EUROGRAPHICS 2003 / P. Brunet and D. Fellner
(Guest Editors)

Volume 22 (2003), Number 3

Progressive Simplification of Tetrahedral Meshes Preserving All Isosurface Topologies

Yi-Jen Chiang[†] and Xiang Lu[‡]

Department of Computer and Information Science, Polytechnic University, Brooklyn, NY 11201, USA

Abstract

In this paper, we propose a novel technique for constructing multiple levels of a tetrahedral volume dataset while preserving the topologies *of all* isosurfaces *embedded in the data. Our simplification technique has two major phases. In the* segmentation *phase, we segment the volume data into* topological-equivalence regions*, that is, the sub-volumes within each of which all isosurfaces have the same topology. In the* simplification *phase, we simplify each topological-equivalence region independently, one by one, by collapsing edges from the smallest to the largest errors (within the user-specified error tolerance, for a given error metrics), and ensure that we do not collapse edges that may cause an isosurface-topology change. We also avoid creating a tetrahedral cell of negative volume (i.e., avoid the* fold-over *problem). In this way, we guarantee to preserve all isosurface topologies in the entire simplification process, with a controlled geometric error bound. Our method also involves several additional novel ideas, including using the* Morse theory *and the* implicit fully augmented contour tree*, identifying types of edges that are not allowed to be collapsed, and developing efficient techniques to avoid many unnecessary or expensive checkings, all in an integrated manner. The experiments show that all the resulting isosurfaces preserve the topologies, and have good accuracies in their geometric shapes. Moreover, we obtain nice data-reduction rates, with competitively fast running times.*

1. Introduction

In this paper, we propose a novel technique for constructing multiple levels of a tetrahedral volume dataset while preserving the *topologies* of all *isosurfaces* embedded in the data. Isosurfaces are surfaces of equal scalar value, and displaying isosurfaces is one of the most powerful techniques for the investigation of volume datasets. It has been used extensively in scientific visualization applications such as biology, medicine, chemistry, computational fluid dynamics, and so on. As the size of the datasets increases dramatically in recent years, the need for multiresolution methods becomes apparent when such large datasets have to be visualized interactively. As we create different levels of details by simplifying the volume dataset, it is extremely important to have each level of the data still capturing the features of the original data. One of the most critical such features is the *topologies* of all isosurfaces. Therefore we want to preserve the exact topologies of all isosurfaces, and control the geometric accuracy of the volume data and isosurfaces by the user-specified error tolerance in a given error metrics.

One might argue that we could achieve the same effect of preserving isosurface topologies by applying a simplification approach without such guarantee, and adjusting the error tolerance back and forth to find the right one to use. However, such try-and-error process involving user inspection lacks the correctness guarantee and is very time-consuming even for just one single isosurface, and in typical scientific datasets there are hundreds or thousands of *critical isovalues* to test where the isosurface changes the topology, making such approach infeasible. Therefore it is desirable to develop a simplification algorithm that automatically guarantees the preservation of all isosurface topologies.

A *scalar volume dataset* consists of tuples $(v, F(v))$, where v is a 3D sample point and F is a scalar function

---

[†] yjc@poly.edu. Research supported in part by NSF CAREER Grant CCR-0093373, NSF Grant ACI-0118915, and NSF ITR Grant CCR-0081964.

[‡] xlu@photon.poly.edu. Research supported by NSF Grant CCR-0093373.

defined over 3D points. We focus on *irregular-grid* volume data represented as tetrahedral meshes. This is the most general class of volumetric data and has been proposed as an effective means of representing disparate field data that arises in a broad spectrum of scientific applications including structural mechanics, computational fluid dynamics, partial differential equation solvers, shock physics, and so on.

Our simplification technique has a theoretical foundation that guarantees the preservation of all isosurface topologies. Moreover, it is based on the edge-collapse operation, which has been widely used as it typically produces very high simplification quality [6]. Previously, there were tetrahedral mesh simplification methods that preserve the topology of the *volume* itself [9, 6] (rather than the *isosurface* topologies), and volume simplification method preserving isosurface topologies that works only for *regular grids* [10]. To the best of our knowledge, our technique is the first one that simplifies *irregular grids* while preserving all isosurface topologies (and therefore also the first such technique for *rectilinear* and *curvilinear grids*, by tetrahedralizing the datasets as is done in [10]).

Our algorithm makes use of the *Morse theory* [2, 11, 14]. One important notion is that of *critical points*, which are the points where the isosurface topology changes (i.e., an isosurface connected component appears/disappears, changes genus, splits to more components, or components merge) as we continuously change the isovalue. In the simplification process, we need to make sure that the isosurface topological features are preserved. More specifically, (1) we do not incorrectly reduce the isosurface topological features, and (2) we do not incorrectly increase the isosurface topological features by introducing new critical points. It is very important to note that a simple local rule of preserving the existing critical points (e.g., not collapsing edges with a critical endpoint) does *not* achieve (1): if we collapse an edge with two *non-critical endpoints* that *connects two connected components* of an isosurface, then even though the set of critical points stays the same, we would still incorrectly merge the two components into one; similar considerations apply to genus changes (where we might incorrectly close a hole in an isosurface). Therefore we need a more delicate technique, considering the problem globally, to achieve both (1) and (2).

Our technique is an edge-collapse simplification algorithm that disallows the collapses if they cause an isosurface-topology change. To check the collapsibility, we introduce two major classes of tests. The first class checks if collapsing an edge will join two regions of different isosurface topologies, and the second class checks if critical points will be removed or created by the collapse. To facilitate these checkings, our algorithm has two phases. In the *segmentation* phase, we segment the volume into *topological-equivalence regions* (*top-eq regions* for short), that is, the sub-volumes within each of which all isosurfaces have the same topology. In the *simplification* phase, we simplify each top-eq region independently, one by one, by collapsing edges from the smallest to the largest errors (within the user-specified error tolerance, for a given error metrics) when they pass the collapsibility checkings. We also avoid creating a tetrahedral cell of negative volume (i.e., avoid the *fold-over* problem). In this way, we achieve both (1) and (2) above and preserve all isosurface topologies in the entire simplification process, with a controlled geometric error bound.

Our method also involves several additional novel ideas. For the segmentation phase, to identify top-eq regions, we make a novel use of the *contour tree* [19, 16, 3], which was previously proposed to speed up isosurface extraction [19]. We enhance the ordinary contour tree by proposing the use of an *implicit fully augmented contour tree*, which captures all events of isosurface topology changes and can be computed *implicitly* by a labeling scheme. For the simplification phase, we identify types of edges that are not allowed to be collapsed, prove a nice property that enables us to replace an expensive checking with a fast checking, and develop an algorithm that avoids many unnecessary checkings—all these are integrated into an efficient algorithm.

Our experiments demonstrate the effectiveness of our technique. The resulting isosurfaces preserve the topologies, and have good accuracies in their geometric shapes. Moreover, we obtain nice data-reduction rates (i.e., percentages of the *removed* cells) of about 48–89% for most of our tested datasets when simplifying to the maximum extent (while preserving isosurface topologies), with competitively fast running times.

