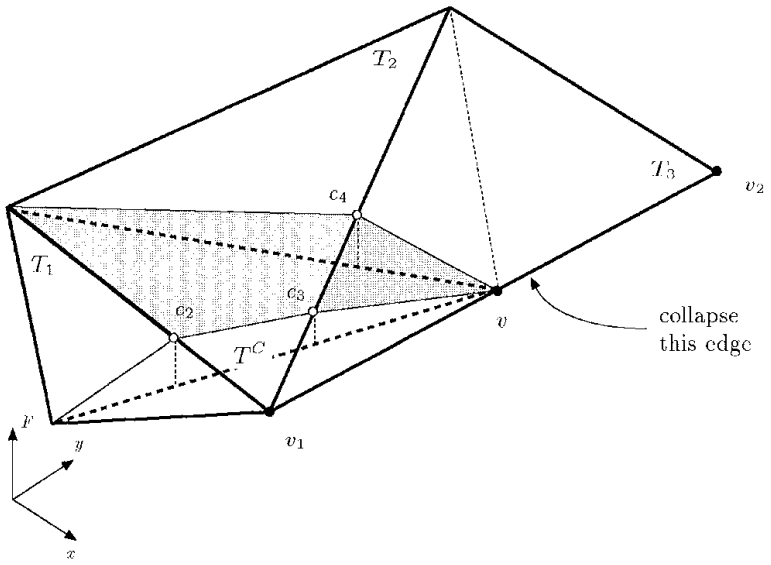


Fig. 6. Calculation of the error for a stretched triangle T^C. Triangle T_1 becomes triangle T_C via the collapse of edge $\overline{v_1 v_2}$ to v. A bound on the increase in the deviation from the triangles T_1, T_2, and T_3 is the maximum of the deviations at the points c_2, c_3, and c_4.

3.2 Boundary Preservation

The boundary of a triangular mesh is given by the set of all edges belonging to exactly one triangle. It is desirable to preserve this boundary as much as possible while collapsing edges. Given an edge e, selected for collapse, we check the following:

- If e has a single vertex v on the boundary, then the edge can only be collapsed to v. Collapsing to any other point on the edge would compromise the boundary, see Fig. 8.
- If e has two vertices v_1 and v_2 on the boundary, then the edge may be collapsed only if the change in the boundary is less than a prescribed threshold.

Each of these cases restricts the number of possible edge collapse operations for a triangle T. In some cases, only one or two edges of a boundary triangle T may be collapsible. 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 any other point on the edge, the boundary would be destroyed.

These conditions ensure that we do not compromise the quality of the boundary and this approach works well for triangular meshes whose boundaries are squares or rectangles. In general, a boundary-preserving scheme should work for an arbitrary boundary and should allow the boundary to be compromised within a certain tolerance.

To accomplish this, we add an "exceptional" point v_∞, the vertex-at-infinity, to the mesh and connect this point to each vertex on the boundary. Triangles having among their vertices are called exceptional triangles. With the addition of the exceptional triangles, we can treat the mesh as a "closed" triangulation and, thus, we can define complete stencils on the boundary. With these additional triangles, the boundary can be defined as the set of edges e such that one of the two edge neighbors of e is an exceptional triangle. We remove exceptional triangles from the mesh by collapsing the boundary edges. The edges of the triangles having v_∞ as an endpoint are not collapsed.

The error bound associated with an exceptional triangle T is the maximum deviation of the boundary edge of T from the actual boundary of the original mesh, measured by the *Hausdorff distance* [19]. If we collapse an edge on the boundary, as shown in Fig. 9, we update the exceptional triangles associated Hausdorff distances—the distance of the simplified boundary from the from the original. This allows us to specify a threshold that implies a maximal deviation of the boundary of a simplified mesh from the boundary of the original mesh.

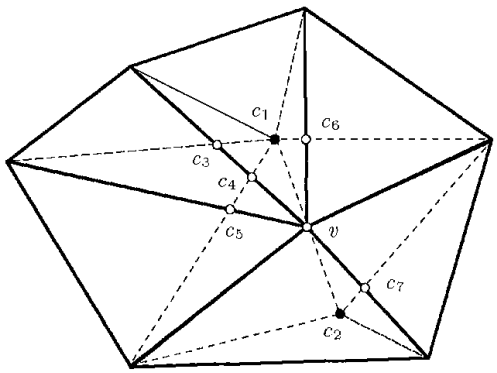


Fig. 7. Contributors to the error bound. The location of the eliminated vertex v_1 is c_1; c_2 is the location of the eliminated vertex v_2; and the points c_i, $i = 3, ..., 7$, are the intersection points of the stretched triangles and the triangles of the previous mesh. A bound on the error increase for each stretched triangle can be calculated by finding the deviations in the two linear splines at the points c_i and adding these values to the maximum of the errors over the original triangles at these points.

3.3 Outline of the Overall Algorithm

We describe a simple algorithm that incorporates the above techniques to simplify a planar mesh of triangles. 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

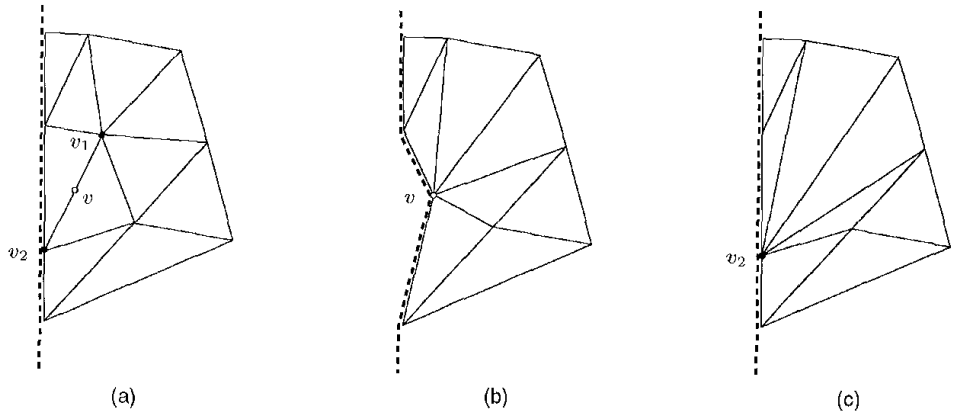


Fig. 8. Collapsing edges that affect the boundary: The requirements ensure that a boundary edge can be collapsed as necessary, but that interior edges containing boundary points can only be collapsed in one direction. Case (a) shows the original mesh, case (b) illustrates the mesh if an interior edge is collapsed, and case (c) illustrates the collapse of the interior edge to the vertex on the boundary. In case (b), the geometry of the mesh boundary is destroyed.

