Simplifying Flexible Isosurfaces Using Local Geometric Measures

Hamish Carr
Department of Computer Science
University of British Columbia

Jack Snoeyink
Department of Computer Science
University of North Carolina at Chapel Hill

Michiel van de Panne
Department of Computer Science
University of British Columbia

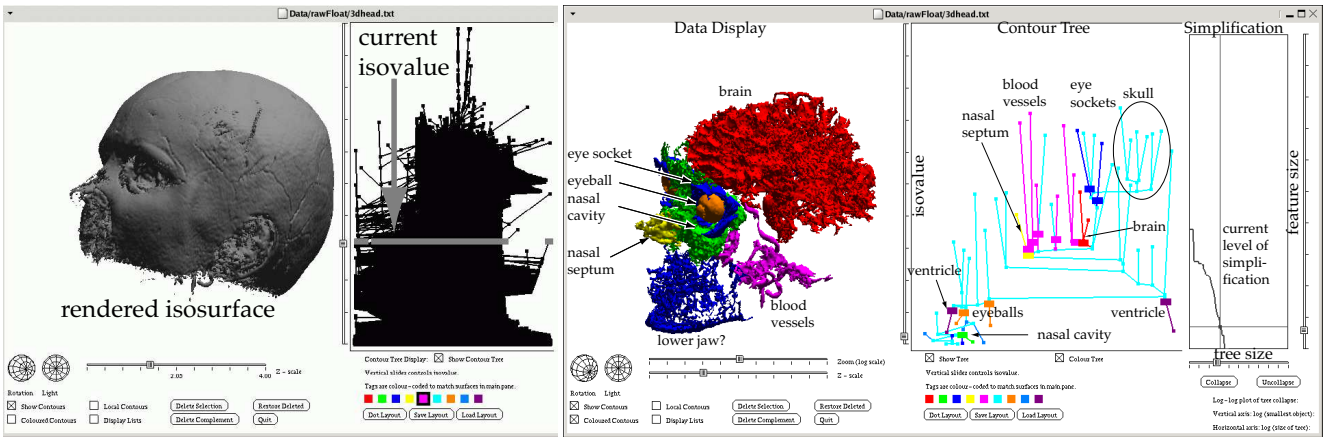


Figure 1: Left, an isosurface of the UNC Head ($109 \times 256 \times 256$ MRI) shows mostly the skull, and the contour tree is unmanageably large (1,573,373 edges). Right, a flexible isosurface chosen from a simplified contour tree. The rightmost pane controls the amount of simplification of the contour tree shown immediately to its left (92 edges as shown). (Annotation and colour chosen to emphasize the structure of the data.) The interface supports interactive exploration of structures that are hidden in the conventional view.

Abstract

The contour tree, an abstraction of a scalar field that encodes the nesting relationships of isosurfaces, can be used to accelerate isosurface extraction, to identify important isovalues for volume-rendering transfer functions, and to guide exploratory visualization through a flexible isosurface interface. Many real-world data sets produce unmanageably large contour trees which require meaningful simplification. We define local geometric measures for individual contours, such as surface area and contained volume, and provide an algorithm to compute these measures in a contour tree. We then use these geometric measures to simplify the contour trees, suppressing minor topological features of the data. We combine this with a flexible isosurface interface to allow users to explore individual contours of a dataset interactively.

Keywords: Isosurfaces, contour trees, topological simplification

1 Introduction

Isosurfaces, slicing, and volume rendering are the three main techniques for visualizing three-dimensional scalar fields on a two-dimensional display. A recent survey [Brodlie and Wood 2001] describes the maturation of these techniques since the mid 1980s. For example, improved understanding of isosurfaces has produced robust definitions of watertight surfaces and efficient extraction methods. We believe that the same improved understanding and structuring leads to new interfaces that give the user better methods to select isosurfaces of interest and that provide a rich framework for data-guided exploration of scalar fields.

Although key ideas in this paper apply to both isosurfaces and volume rendering, the immediate application is to isosurface rendering. An isosurface shows the surface for a fixed value (the *isovalue*) of the scalar field and is the 3D analogue of equal-height contour lines on a topographic map. The *contour tree* represents the nesting relationships of connected components of isosurfaces, which we call *contours*, and is thus a topological abstraction of a scalar field. Since genus changes to surfaces do not affect the nesting relationship, they are not represented in the contour tree. Our contribution is to combine the flexible isosurface interface [Carr and Snoeyink 2003] with online contour tree simplification guided by geometric properties of contours to produce a tool for interactive exploration of large noisy experimentally-sampled data sets. An additional contribution is to draw attention to other potential applications of simplified contour trees, such as detail-preserving denoising, automated segmentation, and atlasing.