2. Previous Work

2.1. Volume Simplification

The multiresolution paradigm is one of the most powerful techniques developed in graphics and scientific visualization in recent years; it has been widely applied to both 3D polygonal models and volume datasets. Tetrahedral simplification based on edge collapses were presented in [18, 15, 17]; these methods do not consider preserving any topological structure in the data. Dey et. al. [9] proposed very nice techniques for simplification of triangle meshes and tetrahedral meshes using edge collapses while preserving the topology of the *meshes themselves*; their methods were later adopted in [6] for tetrahedral volume simplification. Gerstner and Pajarola [10] proposed a volume simplification method that preserves the isosurface topologies, based on tetrahedralizing regular grids and then merging tetrahedra which is in reverse order of bisecting tetrahedra starting from the six tetrahedra of the bounding cube. The method takes advantage of the regularity of the geometry of the data points, and is difficult to extend to rectilinear, curvilinear, or irregular grids. Recently Chopra and Meyer [5] proposed a very fast tetrahedral volume simplification algorithm based on collapsing a tetrahedral cell into a vertex as an atomic operation.

We remark that Wood *et. al.* [21] developed an algorithm that operates on a regular-grid volume representation to simplify the isosurface topology, in the context of *polygonal model acquisition* in 3D graphics. A polygonal model is often obtained by creating a regular grid using laser range scanners (and merging the scanned data into a volumetric grid), CT, or MRI, and then performing isosurface extraction. In the process, the isosurface may contain many artifacts in the form of tiny topological handles. Their approach is to reduce these artifacts by removing handles with size smaller than a given threshold. While their technique works nicely in this setting, the problem is completely different from ours: there is only *one* isosurface (i.e., the zero-isosurface, which is the underlying polygonal model), and the isosurface has only *one* connected component. Our technique is mainly for the scientific visualization applications, where the volume data may contain an arbitrary number of isosurfaces and each isosurface may have an arbitrary number of connected components and genus, and we want to simplify the volume while maintaining the accuracy of the embedded information as much as possible.

2.2. Contour Tree

The *contour tree* is a fundamental data structure that represents the relations between connected components of the isosurfaces embedded in a volume dataset. Two connected components that merge together (as one continuously changes the isovalue) are represented as two edges that join at a node of the tree. This structure was used by van Kreveld *et. al.* [19] to speed up isosurface extraction in the *seed-cell propagation* paradigm. The succinct encoding of the isosurface topologies in the contour tree also leads to other important applications. Bajaj *et. al.* [1] proposed the display of the contour tree to provide the user with insights into the topological structures of the isosurfaces embedded in the volume data. Recently, Pascucci and Cole-McLaughlin [12] gave an elegant algorithm of computing and associating the Betti numbers $(\beta_0, \beta_1, \beta_2)$ with the contour tree so that the isosurface topologies, including the number of connected components and the genus number, can be completely determined and displayed. Very recently, Weber et. al. [20] gave a technique for capturing isosurface topologies based on detecting *critical regions*. We remark that the contour tree of Pascucci and Cole-McLaughlin [12] is more powerful than an ordinary contour tree, and could serve for our purpose of volume segmentation. However, in our application we do not need the Betti numbers, and our *implicit fully augmented contour tree* is easier to compute. Moreover, some procedure for computing our tree can later be used in the simplification process as well.

Incrementally more and more efficient algorithms for computing contour trees were proposed by de Berg and van Kreveld [8], van Kreveld *et. al.* [19], Tarasov and Vyalyi [16], and Carr *et. al.* [3]. Carr *et. al.* [3] simplified and extended the method of Tarasov and Vyalyi [16] so that the contour tree can be efficiently computed in any dimension. This algorithm [3] is simple and elegant, and is the basis for computing our implicit fully augmented contour tree. We review the algorithm [3] in Section 3.1.2. Recently, Pascucci and Cole-McLaughlin [12] gave the first *output-sensitive* algorithm for computing the contour tree, and Chiang et. al. [4] proposed a simple and *optimal output-sensitive* algorithm for computing the contour tree in any dimension.

3. Our Approach

In this section, we present our tetrahedral-mesh simplification technique. There are two major phases. In the *segmentation* phase, we segment the volume data into *top-eq regions*. In the *simplification* phase, we simplify each top-eq region independently, by collapsing edges from the smallest to the largest errors, and ensure that we do not collapse edges that may cause an isosurface-topology change or create a tetrahedral cell of negative volume. We describe the details in the following sections.

3.1. The Segmentation Phase

In this phase, we first compute the *fully augmented contour tree* T, which contains all vertices in the volume and can capture all topological changes of all isosurfaces in the volume. The tree T can be efficiently and *implicitly* computed by a labeling scheme. We then use T to identify and label all vertices into top-eq regions, within each of which the isosurfaces all have the same topology. The vertex labels define the top-eq region(s) each cell belongs to (possibly to more than one such region), which effectively segments the volume into top-eq regions implicitly.

Before describing the fully augmented contour tree T, we first review some background in the *Morse theory* [2, 11, 14]. Recall that *critical points* are the points where the isosurface topology changes as we continuously change the isovalue. There are three types of critical points: minimum, maximum, and saddle points. A vertex with scalar value *smaller* (resp. *larger*) than those of all its neighboring vertices is called a *minimum* (resp. *maximum*), and corresponds to the *appearing* (resp. *disappearing*) of an isosurface connected component if we change the isovalue continuously from $-\infty$ to ∞. A *saddle point* is the remaining type of critical point, and corresponds to an isosurface connected component split/merge or a change of genus number. Morse theory requires that the critical points occur at distinct points and values. A function satisfying this condition is called a *Morse function*.

Recall that the input data consists of tuples $(v, F(v))$. Since a tetrahedral mesh is a simplicial complex (in 3D), we naturally choose the scalar function F to be a piecewise-linear function whose value at a *sample point* v (which is a vertex of the mesh) is the scalar data value $F(v)$ given in the tuple $(v, F(v))$, and whose value at a point *within a simplex* is obtained by linear interpolation from the scalar values of the simplex vertices. Moreover, by assuming that the scalar values at vertices are distinct (achieved by symbolic perturbation), we ensure that the function F is a *Morse function*, and that the critical points occur at vertices of the mesh [2].

The *contour tree* of a Morse function is a graph in which (1) each leaf node represents a minimum or a maximum critical point, at which an isosurface connected component appears or disappears, (2) each interior node represents the joining and/or splitting of two or more isosurface connected components at a critical point, which is necessarily a saddle point, and (3) each edge represents a connected component in the isosurfaces at all isovalues between the scalar values of the two endpoint nodes of the edge. This graph is shown to be a tree [19], hence called a *contour tree*. The tree nodes are sorted by the scalar values from the smallest to the largest, organized from bottom to top (see Fig. 1(a)). Note that such an ordinary contour tree does not capture the *genus change* within the same isosurface connected component, which is the only type of isosurface-topology-change event that is not encoded. Such event corresponds to a saddle point that causes a genus change but not a component change.