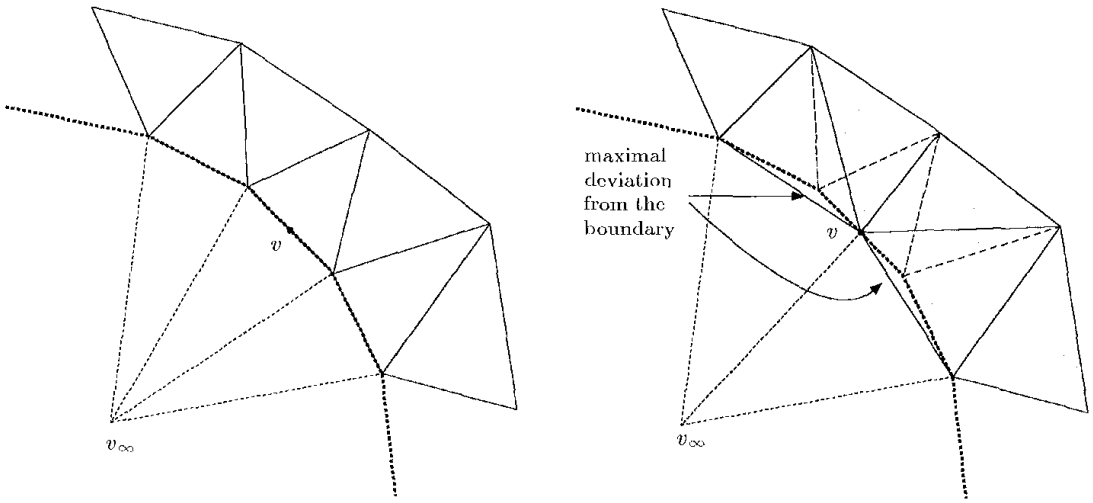


Fig. 9. When collapsing edges on the boundary, two triangles are eliminated from the mesh: one exceptional triangle and one mesh triangle. The remaining exceptional triangles include the new vertex v and the errors of the exceptional triangles are updated to specify the Hausdorff distance of the simplified mesh from the original one.

ϵ_T, initially set to zero. The value ϵ_T represents a bound on the difference between the linear polynomial defined by a triangle T and the linear spline defined by the initial mesh.

For each triangle T in the mesh and each edge e_i of T, we calculate an edge weight δ_i which reflects the "predicted error," i.e., the maximal deviation that would result when collapsing edge e_i of the triangle, as defined by (2). We define the weight of T to be the accumulated error ϵ_T (initially zero) plus the minimum of the calculated edge deviations ($\min_i \delta_i$), and we place the triangles in a priority queue ordered by increasing values of $\epsilon_T + \min_i \delta_i$. 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, we select a maximum error ϵ threshold for which one wishes a mesh to be generated, and iteratively perform the following steps:

Remove a triangle T from the queue;

- if $\epsilon_T + \delta_T > \epsilon$, then the triangles in the queue represent the simplified mesh;
- if $\epsilon_T + \delta_T \leq \epsilon$, then
  - **collapse** the edge e_i that yields the minimal δ_i for T, and remove the edge neighbors of e_i from the queue;
  - **recalculate** ϵ_{T^C} and δ_{T^C} 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 may change.

4 Tetrahedral Meshes

We now generalize these principles to tetrahedral meshes. We assume that we have a set of vertices

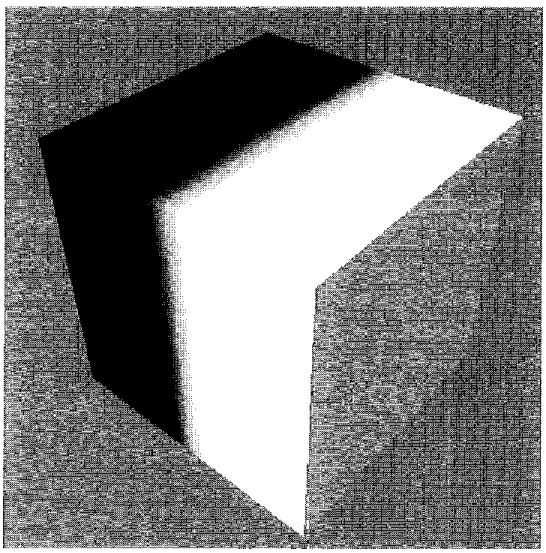


Fig. 10. A piecewise-linear example: The values of the scalar field are only rendered on the boundary of the grid.

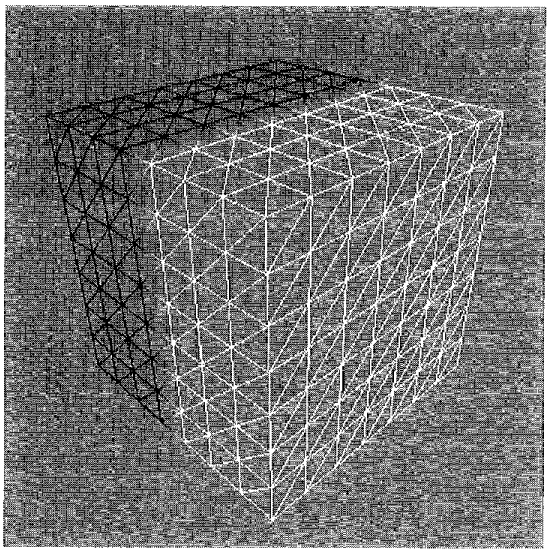


Fig. 11. The underlying tetrahedral mesh for the function shown in Fig. 10. The field contains 2,058 tetrahedra. The edges are colored according to the values of the scalar field.

$\{v_\infty, v_0, v_1, v_2, ..., v_n\}$ in three-dimensional space and a set of tetrahedra $\{T_0, T_1, ..., T_m\}$ defining a triangulation of the vertex set. The triangulation defined by these tetrahedra is closed: Boundary tetrahedra share a face with a tetrahedron that contains the vertex-at-infinity v_∞.