Figure 1 shows a comparison between a conventional isosurface and a flexible isosurface extracted from the same data set after contour tree simplification. On the left, the outermost surface (the skull) occludes other surfaces, making it difficult to study structures inside the head. Moreover, the contour tree for this data set has over 1 million edges, making it impractical as a visual representation. On the right is a flexible isosurface constructed using a simplified contour tree, laid out and coloured to emphasize the structure of the data set. Of particular interest is that there are no “magic numbers” embedded in the code. Instead, the surfaces shown were chosen

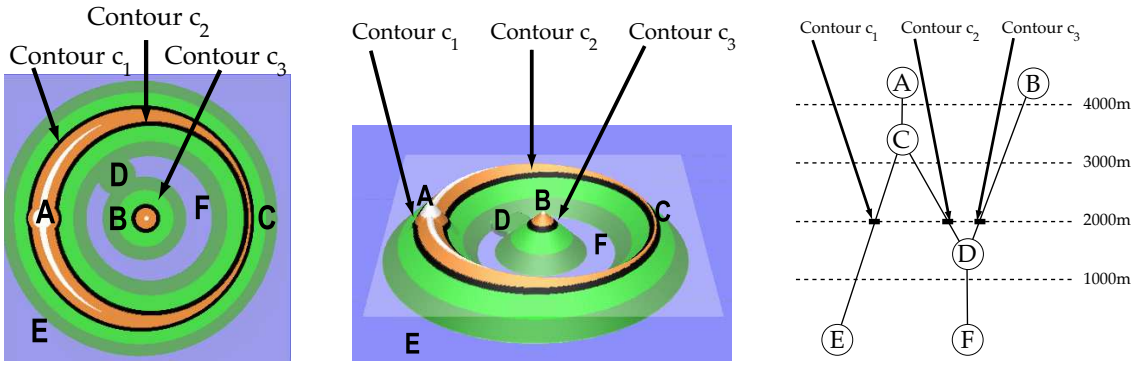


Figure 2: The topographic map (2-d scalar field), surface rendering, and contour tree for a volcanic crater lake with a central island. A: a maximum on the crater edge; B: maximum of island in the lake; F: lake surface; C and D: saddle points.

directly from the simplified contour tree during exploration of this data set, with the level of simplification being adjusted as needed.

The remainder of this paper is as follows. Section 2 reviews work on contour trees in visualization. Section 3 shows how to simplify the contour tree, and the effects on the data. Section 4 shows how to compute *local geometric measures* efficiently to guide simplification. Section 5 gives implementation details, and Section 6 reports results. Finally, Section 7 gives possible future extensions.

2 Related Work

Most of the relevant work deals with a topological structure called the *contour tree* that is becoming increasingly important in visualization. Section 2.1 reviews the contour tree and algorithms to compute it. Section 2.2 then reviews visualization tools that use the contour tree, while Section 2.3 reviews work on topological simplification and on efficient computation of geometric properties.

2.1 The Contour Tree

For a scalar field $f:\mathbb{R}^3 \rightarrow \mathbb{R}$, the *level set* of an *isovalue* h is the set $L(h) = \{(x,y,z) \mid f(x,y,z) = h\}$. A *contour* is a connected component of a level set. As h increases, contours appear at local minima, join or split at saddles, and disappear at local maxima of f. Shrinking each contour to a single point gives the *contour tree*, which tracks this evolution. It is a tree because the domain $\mathbb{R}^3$ is simply-connected; in more general domains we obtain the *Reeb graph* [Reeb 1946], which is used in Morse theory [Matsumoto 2002; Milnor 1963] to study the topology of manifolds.

Figure 2 shows a 2-dimensional scalar field describing a volcanic crater lake with a central island. The contour tree of this field is an abstract, but meaningful, depiction of the structure of all local maxima, minima, and saddle points, and gives clues to interesting contours. Individual contours are represented uniquely as points on the contour tree. For example, the isolines c_1, c_2, and c_3 are all at $2000m$, but each has a unique location on the contour tree.