To enhance the ordinary contour tree so that all isosurface topologies are captured, we insert the genus-change-only saddle points into their corresponding isosurface connected components, i.e., each edge in the contour tree (corresponding to an isosurface component) is further divided into *segments* by the genus-change-only saddle points in that component, where all the points are again sorted by the scalar values from bottom to top (see Fig. 1(b)). Observe that now each *segment* in the resulting tree defines a *top-eq region*, meaning that between the scalar values of the two endpoints of each segment, there is no isosurface topology change. Finally, we insert all *non-critical* vertices to their corresponding isosurface connected components, and again all points are sorted globally by their scalar values. The sorted order of scalar values naturally places the non-critical vertices into the top-eq regions they belong to, and the resulting tree is our *fully augmented contour tree* T (see Fig. 1(c)). Note that in T, each segment contains only non-critical points in the interior, and two critical points at the two endpoints.

To compute our fully augmented contour tree T, we observe that the contour-tree algorithm of Carr *et. al.* [3] readily computes an ordinary contour tree, with all vertices (including non-critical vertices and genus-change-only saddle points) presented in the tree, in sorted order, in their corresponding contour-tree edges (later, the vertices that are not endpoints of the contour tree edges are removed). In other words, the algorithm readily computes our fully augmented contour tree T; the only information that is missing is the distinction between genus-change-only saddle points and non-critical points. Our solution is simple: we first *classify* all vertices into critical and non-critical points, and then apply the algorithm of Carr *et. al.* [3]. The result is our desired tree T. We actually simplify the algorithm [3] by computing T *implicitly* by a labeling scheme in the last stage. Finally, we use T to segment the volume. In summary, our segmentation algorithm consists of the following steps.

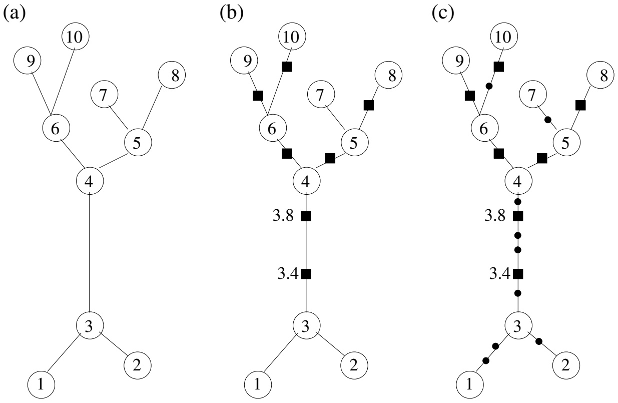

Figure 1: *Schematic examples of contour trees (the label of a node denotes its scalar value): (a) ordinary contour tree; (b) augmented contour tree; (c) our fully augmented contour tree T. In (b), the square nodes are inserted genus-change-only saddle points. Each* segment *defines a* top-eq region*, where an endpoint of a segment can either be a square node or a non-filled circle node. The edge* $(3,4)$ *is divided into three segments* $(3,3.4),(3.4,3.8)$ *and* $(3.8,4)$*. In (c), the additional filled circle nodes are non-critical vertices.*

1. Criticality classification Classify each vertex into a critical or a non-critical point.

2. Contour tree computation Use the algorithm [3] to compute the contour tree. We simplify the last stage of the algorithm so that the final tree is computed *implicitly*. Combining with the criticality information, we obtain the desired fully augmented contour tree T.

3. Volume Segmentation Use T to identify top-eq regions, and label vertices and cells to their corresponding top-eq regions.

We describe these steps in the following sections.

3.1.1. Step 1: Criticality Classification

For each vertex v, we want to classify it into either a critical or a non-critical point. This can be done by locally checking its neighboring vertices [4, 10], summarized as follows.

To classify an *internal* vertex v (i.e., v does not lie on the boundary of the dataset), we take all the tetrahedral cells sharing v as one of their cell vertices. For each such cell, there is one triangle without using v as a vertex (i.e., the triangle facing v). We take all such triangles, whose vertices are exactly the neighbors of v and whose edges connect these neighbors together. These triangles then form a graph G with nodes and edges being the vertices and the edges of the triangles. For each node p in G, we classify p as "$+$" if its scalar value is larger than the scalar value of v, and "$-$" otherwise. Recall that all the scalar values at vertices are distinct (since F is a Morse function). Now in the graph G, we remove edges connecting two nodes of opposite signs (a "$+$" and a "$-$") and obtain a new graph G'. We then compute the connected components in G' via depth-first search or breadth-first search, and count how many connected components there are. Note that the nodes in the same connected component all have the same sign. If there are exactly two components (necessarily one "$+$" and one "$-$") then v is a non-critical point. Otherwise v is a critical point, with the following cases: one component—a minimum (for a "$+$" component) or a maximum (for a "$-$" component), and more than two components—a saddle point.

To classify a *boundary* vertex v, we use the following method of Chiang et. al. [4]: we first perform the same process above; if the number of components is not two, then v is critical as before. When the number of components is two, we make an additional *boundary min/max test*: if the scalar value of v is smaller (resp. larger) than the scalar values of *all* its neighbors *lying on the boundary*, then v is called a *boundary minimum* (resp. *boundary maximum*) and v is a saddle point; otherwise v is non-critical. We remark that Gerstner and Pajarola [10] propose the method of first making v an internal vertex by "mirroring" the neighbors of v across the boundary to obtain the missing vertices, and then applying the process of classifying internal vertices. Our method gives the same result as theirs (as shown in [4]) but is more explicit, which also makes it very easy to deal with the boundary vertices in the criticality-related checkings of types 4 and 5 in the simplification phase; see Section 3.2.

The above process takes time proportional to the number of neighbors of v, which is its vertex degree in the original mesh. The total amount of work is thus bounded by the total number of edges in the mesh, which is linear in the total number of cells. In practice, the number of cells is about $4.5n$ (where n is the number of vertices), and hence this process takes $O(n)$ time.

3.1.2. Step 2: Contour Tree Computation

In this step, we apply the algorithm of Carr *et. al.* [3] to compute the *fully augmented contour tree* T, i.e., the contour tree with all vertices of the mesh present in the tree. The criticality information obtained in Step 1 (Sec. 3.1.1) enables us to distinguish between genus-change-only saddle points and non-critical points, so that the tree T encodes all isosurface topology changes. We also simplify the last stage of the algorithm [3] so that T is built *implicitly*. The algorithm [3] has the following stages.

1. Sort all n vertices of the mesh by the scalar values.
2. Perform a sweep of the n vertices from the smallest scalar value to the largest, and build the *join tree* (defined below).
3. Perform another sweep of the n vertices, now from the largest scalar value to the smallest, and build the *split tree* (defined below).
4. Merge the join tree and split tree together to obtain the fully augmented contour tree.

The formal definitions of *join tree* and *split tree* are given in [3], but the concept is best understood by a sweeping process (see Fig. 2). Given a contour tree, if we sweep the contour tree nodes from bottom to top, and ignore the splitting events, namely, components can only merge (i.e.,*join*) but never split, then we get the *join tree*. Observe that the leaves correspond to the vertices of local minimum and to the creation of isosurface components, and that eventually all components merge into one component, for which the highest node, namely the root, corresponds to a vertex of local maximum (see Fig. 2(b)). Similarly, if we sweep the contour tree nodes from top to bottom and only allow components to merge, then we obtain the *split tree* (see Fig. 2(c)). Note that if we fix the sweeping direction to be always from bottom to top, then the split tree always splits and never merge.