Similarly to the planar case, we define the vertex neighbors of a point v to be the set of tetrahedra T that share v as a vertex. The edge neighbors of an edge e are defined to be the set of tetrahedra that share e as an edge. In addition, we define the face neighbors of a face f to be the 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 four face neighbors, but it can have any number of edge and vertex neighbors. The stencils of a tetrahedron T, a face f, and an edge e, and the extended stencil of an edge e are defined similarly to the planar case.

We collapse a tetrahedron T by collapsing one of its edges. Given an edge e, with end points v_1 and v_2, we collapse the edge to a new point v by removing all edge neighbors of the edge e, stretching the remaining elements of the stencil of e to share v as a vertex. This operation removes several tetrahedra from the mesh. On the boundary, it may be necessary to collapse a tetrahedron to one of the two vertices of the edge e.

We ensure that the collapse operation does not produce intersecting tetrahedra by comparing the signs on the volumes of the original and stretched tetrahedra. If collapsing an edge of a tetrahedron T leads to a change in sign of the volume of the affected tetrahedra, we label the edge as "noncollapsible."

4.1 Error Bounds

To bound the error, we assume 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 has an associated error bound ϵ_T, which represents 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 an error value of $\epsilon_T = 0$ 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 (1), i.e.,

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

where the points c_i are the intersection points of the edges of the stretched tetrahedron T^C and the faces of the tetrahedra in the previous mesh, the intersection points of the faces of T^C, and the edges of the tetrahedra in the previous mesh, and the eliminated vertices of the collapsed edge. 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 edges of a tetrahedron T, and the maximum value is taken over the tetrahedra stretched by collapsing the six edges of T. This value is the difference between a predicted error bound for the region and the error bound of 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 that represent the boundaries of their convex hulls, while most have boundaries that are much more complex. Given an edge e, selected for collapse, we apply the following tests:

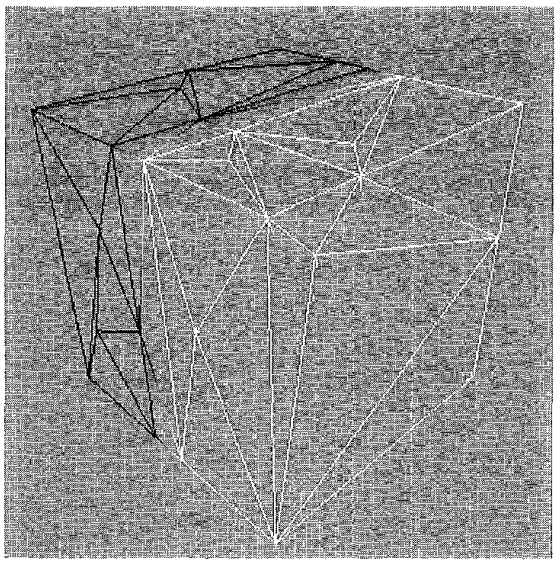


Fig. 12. The reduced tetrahedral mesh resulting from the application of Algorithm 1 to the linear scale field of Fig. 10. This mesh contains 115 tetrahedra and represents the scalar field exactly.

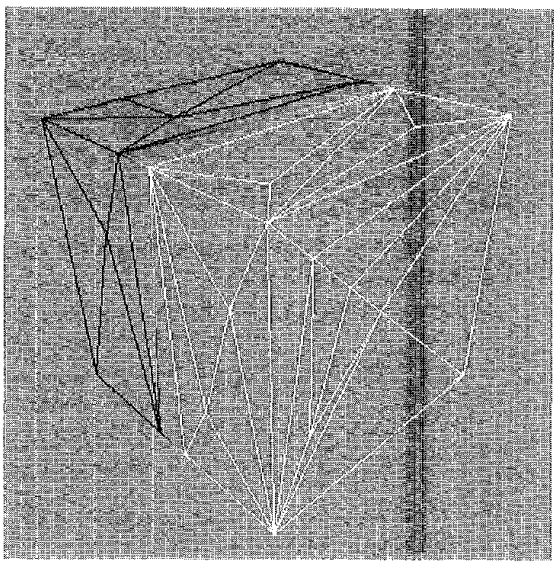


Fig. 13. The reduced tetrahedral mesh resulting from the application of Algorithm 2 to the linear scalar field of Fig. 10. This mesh contains 69 tetrahedra and represents the scalar field exactly.

- If e has a single vertex v on the boundary, then the edge can only be collapsed to v. Collapsing to any other point on the edge will compromise the boundary.
- If e has two vertices v_1 and v_2 on the boundary, then the edge may be collapsed only if the change to the boundary is less than a prescribed error bound.

Each of these cases restricts the number of possible edge collapse operations for tetrahedra that have vertices on the boundary. In some cases, only one or two edges of T may be collapsible. In general, if T has an edge with a single vertex v on the boundary, then the edge must be collapsed to v. If we collapsed to the other vertex, the boundary would be destroyed.

As in the planar case, we add an "exceptional" point v_∞, the vertex-at-infinity, to the mesh and connect this point to each face on the boundary. With the addition of the resulting "exceptional" tetrahedra we can treat the mesh as a "closed" triangulation and, thus, we can define complete stencils on the boundary. With these additional tetrahedra, the boundary can be defined as the set of faces f such that one of the two face neighbors of f is an exceptional tetrahedron. Exceptional tetrahedra can only be removed from the mesh by collapsing the boundary edges. Edges of exceptional tetrahedra having v_∞ as an endpoint are not collapsed.

The error bound associated with an exceptional tetrahedron T is the maximum deviation of the simplified boundary from the original boundary of the original mesh, measured by the Hausdorff distance [19]. If we collapse an edge on the boundary, we update the exceptional tetrahedra's associated Hausdorff distance. If the prescribed error threshold on the boundary is zero, then the algorithm will only collapse edges where the boundary is planar.

Care must be taken to accurately integrate the boundary simplification steps with the general mesh simplification algorithm, as the error values have a different meaning. In the case of the internal tetrahedra of the mesh, the error is based upon the change in the scalar values. However, in the case of the boundary, the error is based upon the Hausdorff distance. In each case, our algorithms consider the error of the internal tetrahedra and select edges for collapse. If the selected edge e is on the boundary, then we consider the Hausdorff distance criteria contained in the exceptional tetrahedra that contain the edge e. The edge is collapsed only if the error threshold for the boundary and the error threshold for the internal tetrahedra are not exceeded.