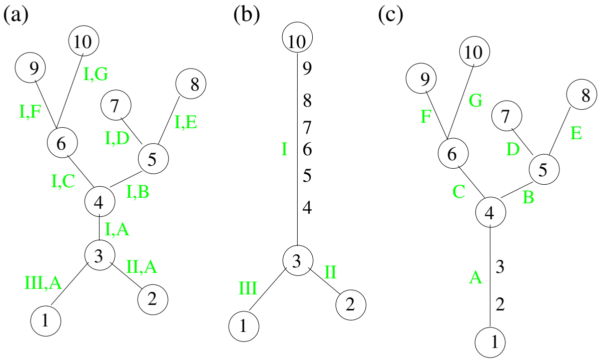

Figure 2: *A contour tree and its corresponding join tree and split tree: (a) contour tree; (b) join tree; (c) split tree. To implicitly merge the join tree and the split tree into the original contour tree, we label the edges of the join tree by I, II, III, and the edges of the split tree by A,...,G. It can be seen that the combined labels, such as "I,A", give a unique ID to each edge of the contour tree and hence define the edges of the contour tree.*

Stage 2 and Stage 3 are completely symmetric, and are essentially the same process performed in the opposite sweeping directions. Therefore it suffices to describe Stage 2, the process of computing the join tree. During the sweep process, the union-find data structure [7] is used to maintain the connectivity information for the isosurface connected components. Initially, each vertex is added to its own singleton set. As we sweep from the vertex of the smallest scalar value to the vertex of the largest scalar value (viewed as changing the isovalue of the isosurface continuously), the vertices corresponding to the same connected component of the isosurface are unioned together into the same set, which corresponds to an edge in the join tree.

For the current vertex v during the sweep, we look at its neighboring vertices (neighboring in the mesh) with scalar value less than v. If there is no such neighbor, then v is a minimum critical point corresponding to the appearing of a new isosurface connected component—we create a new set containing v and also a new leaf of the join tree. If v has some neighbors with smaller scalar values, then each such neighbor has been processed (since the sweep is in the order of increasing scalar values) and put to the correct set. We perform the find-set operation on each of such neighbors. If these neighbors are all in the same set, i.e., in the same connected component, then we just add v to this set, effectively adding v to this connected component. If some of these neighbors belong to different sets, then it means that these different connected components, each from a distinct set, now merge together at v to become a single connected component—v is a saddle point for components merge. We perform union operations on these sets to union them together, as well as putting v to this newly unioned set, and grow the join tree accordingly. Assuming the number of cells is $O(n)$, this sweeping process takes $O(n\alpha(n))$ time, where $\alpha()$ is the inverse of Ackermann's function and $\alpha(n)$ is no more than 4 for all practical values of n [7].

It is shown in [3] that Stage 4 can be performed in time linear in the total size of the join and split trees. We simplify this stage by the following marking scheme. First, in the join tree, we label each edge with some labeling system, say I, II, III.... Then, in the split tree, we label each edge with a different labeling system, say A, B, C.... The concatenated labels, say "I,A", give a unique ID to each edge of the contour tree and hence define the edges of the contour tree (see Fig. 2). We remark that by "labeling a tree edge e", we mean labeling the vertices (including the saddle points and the non-critical points) that belong to the edge e. Clearly, this marking scheme takes $O(n)$ time, and hence the entire contour tree construction takes time $O(n\log n)$.

3.1.3. Step 3: Volume Segmentation

Intuitively, we might want to partition the volume into sub-volumes, each of the same isosurface topology, then simplify each sub-volume independently, and finally stitch the sub-volumes back. However, it is not easy to define the sub-volume boundaries, and stitching back the simplified sub-volumes may be a difficult task.

Our solution is simple. We do not attempt to partition the volume and then stitch back the simplified sub-volumes. Rather, we always keep a *single* volume, with the volume segmentation done *implicitly* by a marking scheme. This marking scheme is based on labeling the vertices, using the fully augmented contour tree T computed in previous steps.

After computing T, we use the genus-change-only saddle points to further divide each edge of T into segments, which correspond to top-eq regions. Recall that each segment contains only non-critical points in the interior (and hence there is no isosurface topology change within a segment), and two critical points at the two segment endpoints. We label all non-critical points of each segment by the unique segment ID. At the end, each non-critical vertex is assigned to the unique top-eq region it belongs to, and each critical vertex is marked as critical.

We assign each cell c to the top-eq region(s) that c belongs to based on the vertex labels. If all vertices of c are *non-critical* and are in the same top-eq region, then c is assigned to that top-eq region; we call such c a *pure cell*. For the remaining cases, c is called an *impure cell*. If all vertices of c are non-critical but belonging to different top-eq regions, then c lies across these top-eq regions and we consider c as belonging to *all* such regions. Finally, if some vertex of c is a critical point, we assign c to *each* top-eq region that contains a non-critical vertex of c (if any). In this way, for each top-eq region r we have the cells assigned to r. Note that a cell may be assigned to more than one top-eq region.

3.2. The Simplification Phase

Our simplification approach is based on the widely used *edge collapse* operation: an edge $e = (v_1, v_2)$ is collapsed to a new point v which may lie anywhere on e, with the scalar value of v obtained by linear interpolation between v_1 and v_2 along e. (In the most general setting, v is not restricted to lying on e.) We conveniently choose v to be on either v_1 or v_2 in our implementation. In addition, all edges incident on v_1 or on v_2 change an endpoint from v_1 (resp. v_2) to v. The edge collapse operation is known to produce very high simplification quality [6]. Moreover, by recording the edge collapse sequence and using the inverse *vertex split* operation, one can build a *multi-resolution hierarchy* which allows one to go back and forth among different levels of details.

For each top-eq region r, we simplify r independently by collapsing edges of r (obtained from the cells assigned to r) from the smallest error to the largest (with respect to a given error metrics), until the user-specified error tolerence ε is met. We keep a *single* mesh of the entire volume throughout the process, and update the mesh accordingly for each edge collapse. To ensure that all isosurface topologies are preserved, we identify a set of conditions to test so that we can determine whether an edge is allowed to be collapsed. We remark that if each top-eq region r only contains *pure cells* to be simplified, then the data-reduction rate is fairly low, as typically there are only about 5–23% pure cells (see Section 4). By putting also *impure cells* to r, we can aggresively achieve a much higher data-reduction rate, but the conditions for edge collapsibility have to be developed more carefully.

We summarize the types of checkings on each edge e for deciding whether e is allowed to be collapsed, as follows.

1. Critical edges If e has one or two critical endpoints, then we never collapse e (e is never put to the priority queue Q for collapsible edges). (For the remaining cases, both endpoints of e are *non-critical*.)