5 Algorithms for Mesh Simplification

We present two different algorithms that can be used to simplify a mesh. The first, similar to that presented in Section 3.3, utilizes a priority queue and the error prediction mechanism. The second algorithm uses a greedy strategy and makes a pass through the complete tetrahedral mesh, attempting collapse operations for each tetrahedron. Each error resulting from a collapse operation is compared to an error threshold and the collapse is retained if the approximated error is less than the threshold value.

The input to each algorithm consists of two error values, ϵ and ϵ_B. The value ϵ represents the maximum allowable error in the interior of the mesh (the deviation of the linear splines), and ϵ_B represents the maximum allowable deviation on the boundary of the mesh (the Hausdorff distance). Each tetrahedron in the interior of the mesh stores an error value that is compared against ϵ and each exceptional tetrahedron (containing the vertex v_∞) stores an error value that is compared against ϵ_B. For simplicity, we describe the algorithms referring to the single global error threshold ϵ. The boundary threshold ϵ_M only is considered if the collapsed edge is on the boundary.

TABLE 1

Data Set	Algorithm	Internal Error Tol.	Boundary Error Tol.	Tetrahedra Remaining	Percent Remaining	Simplification Time (Total)
Piecewise-Linear	1	0.0001	0.0	115	5.58%	35.0 min.
Piecewise-Linear	2	0.0001	0.0	69	3.35%	3.4 min.
Blobby	2	0.02	0.01	6,166	36.1%	85.6 min.
Blunt Fin	2	0.0001	0.0	419,297	93.2%	684 min.
Blunt Fin	2	0.001	0.0	297,237	66.1%	1,243 min.
Blunt Fin	2	0.01	0.0	160,948	35.8%	1,866 min.
Blunt Fin	2	0.1	0.0	72,362	16.1%	2,557 min.

5.1 Algorithm 1

Algorithm 1 is based on a priority queue, where the tetrahedra are ordered by their influence on the error. For each tetrahedron T, we calculate δ_T, as defined in (4), reflecting the predicted error increase resulting from collapsing an edge of the tetrahedron. To specify δ_T, we must calculate the cost of collapsing each edge of the tetrahedron and select that edge that yields the minimum error. We insert tetrahedra in a priority queue that is ordered by increasing $\epsilon_T + \delta_T$ values.

Having selected a maximum error threshold ϵ used to terminate the algorithm, we iterate over the following steps:
remove a tetrahedron T from the queue;

- if $\epsilon_T + \delta_T > \epsilon$, then the set of tetrahedra in the queue represents the simplified mesh;
- if $\epsilon_T + \delta_T \leq \epsilon$, then
  - **collapse** the selected edge e of the tetrahedron T and eliminate the edge neighbors of T from the priority queue;
  - **recalculate** ϵ_{T^C} and δ_{T^C} for each tetrahedron T^C in the stencil of e—these are the tetrahedra modified by the collapse operation;
  - **recalculate** δ_{T_E} for each tetrahedron T_E in the extended stencil of e—the error bounds of these tetrahedra have been changed by the collapse of e;

Three methods can be implemented that utilize the collapse operation to generate sets of meshes $\mathcal{M}_0, \mathcal{M}_1, ..., \mathcal{M}_n$ approximating the original mesh at multiple levels of resolution:

1. Let the initial mesh be $\mathcal{M}_0$. Choose a sequence of error bounds $\epsilon_1 < \epsilon_2 < ... < \epsilon_n$. 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 violates the condition $\epsilon_T + \delta_T \leq \epsilon_1$. The set of tetrahedra in the priority queue define the mesh $\mathcal{M}_1$.

   Using the mesh $\mathcal{M}_1$, collapse the tetrahedra in the queue until the tetrahedron T at the front of the queue violates the condition $\epsilon_T + \delta_T \leq \epsilon_2$. The set of tetrahedra in the queue defines the mesh $\mathcal{M}_2$.

   The algorithm terminates when mesh $\mathcal{M}_n$ is generated.
2. Specify a number (or a percentage) of tetrahedra to be collapsed in each step. The intermediate meshes are defined by those tetrahedra remaining in the

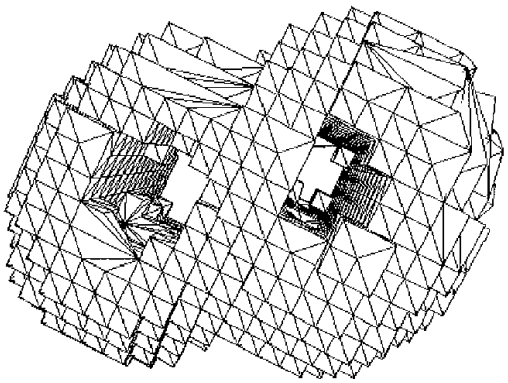


Fig. 14. The boundary of the blobby data set together with its tetrahedra. The data set was initially generated as a rectilinear data set. The boundary of the set was generated by eliminating all tetrahedra whose scalar values were zero at the vertices. The data set contains 17,076 tetrahedra.

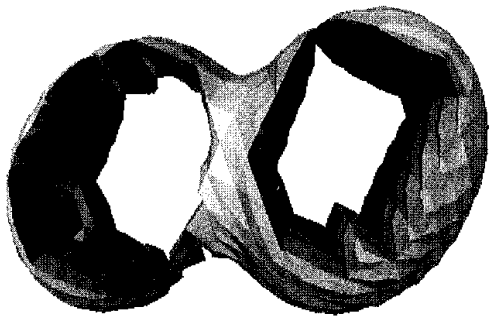


Fig. 15. An isosurface of the original blobby data set.

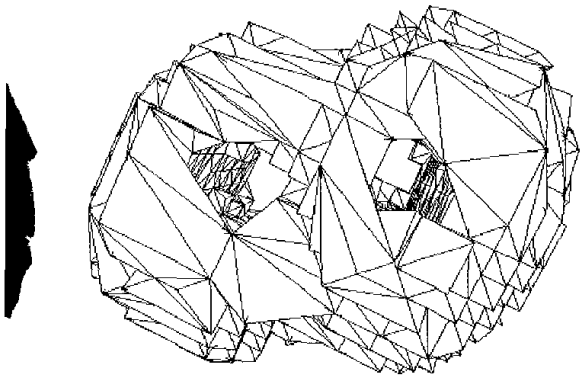


Fig. 16. The simplified blobby data set. This data set contains 6,166 tetrahedra or 36.1 percent of the original number. We note that some boundary simplification has taken place.

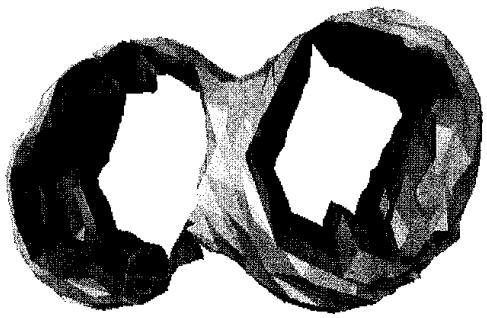


Fig. 17. An isosurface of the simplified data set using the same isovalue as in Fig. 15.

queue after each step. The error for each mesh is stored.

3. Similarly to Gieng et al. [15], 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 in parallel must not intersect.

The resulting sequence of meshes can be used for "smooth" transitions (geomorphs) between mesh levels.

This algorithm attempts to select the tetrahedron and edge that produces the minimal increase in error at each step. A problem with the algorithm is that the increase in error must be calculated a priori. This causes the initial setup step to be computationally intensive, as the error calculations can be time-consuming; most of these calculations, in the end, are not necessary. When we collapse an edge of a tetrahedron, all the edge neighbors are removed from the mesh. For each of these neighbors, the calculation of δ_T is only used in the initial ordering.

5.2 Algorithm 2

Algorithm 1 is a robust simplification algorithm, but is time-intensive. To improve the efficiency of the algorithm, we use a greedy strategy which examines each tetrahedron T individually. Algorithm 2 attempts the collapse of an edge of T and accepts the collapse if the resulting error is within the required tolerance and rejects the collapse otherwise.

For a mesh $\mathcal{M}$, we make passes through the tetrahedra of the mesh, collapsing edges if the error bound is not exceeded. We continue to process tetrahedra in this way until we cannot guarantee that further collapse operations will exceed the error bound. Algorithm 2 proceeds as follows:

Given a mesh $\mathcal{M}$ and an error tolerance ϵ, perform the following steps:

1. Select a tetrahedron T of mesh $\mathcal{M}$ and an edge e of T;
2. Attempt to collapse e; if the collapse induces an error threshold below ϵ, accept the collapse, and remove the tetrahedron T and all edge neighbors of e from the mesh;
3. A successful collapse modifies some tetrahedra in the stencil of the collapsed edge e—for each modified tetrahedron T_C, calculate the error change and add this to the accumulated error ϵ_{T_C} for this tetrahedron;
4. Continue until the simplified mesh cannot be collapsed any further without increasing the error above ϵ.

This algorithm selects an edge of an arbitrary tetrahedron and attempts to collapse it. If the collapse results in a mesh with an error below the threshold, it is accepted and the process continues. This approach avoids the complex δ_T calculations for tetrahedra of Algorithm 1. Here, the δ_T calculations are not necessary. Thus, we save the a priori calculation of the δ_T values for all tetrahedra in the mesh and save the calculation of δ_T for those tetrahedra in the extended stencil of a collapsed edge e.

We can use this algorithm to produce a sequence of meshes. The input is a sequence of error tolerances $\epsilon_1 < \epsilon_2 < \ldots < \epsilon_n$, and the algorithm generates a sequence of meshes $\mathcal{M}_0, \mathcal{M}_1, ..., \mathcal{M}_n$. Here, $\mathcal{M}_0$ is the original mesh, and each mesh $\mathcal{M}_i$ has a deviation from the original mesh that is less than ϵ_i. If we add the condition that an edge e is rejected for collapse if the stencil of e overlaps with the extended stencils of previously collapsed edges in the mesh, then the sequence of meshes will the property that the mesh $\mathcal{M}_{i+1}$ can be constructed from mesh $\mathcal{M}_i$ by collapsing a set of edges in a geometrically continuous fashion, see Gieng et al. [15].

However, we can improve this algorithm by saving the induced error change δ_T when a tetrahedron collapse is attempted. Given an initial error bound ϵ, we can use the greedy strategy to choose tetrahedra and edges for collapse. If we find that a collapse is not successful, i.e., the resulting error of collapsing any edge of the tetrahedron is greater than ϵ, then we retain δ_T for this tetrahedron. In subsequent passes through the mesh, the δ_T values provide estimates of the cost of collapsing this tetrahedron. If the sum of the estimated cost and the accumulated error is greater than the prescribed bound, we do not attempt to collapse T. If the tetrahedron is stretched by another collapse operation, the estimate is eliminated and we must attempt the collapse operation to determine the "collapsibility" of T.