2. Cross-region edges If two endpoints of e are in *different* top-eq regions, then we never collapse e (e is never put to the priority queue Q). Notice that we need the segmentation phase to be able to identify this type of edges. (For the remaining cases, both endpoints of e are in the *same* top-eq region.)

3. Boundary-vertex edges If both endpoints of e are boundary vertices then we never collapse e (e is never put to the priority queue Q). If exactly one endpoint of e is a boundary vertex, then we require that e be collapsed only by moving its internal endpoint to its boundary endpoint (in case it passes all the remaining checkings below). Finally, if both endpoints of e are internal vertices, then we do not put any restriction on e from this type of checking. (For the remaining cases, either both endpoints of e are internal vertices, or exactly one endpoint of e is a boundary vertex and we want to collapse e by moving its internal endpoint to its boundary endpoint.)

The purpose of this checking is to ensure that the geometry of the volume boundary is preserved, which, though not necessary for preserving the isosurface topologies, is often a desirable property for volume simplification. Moreover, this checking makes it easy to deal with the boundary vertices in the criticality-related checkings in types 4 and 5 below.

4. Critical-neighbor edges If $e = (v_1, v_2)$ has a critical neighbor c (say c is a neighbor of v_1), then in collapsing e, it is possible to change the criticality of c (i.e., making c *non-critical*). If we collapse e by moving v_2 to v_1, denoted by $v_2 \rightarrow v_1$, then the scalar values of the neighbors of c do not change, and hence c does not change its criticality (recall from the criticality classification in Section 3.1.1). In this case, we pass the checking for this type and go to the next type.

If on the other hand we want to collapse e by $v_1 \rightarrow v_2$, we first test whether v_1 and v_2 have scalar values *both* larger than or *both* smaller than the scalar value $F(c)$ of c so that the number of "+" and "−" neighbor-components of c stays the same after the collapse. If the test fails, we disallow the collapse right away. When the test succeeds, if c is *not* a boundary min/max, then its number of components remains not equal to two (hence c remains critical; see Section 3.1.1) after the collapse and we pass this type of checking. When c is a boundary mininum (resp. maximum), we need to check the possible changes in the boundary neighbors of c. Since we want to collapse e by $v_1 \rightarrow v_2$, v_1 is an internal vertex (by the type 3 checking above) and we only need to check v_2: if v_2 is an internal vertex, then the boundary neighbors of c stay the same and we pass this checking; when v_2 is a boundary vertex, it becomes a new boundary neighbor of c after the collapse, and we pass the checking if and only if $F(c) < F(v_2)$ (resp. $F(c) > F(v_2)$).

5. Criticality checking The criticality checkings so far make sure that we do not *reduce* any isosurface topological features (i.e., do not make a critical point non-critical in types 1 and 4, and do not merge two isosurface connected components or close a hole in type 2). Now we also need to make sure that collapsing edge e (which has two non-critical endpoints) to a point v will not incorrectly introduce a *new critical point*, i.e., the resulting v will not be a new critical point. We have two steps of checking the condition: first we use the *easy checking* step, passing the test and go to the next type if the easy checking succeeds, or otherwise we use the additional *full checking* step. We describe these steps in detail below. We disallow collapsing e if the two-step checking fails, or go to the next test type otherwise.

6. Foldover checking Finally, we want to avoid the *fold-over* problem. That is, for each tetrahedral cell $t = (u_1, u_2, u_3, u_4)$ affected by collapsing the edge e, vertex u_i should stay on the *same side* of the triangle $\triangle u_j u_k u_l$ facing u_i before and after collapsing e, where $i \notin \{j, k, l\}$, for each u_i. In other words, after collapsing e, we should not produce a tetrahedral cell of *negative volume*. We test, for each affected cell t, whether its *signed volume* has the same sign (positive or negative) before and after collapsing e. If this is true for each t, then we allow collapsing e, otherwise we disallow the collapse.

By the above types of tests, we guarantee that the simplification technique preserves the geometry of the volume boundary, avoids the fold-over problem, and preserves all isosurface topologies in the entire simplification process. In the rest of this section, we consider how to avoid unnecessary or expensive tests to make the algorithm very efficient.

Efficient Implementation for the Type 4 Checking For the type 4 checking, we need to see whether any endpoint of e has a critical neighbor. As shown in Section 4, there are only less than 5% of critical vertices, and thus for most cases e does not have a critical neighbor. We want to avoid checking all neighbors for each endpoint of e, for each edge e being considered. We use the following simple solution. For each vertex v we maintain a list $C(v)$ of critical neighbors of v and the list size $|C(v)|$. This information is updated locally each time we actually collapse an edge. For the current edge e, we do not need to consider the type 4 test if both endpoints have empty lists.

Details of the Type 5 Checking Now we describe the two-step tests for the type 5 checking. The goal is to test, when collapsing an edge $e = (v_1, v_2)$ (with both endpoints being non-critical) to a vertex v, whether v is still *non-critical*. We first show that we can use the *same* procedure to treat both *internal* and *boundary* vertices. If both endpoints of e are internal vertices, then v is also an internal vertex, and we only need to test whether the number of "$+$" and "$-$" components of v is two (recall from the criticality classification in Section 3.1.1). If exactly one endpoint, say v_1, is a boundary vertex, then by type 3 checking we only try to collapse e by $v_2 \rightarrow v_1$ and the resulting v is a boundary vertex. In this case, v is non-critical if and only if (1) its number of components is two *and* (2) it is *not* a boundary min/max (recall from Section 3.1.1). However, since v_1 is a *non-critical boundary* vertex, v_1 is *not* a boundary min/max in the first place, and thus v, having the same scalar value as v_1 and inheriting the boundary neighbors of v_1 as (part of) its boundary neighbors, automatically satisfies the condition (2). Therefore, our treatment of boundary vertices for the type 5 checking is *transparent*: regardless of whether e contains a boundary vertex, we only test whether the number of components of v is two.

To perform the type 5 checking, we use a two-step test. The *full checking* step is to actually perform the collapse, and check whether the resulting vertex v has two components by the classification method of Section 3.1.1. While the method works, it is rather expensive. We develop an *easy checking* step based on the following nice property.

Lemma 1 *Let $e = (u,v)$ be an edge with two non-critical endpoints, and we want to collapse e to a new vertex p with the scalar value of p assigned to be any value in the interval $I = [F(u), F(v)]$ ($F(u) < F(v)$). If the neighbors of u and of v all have scalar values* outside *I, then the number of "+" and "-" components of p is two (and thus the type 5 checking succeeds).*

Proof: Since u, v are non-critical, each of them has *two* connected components of "$+$", "$-$" neighbors; recall from Section 3.1.1. Also, since all neighbors of u and of v have scalar values *outside* I, the "$+$" component of u (resp. of v) has scalar values larger than both $F(v)$ and $F(u)$, and similarly the "$-$" component of v (resp. of u) has scalar values smaller than both $F(u)$ and $F(v)$.

Let A be the set of all the neighbors of u with "$+$", *excluding* v (note that v is a neighbor of u with a "+" mark since $F(v) > F(u)$), B the set of all the neighbors of v with "$+$", C the set of all the neighbors of u with "$-$", and D the set of all the neighbors of v with "$-$", *excluding* u (again u is a neighbor of v with "$-$" since $F(u) < F(v)$); see Figure 3. We claim that after collapsing (u,v) to p, there is only one "$+$" connected component for p.