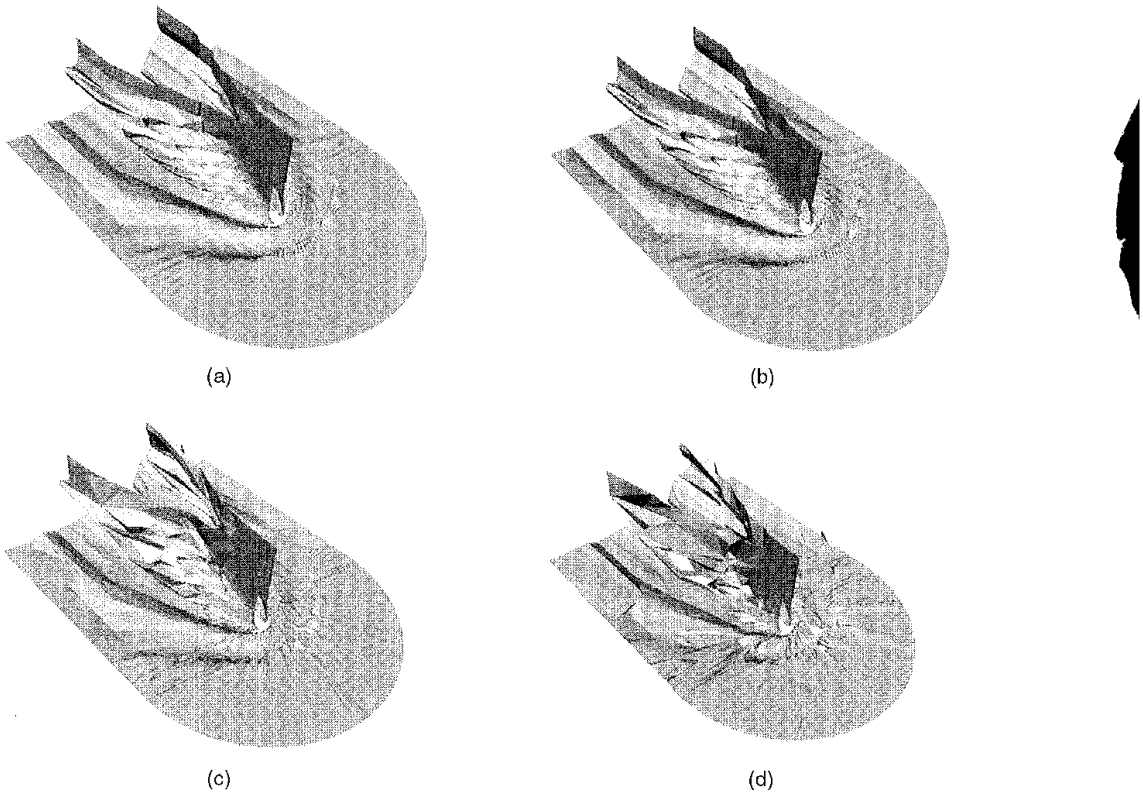


Fig. 18. Isosurfaces of the blunt-fin data set. (a) was generated from the original data set. (b)-(d) were generated from simplified data sets containing 93 percent, 35 percent, and 16 percent of the number of tetrahedra in the original data set.

After the initial pass through the mesh, we can insert the tetrahedra into a priority queue ordered by increasing values of δ_T. On subsequent passes through the mesh, we utilize the ordering in the queue to determine a sequence of tetrahedra to be examined. If, for a tetrahedron T in the queue, the predicted increase in error causes the total error to exceed the threshold, we can terminate the search for new collapses and move to the next mesh level.

6 Implementation Issues

We have implemented the two algorithms using a simple data structure for tetrahedral meshes. We store a list of vertices and a list of tetrahedra. Each tetrahedron contains links to the four vertices of the tetrahedron and the four face neighbors. Calculating the vertex neighbors, the edge neighbors, and the face neighbors of a tetrahedron is straightforward using this data structure. Each tetrahedron T carries an error bound ϵ_T and a predicted deviation δ_T.

We have found that care must be taken to ensure that the topology of the mesh is not compromised. Locally, an edge collapse can cause a tetrahedron in the stencil of the edge to flip, causing the the tetrahedra to intersect in this area. We detect these changes by examining the signs of the volumes of the tetrahedra in the stencil of the collapsed edge. If, due to the collapse of the edge, the volume of such a tetrahedron changes sign, then the collapse is rejected.

Care must also be taken when collapsing tetrahedra on the boundary of a mesh. If the maximum error bounds are too large, inconsistencies and intersections on the boundary may occur. Staadt and Gross [9] discuss these artifacts for the general case. We do the following: If the boundary error tolerance is set to zero, the degeneracies cannot occur (as long as we monitor the volume of the stretched tetrahedra as discussed in the paragraph above). With a zero boundary tolerance, collapse operations on the boundary would only take place on the portions of the boundary that are planar.

7 Results

Results of our work are shown in Figs. 10, 11, 12, 13, 14, 15, 16, 17, 18, 19. We have generated examples for three data sets: a simple piecewise-linear function, a simple blobby model, and the blunt-fin. We utilized both Algorithm 1 and Algorithm 2 for the linear function to demonstrate the speed of Algorithm 2. We found for the blobby model and for the blunt-fin that Algorithm 1 was not competitive and so the results are given only for Algorithm 2.

Our first example is shown in Figs. 10, 11, 12, 13. Fig. 10 shows a piecewise-linear scalar field over a cube containing 2,058 tetrahedra. Each voxel of the original $7 \times 7 \times 7$ data set is initially split into six tetrahedra, see [21]. The piecewise-linear function is defined by

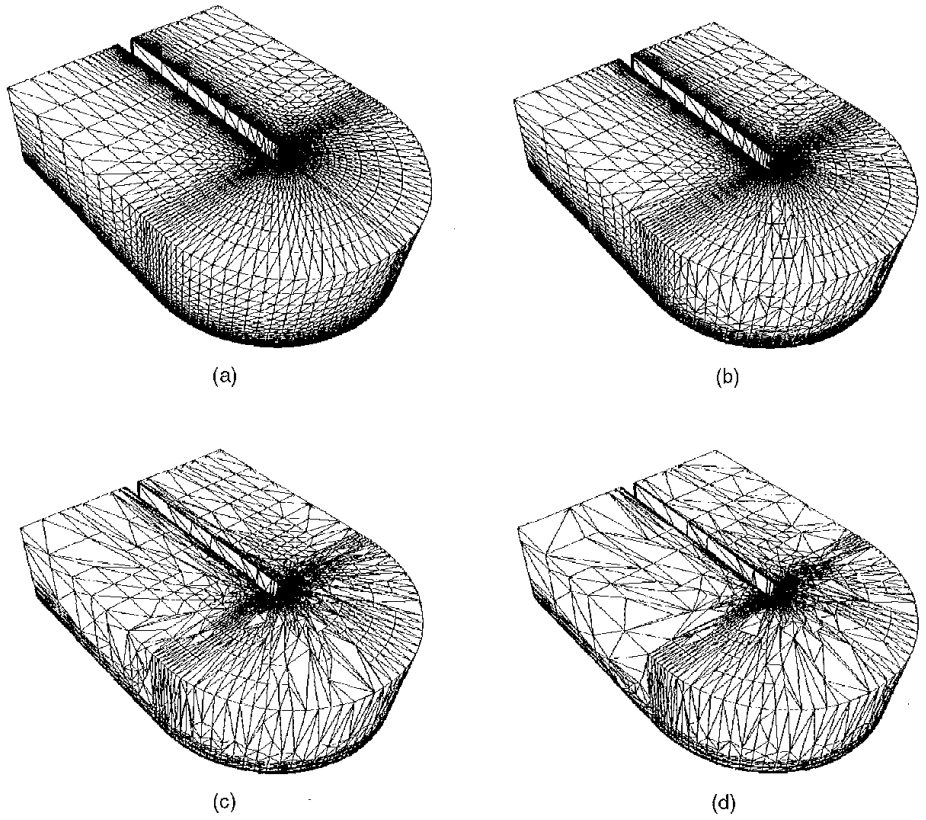


Fig. 19. The boundary surface of the blunt-fin data set. (a) was generated from the original data set. (b)-(d) were generated from the simplified data sets of Fig. 18. The boundary error tolerance was zero. Note that the simplification only takes place on those areas of the boundary that lie on a plane.

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

Fig. 11 shows the underlying tetrahedral mesh for this example.

Simplifying the mesh with Algorithm 1, and any small user-specified error threshold, yields the mesh shown in Fig. 12, which contains 115 tetrahedra and still represents the scalar field exactly. (We note that 15 tetrahedra would be optimal.) Simplifying the mesh with Algorithm 2, and any small error threshold, yields the mesh shown in Fig. 13. The resulting mesh contains 69 tetrahedra and represents the scalar field exactly.

The statistics for the operation of Algorithm 1 and Algorithm 2 are reported in Table 1.

Our second example is the isosurface represented by a blobby model, see Wyvill et al. [22]. This model is represented by a number of positive and negative blobby elements (negative to create the holes) using a linear superposition of functions of the form

$$C(r) = \begin{cases} -49(rR)^6 + 179(rR)^4 - 229(rR)^2 + 1 & \text{if } 0 \le r \le R \\ 0 & \text{if } R < r, \end{cases}$$

where R is the radius of the blob and r is the distance of the point from the center of the blob.

We first created a rectilinear data set surrounding the blobby model and sampled the blobby data set at the vertices of the rectilinear set. To represent an interesting boundary case, we then removed the tetrahedra that contained only zero values of the function. The resulting data set contains 17,076 tetrahedra and is shown in Fig. 14. An isosurface generated from the data set is shown in Fig. 15.

We applied Algorithm 2 to this data set, using an error threshold of $\epsilon = 0.1$ and a boundary error threshold of $\epsilon_B = 0.1$. The results are shown in Fig. 16 and Fig. 17. After simplification, the data set contains 6,803 tetrahedra.

The third example is the well-known blunt-fin data set. This curvilinear scalar field has been generated from the original blunt-fin vector field by using the magnitude of the velocity vector at each grid point. To generate the tetrahedral grid, each grid cell was split into six tetrahedra. We utilized Algorithm 2 to simplify the data set with increasing error tolerances on the interior of the mesh, and a zero error tolerance on the mesh boundary.

Fig. 18 illustrates an isosurface generated for various levels of simplification. The original blunt-fin data set was used to generate the isosurface shown in Fig. 18a. This data set contains 449,748 tetrahedra. The data set used for the isosurface shown in Fig. 18b was generated with a error tolerance of 0.0001 and contains 419,297 tetrahedra. The data set used for the isosurface shown in Fig. 18c was

generated with a error tolerance of 0.01 and contains 160,948 tetrahedra. The data set used for the isosurface shown in Fig. 18d was generated with an error tolerance of 0.1 and contains 72,362 tetrahedra. This data set contains only 16.1 percent of the number of the tetrahedra in the original data set.

Fig. 19 illustrates the boundary of the blunt-fin data set and the tetrahedra of the mesh for each of the respective error tolerances shown in Fig. 18. With the zero boundary tolerance, we see that boundary edge collapses only occur in areas where the boundary surface is planar.

Table 1 gives the statistics for the operation of the algorithms on each of the data sets. The images were all generated on an SGI Onyx 2, with 512 MB of RAM, using a single 195 Mz R10000 processor.

8 Conclusions

We have presented a method for the simplification of tetrahedral meshes approximating a trivariate function. The simplification of a mesh is based on a tetrahedral collapse algorithm and local error calculations that ensure that the mesh remains within a user-specified tolerance of the original one. Several methods can be applied to generate various mesh hierarchies to be used for level-of-detail visualization. This is a powerful tool for the hierarchical representation of massive three-dimensional data sets defined on 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 [23]. Furthermore, we intend to extend and apply our algorithm to vector fields and time-varying fields. We also can extend this algorithm to include sign information in the error bound, which has been done previously for surface meshes by Bajaj and Schikore [17].

Acknowledgments

This work was supported by the U.S. National Science Foundation under contract ACI 9624034 (CAREER Award), the U.S. Office of Naval Research under contract N00014-97-1-0222, the U.S. Army Research Office under contract ARO 36598-MA-RIP, the NASA Ames Research Center through an NRA award under contract NAG2-1216, the Lawrence Livermore National Laboratory through an ASCI ASAP Level-2 under contract W-7405-ENG-48 (and B335358, B347878), and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of Silicon Graphics, Inc.

The blunt-fin data set is courtesy of NASA Ames Research Center.

We would like to thank Moritz Voelker, Greg Schussman, David Wiley, and 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] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen, "Decimation of Triangle Meshes," *Computer Graphics (SIGGRAPH '92 Proc.)*, E.E. Catmull, ed., vol. 26, pp. 65-70, July 1992.

[2] K.J. Renze and J.H. Oliver, "Generalized Unstructured Decimation," *IEEE Computer Graphics and Applications*, vol. 16, no. 6, pp. 24-32, Nov. 1996.

[3] H. Hoppe, "Progressive Meshes," *SIGGRAPH 96 Conf. Proc.*, H. Rushmeier, ed., pp. 99-108, Aug. 1996.

[4] H. Hoppe, "View-Dependent Refinement of Progressive Meshes," *SIGGRAPH 97 Conf. Proc.*, T. Whitted, ed., pp. 189-198, Aug. 1997.

[5] J. Popovic and H. Hoppe, "Progressive Simplicial Complexes," *SIGGRAPH 97 Conf. Proc.*, T. Whitted, ed., pp. 217-224, Aug. 1997.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," *Computer Graphics (SIGGRAPH '93 Proc.)*, J.T. Kajiya, ed., vol. 27, pp. 19-26, Aug. 1993.

[7] J.C. Xia and A. Varshney, "Dynamic View-Dependent Simplification for Polygonal Models," *Proc. IEEE Visualization '96*, pp. 327-334, Oct. 1996.