To prove the claim, consider two cases. If A is an empty set, then the claim is trivially true: p has only one connected component B. Now suppose A is not an empty set. Then A is a non-empty "$+$" component of u. Since v is also a "$+$" of u, v must be connected to A, or otherwise u would have more than two connected components and hence u would be a critical point, a contradiction. Suppose v is connected to A via a vertex $t \in A$, then t is a neighbor of v with "$+$", we have that t is in B (by the definition of B). This means that t is in the connected component A and is also in the connected component B, i.e., A and B are in the *same* connected component, and thus there is only one "$+$" component for p. This completes the proof of the claim. By the same argument, there is only one connected component of "$-$" for p after collapsing (u,v) to p. Hence, p has two connected components, one "$+$" and one "$-$". ☐

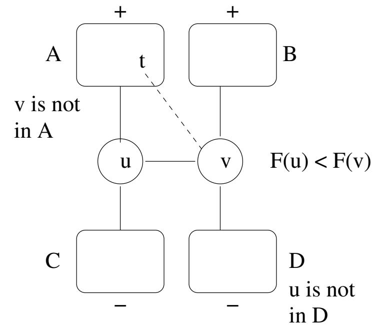

Figure 3: *Proof of Lemma 1.*

Notice that the condition in Lemma 1 is much easier to test. In summary, for the type 5 checking, we first perform the *easy checking* step by applying Lemma 1. If this step succeeds, then the type 5 checking is successful and we can pass it; otherwise, we continue to perform the more expensive *full checking* step.

Efficient Method for the Overall Checkings Now we consider the efficiency issue from a more global viewpoint. Suppose the current "shortest" edge (i.e., edge with the smallest error) e cannot be collapsed because it does not pass the criticality (type 5) checking or the fold-over (type 6) checking. Then the problem continues to exist (i.e., e cannot pass the checking) as long as the neighbors of e do not change, in which case it would be very inefficient to keep checking e, especially for the expensive full checking step for type 5 and the fold-over checking of type 6.

We employ the following ideas. For such edge e, we mark e, and do *not* put e back to the priority queue Q (and continue to examine the next shortest edge); in the process, there will be a number of marked edges that we do not put back to Q. Such marked edges e will be put back to Q only when some new edge neighboring to e is collapsed that may affect the checking results of e. Therefore, we check the failed edge again only when there is a new hope for the checking to succeed. In this way, we avoid many unnecessary tests and greatly speed up the process.

We describe the operations for the type 5 (criticality) checking; similar considerations apply to the fold-over checking. When we collapse an edge (a,b) (which is of course not a marked edge e) to a new vertex v, all the neighbors of v could be affected. There are two types of edges that

Data	# cells	# vertices	# pure cells (%)
spx	12936	20108	97 (0.75%)
blunt	187395	40960	19330 (10.32%)
comb	215040	47025	10233 (4.76%)
post	513375	109744	348460 (67.88%)
tpost	615195	131072	105381 (17.13%)
delta	1005675	211680	235976 (23.46%)

Data	crit. pts (%)	top-eq-p	max cell	seg (s)
spx	17685 (87.95%)	19	44	0.81
blunt	1891 (4.62%)	30	14287	14.27
comb	641 (1.36%)	57	3818	7.58
post	1116 (1.02%)	44	267946	27.05
tpost	1610 (1.23%)	116	24372	44.16
delta	1624 (0.77%)	128	42509	68.96

Table 1: *Experimental results for the segmentation phase. For each dataset, we list the numbers of cells, of vertices, of pure cells (and the percentage), of critical points (and the percentage), of top-eq regions* with at least one pure cell, *and the maximum number of pure cells among the top-eq regions, respectively. Finally, we show the overall running time (in seconds) for the segmentation phase, excluding the time for reading the input file.*

may be affected with respect to the criticality checking: (1) edges using v as an endpoint, and (2) edges using a neighbor of v as an endpoint (more efficiently, for $b \longrightarrow a$, say, we can look at only the edges using a neighbor of b as an endpoint). When we collapse (a,b), we look at all the edges of types (1) and (2); for those that are marked, we unmark them and put them back to Q as new candidates for collapsing. In this way, we avoid checking the same situation repeatedly, while still ensuring that any new possible candidates are not missed.

4. Results

We have implemented our simplification technique in C++/C, and ran our experiments on a Sun Blade 1000 workstation with 750MHz UltraSPARC III CPU, 4GB of main memory, and an Expert3D graphics. The datasets we tested are listed in Table 1; they are either a tetrahedral dataset (spx) or curvilinear datasets tetrahedralized into tetrahedral meshes (the others), where tpost was obtained by taking one time step from a time-varying dataset.

Data	ε	cell %	easy	full	simp (s)
comb	g 0.4	70.80	14257	1	11.25
comb	g 0.65	63.73	17466	3	16.71
comb	g ∞	52.15	22019	8	73.81
comb	h ∞	51.82	21930	10	88.30
comb	s ∞	51.88	21828	9	91.63
spx	s ∞	82.01	504	0	0.32
blunt	g ∞	64.60	20828	23	20.05
tpost	g ∞	40.89	86449	156	341.74
post	g ∞	11.13	149026	753	2686.06
delta	g ∞	35.42	222837	370	1056.28

Table 2: *Experimental results for the simplification phase. We show the error tolerance ε (and the error metrics used: 'g' for edge length, 's' for scalar-value difference, 'h' for combined and weighted half and half), where ∞ means we simplify to the maximum extent possible, and the final number of* remaining *cells as percentage of the original number of cells. We also show the number of the* easy checking *steps of Lemma 1 that are successful ('easy'), and the number of the* full checking *steps (for criticality checking, type 5 test) that are successful ('full'). Finally, we show the overall running time (in seconds) for the simplification phase.*

Segmentation Phase We show in Table 1 the statistics of running our segmentation phase. It can be seen that typically the number of pure cells is fairly small, only about 5–23% of the original number of cells (except for the two extreme cases, spx (0.75%) and post (67.88%)). If we only simplify the pure cells, then the data reduction rate would be no more than 5–23% in typical cases, which would be very undesirable. By aggressively simplifying the large number of *impure* cells as well, we can obtain a much higher data-reduction rate (see Table 2 below). We also see that the number of critical points is quite small too, typically less than 5% of the original number of vertices. This is why in our type 4 checkings (critical-neighbor edges) in the simplification phase (see Section 3.2) we want to have an efficient method to quickly tell whether the endpoints of the current edge have critical neighbors. Also, recall that top-eq regions correspond to the segments in the *augmented contour tree* whose nodes are all critical points, and hence the number of top-eq regions is always the number of critical points minus one; we omit showing this number but rather list the number of top-eq regions *with at least one pure cell*. Finally, we observe that the running time for the segmentation phase is quite small, ranging from 0.81 to 68.96 seconds.

Simplification Phase We used three different error metrics: edge length (geometry, denoted as 'g'), scalar-value difference ('s'), and combined, with each one weighted a half ('h'). For the purpose of comparisons, we have also implemented a volume simplification method, `no_top`, which performs the same steps as our technique in the simplification phase but skips all topology-related steps (i.e., it only preserves the geometry of volume boundary and avoids the fold-over problem). We show the isosurfaces generated from the volumes simplified by `no_top` and by our technique in Figures 4 and 5, with various error metrics and error tolerances. Observe that for `no_top`, isosurface components are incorrectly merged or split and holes are incorrectly closed or created, since `no_top` may collapse edges connecting different top-eq regions as well as edges producing new critical vertices (reducing and increasing the topological features). On the other hand, our technique always preserves the isosurface topologies. By setting the error tolerance ε to ∞, we allow the simplification process to continue to the maximum extent possible, but still the isosurface topologies are all correctly preserved, no matter which error metrics we choose. This is a feature that other existing tetrahedral-mesh simplification methods can not guarantee, and shows the correctness and strength of our technique.

In Table 2, we show the statistics of running our simplification phase. It can be seen that our data-reduction rates are much higher than the percentages of pure cells (see Table 1). For example, in the comb dataset, the pure cells account for 4.76%, but we are able to achieve reduction rates about 48% when simplifying to the maximum extent, for all metrics used. We remark that the spx dataset has complicated topologies and hence we can not simplify much. This can be seen by the extremely large percentage of the critical points (87.95%) contained in the data (see Table 1). Our algorithm runs competitively fast; although it is slower than the fastest reported method of Chopra and Meyer [5], it is certainly several orders of magnitude faster than the techniques of Trotts *et. el.* [17] (their error metrics is different though), which took 1243 minutes (about 20.7 hours) to simplify the blunt dataset to 66.1% cells remaining.

It is very interesting to compare the numbers in 'easy' and in 'full' from Table 2: the sum of these two numbers means the total number of the criticality checkings (type 5 checkings) that are successful, and 'easy' accounts for more than 99.5% of them. This shows that in practice, we could actually use the *easy checking* step of Lemma 1 to replace the expensive *full checking* step: if an edge e does not pass the *easy checking*, then we just disallow collapsing e without doing the *full checking*. This does not affect the correctness of our technique (as proved in Lemma 1), but would greatly speed up our algorithm, with a slight decrease in the data-reduction rate. To verify these ideas, we made such easy-checking-only approach an option in our algorithm, and repeated the same runs as those in Table 2 for this option; the results are shown in Table 3. We see that the data-reduction rates are indeed almost the same (surprisingly, some reduction rates are slightly higher, which could be explained by the fact that the edge-collapse sequence is slightly changed). More importantly, the algorithm runs about 2–3 times as fast (about 2.7 times as fast for the three longest runs), showing that the *easy checking* step is indeed a nice replacement for the bottleneck *full checking* step.

Data	ε	cell %	easy	simp (s)
comb	g 0.4	70.98	14155	6.28
comb	g 0.65	63.76	17444	8.49
comb	g ∞	52.04	21996	25.56
comb	h ∞	52.07	21871	26.28
comb	s ∞	52.17	22007	30.63
spx	s ∞	87.95	484	0.22
blunt	g ∞	64.44	19668	17.41
tpost	g ∞	40.82	86681	127.09
post	g ∞	11.33	148898	1013.3
delta	g ∞	35.52	213906	375.29

Table 3: *Experimental results for the simplification phase, in which we only perform the* easy checking *step of Lemma 1 for the criticality checkings (type 5 checkings), without performing any* full checking *step. The meanings of the entries are the same as those in Table 2.*

5. Conclusions

We have presented a novel tetrahedral-mesh simplification technique that is guaranteed to preserve the topologies of all isosurfaces embedded in the data, with a controlled geometric error bound. We have developed the theoretical foundation for our technique, and demonstrated its effectiveness in practice. Moreover, we obtain nice data-reduction rates, with competitively fast running times.

A possible extension of this work is to apply our simplification algorithm to build a multiresolution volume hierarchy stored in a data structure, and support efficient run-time isosurface extraction satisfying the user-specified error tolerance from the right level of details while preserving the correct isosurface topology. We already have some preliminary results along this direction.

Acknowledgements

We thank Boris Aronov and Günter Rote for discussions on critical points. We thank Han-Wei Shen for providing his