[8] M. Garland and P.S. Heckbert, "Surface Simplification Using Quadric Error Metrics," *SIGGRAPH 97 Conf. Proc.*, T. Whitted, ed., pp. 209-216, Aug. 1997.

[9] O.G. Staadt and M.H. Gross, "Progressive Tetrahedralizations," *Proc. Visualization 98*, D.Ebert, H. Hagen, and H. Rushmeier, eds., pp. 397-402, Oct. 1998.

[10] P. Cignoni, L. De Floriani, C. Montoni, E. Puppo, and R. Scopigno, "Multiresolution Modeling and Visualization of Volume Data Based on Simplicial Complexes," *Proc. 1994 Symp. Volume Visualization*, A. Kaufman and W. Krueger, eds., pp. 19-26, Oct. 1994.

[11] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno, "Multiresolution Representation and Visualization of Volume Data," *IEEE Trans. Visualization and Computer Graphics*, vol. 3, no. 4, pp. 352-369, 1997.

[12] B. Hamann, "A Data Reduction Scheme for Triangulated Surfaces," *Computer Aided Geometric Design*, vol. 11, pp. 197-214, 1994.

[13] B. Hamann and J.-L. Chen, "Data Point Selection for Piecewise Linear Curve Approximation," *Computer-Aided Geometric Design*, vol. 11, no. 3, pp. 289-301, June 1994.

[14] T.S. Gieng, B. Hamann, K.I. Joy, G.L. Schussman, and I.J. Trotts, "Smooth Hierarchical Surface Triangulations," *Proc. Visualization '97*, H. Hagen and R. Yagel, eds., pp. 379-386, Oct. 1997.

[15] T.S. Gieng, B. Hamann, K.I. Joy, G.L. Schussman, and I.J. Trotts, "Constructing Hierarchies for Triangle Meshes," *IEEE Trans. Visualization and Computer Graphics*, vol. 4, no. 2, pp. 145-161, 1998.

[16] J. Cohen, D. Manocha, and M. Olano, "Simplifying Polygonal Models Using Successive Mappings," *Proc. IEEE Visualization '97*, R. Yagel and H. Hagen, eds., pp. 395-402, Nov. 1997.

[17] C.L. Bajaj and D.R. Schikore, "Topology Preserving Data Simplification with Error Bounds," *Computers and Graphics*, vol. 22, no. 1, pp. 3-12, 1998.

[18] H. Hoppe, "Smooth View-Dependent Level-of-Detail Control and Its Application to Terrain Rendering," *Proc. Visualization 98*, D. Ebert, H. Hagen, and H. Rushmeier, eds., pp. 35-42, Oct. 1998.

[19] R. Klein, G. Liebich, and W. Straßer, "Mesh Reduction with Error Control," *Proc. IEEE Visualization '96*, pp. 311-318, Oct. 1996.

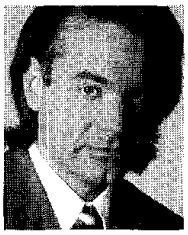
[20] I.J. Trotts, B. Hamann, K.I. Joy, and D.F. Wiley, "Simplification of Tetrahedral Meshes," *Proc. Visualization 98*, D. Ebert, H. Hagen, and H. Rushmeier, eds., pp. 287-296, Oct. 1998.

[21] *Scientific Visualization: Overviews, Methodologies, and Techniques*, G. Nielson, H. Müller, and H. Hagen, eds. Academic Press, 1997.

[22] B. Wyvill, C. McPheeters, and G. Wyvill, "Animating of Soft Objects," *The Visual Computer*, vol. 2, no. 4, pp. 235-242, 1986.

[23] B. Hamann and B. Jordan, "Triangulations from Repeated Bisection," *Mathematical Methods for Curves and Surfaces II*, M. Dæhlen, T. Lyche, and L.L. Schumacker, eds., pp. 229-236. Nashville, Tenn.: Vanderbilt Univ. Press, 1998.

Issac Trotts is pursuing a BS in mathematics from the University of California, Davis. His research interests are computer graphics, visualization, and computational neuroscience. He is a member of the ACM.



Bernd Hamann received a BS in computer science, a BS in mathematics, and an MS in computer science from the Technical University of Braunschweig, Germany. He received his PhD in computer science from Arizona State University in 1991 under the supervision of Gregory M. Nielson. He is an associate professor of computer science and co-director of the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis and an adjunct professor of computer science at Mississippi State University. He collaborates as a participating guest with the Lawrence Livermore National Laboratory. From 1991 to 1995, Dr. Hamann was a professor in the Department of Computer Science at Mississippi State University and a research faculty member at the U.S. National Science Foundation (NSF) Engineering Research Center for Computational Field Simulation. He joined the University of California, Davis, in 1995. His main research interests are visualization, geometric modeling/computer-aided geometric design (CAGD), computer graphics, and immersive visualization environments. His current reserach focuses on hierarchical representation methods and visualization techniques for very large data sets. He is the author or coauthor of more then 60 revied publications and has given presentations, often by invitation, at leading conferences and workshops in the U.S. and Europe.

Dr. Hamann was awarded a 1992 Research Initiation Award by Mississippi State University, a 1992 Research Initiation Award by the NSF, and a 1996 CAREER Award by the NSF. In 1995, he received a Hearin-Hess Distinguished Professorship in Engineering from the College of Engineering, Mississippi State University. In 1998, he was nominated for an Alan T. Waterman Award, which is awarded annually by the NSF to one outstanding young researcher. Dr. Hamann served on the Editorial Board of the *IEEE Transactions on Visualization and Computer Graphics* and is a paper co-chair/proceedings co-editor for the IEEE Visualization 1999 and Visualization 2000 conferences. Hamann is a member of the ACM, the IEEE, SIAM, and the IEEE Technical Committee on Visualization and Graphics.

Kenneth I. Joy received a BA and and MA in mathematics from the University of California Los Angeles, and a PhD in mathematics from the University of Colorado in 1976. He is an associate professor of computer science at the University of California at Davis. He came to UC Davis in 1980 in the Department of Mathematics and was a founding member of the Computer Science Department in 1983. His primary interests are visualization, geometric modeling, and computer graphics, focusing on multiresolution methods. Dr. Joy is a member of the ACM, the IEEE, and SIAM.