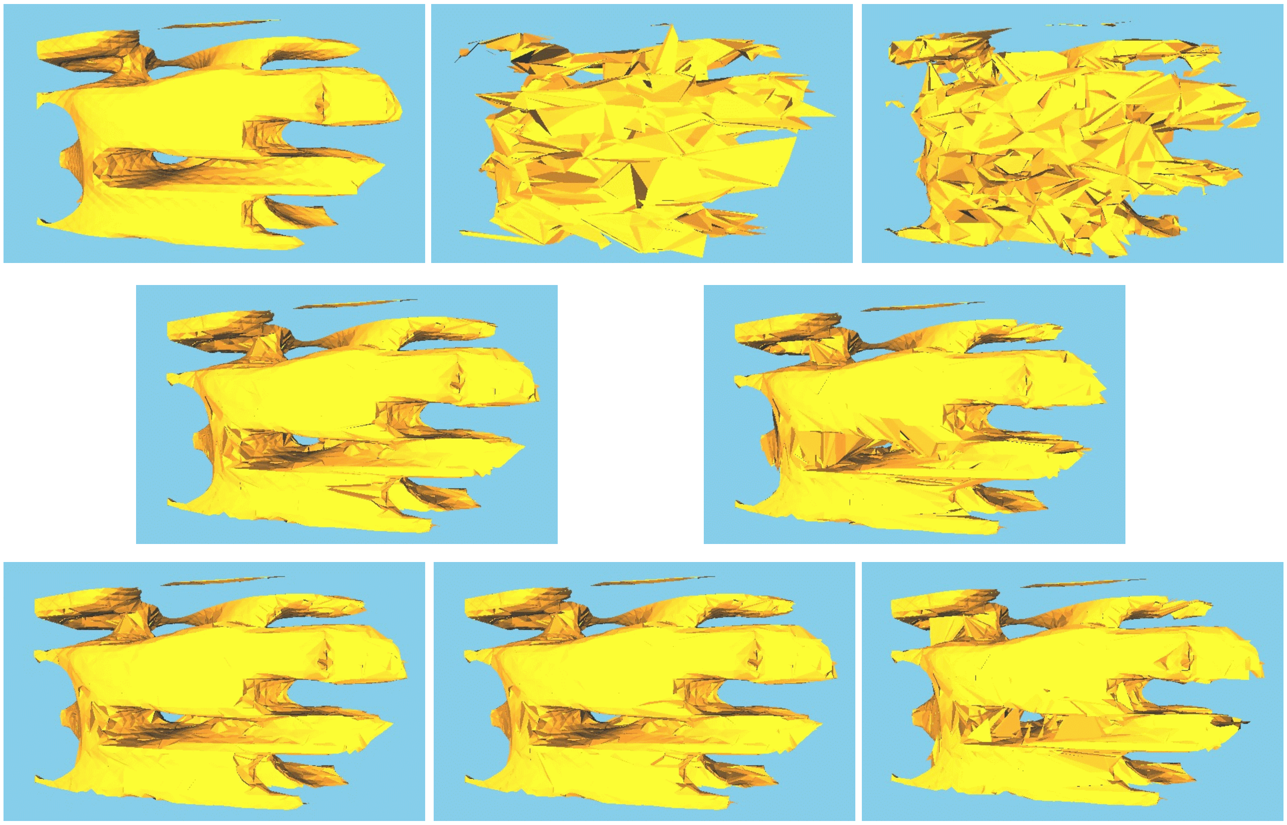

Figure 4: *Isosurfaces from the simplified volumes: the comb dataset with isovalue 0.251. Top row: left: original volume (38512 triangles); middle and right:* without *preserving isosurface topologies. Middle: combined half error metrics,* $\varepsilon = 0.64$ *(7.74% cells remaining, 9944 triangles); right: edge-length error metrics,* $\varepsilon = 0.65$ *(21.36% cells remaining, 18085 triangles). Middle row (*our method*, combined half error metrics): left:* $\varepsilon = 0.64$ *(62.05% cells remaining, 28986 triangles); right:* $\varepsilon = \infty$ *(51.82% cells remaining, 27099 triangles). Bottom row (*our method*, edge-length error metrics): left:* $\varepsilon = 0.4$ *(70.80% cells remaining, 30897 triangles); middle:* $\varepsilon = 0.65$ *(63.73% cells remaining, 29276 triangles); right:* $\varepsilon = \infty$ *(52.15% cells remaining, 27078 triangles).*

Marching Cubes/Tetrahedra code and the tpost dataset. We thank Cláudio Silva, Peter Williams and Prashant Chopra for the spx dataset. The Blunt Fin, the Liquid Oxygen Post, and the Delta Wing datasets are courtesy of NASA. The Combustion Chamber dataset is from Vtk [13].

References

1. C. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *Proc. IEEE Visualization*, pages 167–175, 1997.
2. T. F. Banchoff. Critical points and curvature for embedded polyhedra. *J. Diff. Geom.*, 1:245–256, 1967.
3. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, pages 918–926, 2000.
4. Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and output-sensitive construction of contour trees using monotone paths, 2003. Manuscript. http://cis.poly.edu/chiang/contour.pdf.
5. P. Chopra and J. Meyer. Tetfusion: An algorithm for rapid tetrahedral mesh simplification. In *Proc. IEEE Visualization*, pages 133–140, 2002.
6. P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. In *Proc. IEEE Visualization*, pages 85–92, 2000.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and

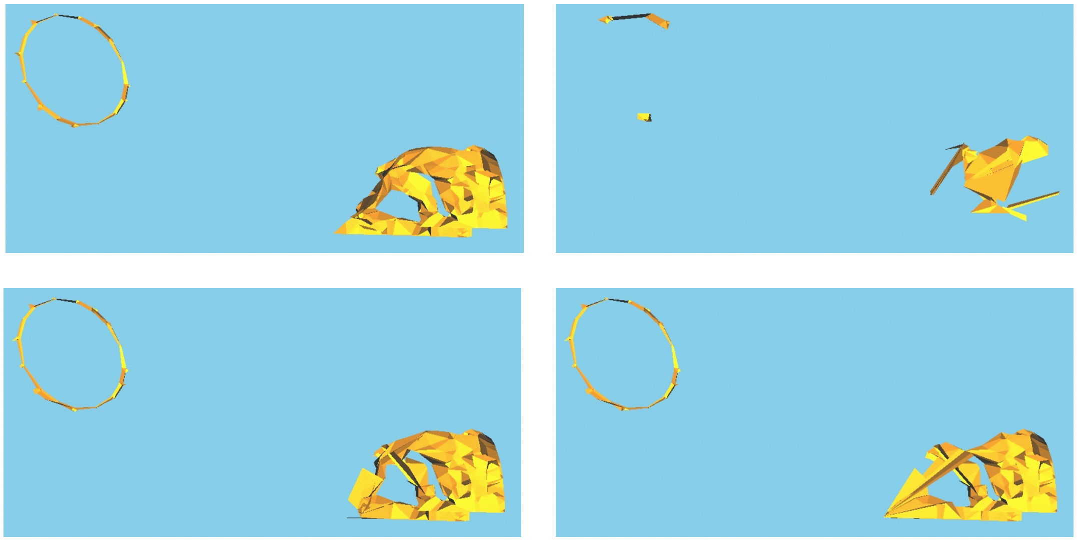

Figure 5: *Isosurfaces from the simplified volumes: the spx dataset with isovalue 1.25. Top row: left: the original volume (850 triangles); right:* without *preserving isosurface topologies, edge-length error metrics,* $\varepsilon = 0.61$ *(23.64% cells remaining, 161 triangles). Bottom row:* our method*. Left: edge-length error metrics,* $\varepsilon = 0.61$ *(85.48% cells remaining, 827 triangles); right: scalar-value difference,* $\varepsilon = \infty$ *(82.01% cells remaining, 799 triangles).*

C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

8. M. de Berg and M. van Kreveld. Trekking in the Alps without freezing or getting tired. *Algorithmica*, 18:306–323, 1997.

9. T. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology preserving edge contraction. *Publications de l'Institut, Mathematique (Beograd)*, 60(80), 1999.

10. T. Gerstner and R. Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *Proc. IEEE Visualization*, pages 259–266, 2000.

11. J. W. Milnor. *Morse Theory*. Princeton University Press, Princeton, NJ, 1963.

12. V. Pascucci and K. Cole-McLaughlin. Efficient computation of the topology of level sets. In *Proc. IEEE Visualization*, pages 187–194, 2002.

13. W. Schroeder, K. Martin, and W. Lorensen. *The Visualization Toolkit*. Prentice-Hall, 1996.

14. Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien. Surface coding based on Morse theory. *IEEE Comput. Graph. Appl.*, 11:66–78, September 1991.

15. O.G. Staadt and M.H. Gross. Progressive tetrahedralizations. In *Proc. IEEE Visualization*, pages 397–402, 1998.

16. S. Tarasov and M. Vyalyi. Construction of contour trees in 3D in $O(n \log n)$ steps. In *Proc. ACM Sympos. Comput. Geom.*, pages 68–75, 1998.

17. I.J. Trotts, B. Hamann, and K.I. Joy. Simplification of tetrahedral meshes with error bounds. *IEEE Trans. Visualization and Computer Graphics*, 5(3):224–237, 1999.

18. I.J. Trotts, B. Hamann, K.I. Joy, and D.F. Wiley. Simplification of tetrahedral meshes. In *Proc. IEEE Visualization*, pages 287–295, 1998.

19. M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. ACM Sympos. Comput. Geom.*, pages 212–220, 1997.

20. G. H. Weber, G. Scheuermann, and B. Hamann. Detecting critical regions in scalar fields. In *Data Visualization 2003 (Proc. VisSym)*, 2003.

21. Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder. Isosurface topology simplification, 2002. Manuscript. http://www.multires.caltech.edu/pubs/topo_filt.pdf.