The contour tree has been used for fast isosurface extraction [van Kreveld et al. 1997; Carr and Snoeyink 2003], to guide mesh simplification [Chiang and Lu 2003], to find important isovalues for transfer function construction [Takahashi et al. 2004b], to compute topological parameters of isosurfaces [Kettner et al. 2001], as an abstract representation of scalar fields [Bajaj et al. 1997], and to manipulate individual contours [Carr and Snoeyink 2003].

Algorithms to compute the contour tree efficiently in three or more dimensions have been given for simplicial meshes [van Kreveld et al. 1997; Tarasov and Vyalyi 1998; Carr et al. 2003; Chiang et al. 2002; Takahashi et al. 2004b] and for trilinear meshes [Pascucci and Cole-McLaughlin 2002]. Much of this work focusses on "clean" data from analytic functions or numerical simulation – see for example [Bajaj et al. 1997; Takahashi et al. 2004b]. All of the topology in this data is assumed to be important and significant effort is expended on representing it accurately using trilinear interpolants [Pascucci and Cole-McLaughlin 2002] and topology-preserving simplifications [Chiang and Lu 2003].

In contrast, we are interested in noisy experimentally-acquired data such as medical datasets. We expect to discard small-scale topological features so that we can focus on large-scale features. We have therefore chosen to work with the well-known Marching Cubes cases [Lorenson and Cline 1987; Montani et al. 1994], and with approximate geometric properties. This paper does not turn on these choices, however, and can also be applied to trilinear interpolants and exact geometric properties.

2.2 Flexible Isosurfaces

The *contour spectrum* [Bajaj et al. 1997] uses the contour tree to represent the topology of a field, alongside global measures of level sets such as surface area and enclosed volume. In contrast, the *flexible isosurface* interface [Carr and Snoeyink 2003] uses the contour tree actively instead of passively. The user selects an individual contour from the contour tree or from the isosurface display, then manipulates it. Operations include contour removal and contour evolution as the isovalue is changed, using the contour tree to track which contours to display. This interface depends on attaching isosurface seeds called *path seeds* to each edge of the contour tree so that individual contours can be extracted on demand.

A major disadvantage of both these interfaces is that contour trees with more than a few tens of edges make poor visual abstractions. A principal contribution of this paper to simplify the contour tree while preserving the exploratory capabilities of the flexible isosurface. This requires that each point in a simplified contour tree represents an extractable contour. Moreover, extracted contours must evolve as smoothly as possible when the isovalue is adjusted.

We satisfy these constraint with simplifications that have predictable effects on the scalar field and geometric measures that identify unimportant contour tree edges for simplification

2.3 Simplification and Geometric Measures

The distinction between this paper and other work that simplifies contour trees or Reeb graphs is our emphasis on using tree structure for local exploration. [Takahashi et al. 2004a] simplify the contour tree by replacing three edges at a saddle point with a single new edge, based on the height of the edge. [Takahashi et al. 2004b] use the approximate volume of the region represented by the subtree that is discarded. Saddles are processed until only a few remain, then a transfer function is constructed that emphasizes the isovalues of those saddles. Our simplification algorithm extends this work to preserve local information such as isosurface seeds and to compute arbitrary geometric measures of importance. We also describe the effects of simplification on the scalar field.

Since removing a leaf of the contour tree cancels out a local extremum with a saddle, this form of simplification can be shown to be equivalent to topological persistence [Edelsbrunner et al. 2003; Edelsbrunner et al. 2002; Bremer et al. 2003] if the geometric measure used is height. For other measures, such as volume or hypervolume, the method described in this paper is necessary to define these properties, but thereafter, the process can optionally be described in terms of persistence.

Moreover, work on persistence has focussed on the Morse complex, which is difficult to compute and segments data according to the gradient of the field. When the boundary of an object such as an organ is better described by a contour than by drainage, contour trees are more directly applicable than Morse complexes, and the additional overhead of working with the Morse complex is unnecessary.

[Hilaga et al. 2001] have shown how to simplify the Reeb graph by progressive quantization of the isovalue to obtain a multi-resolution Reeb graph. This suffers from several drawbacks, in particular that it is strictly tied to a function value which is treated as height (or persistence). Extension to geometric measures of importance such as volume or hypervolume is therefore problematic. Moreover, the quantization used imposes serious restrictions on isosurface generation and the level of simplification, as well as generating artifacts related to the quantization. In particular, we note that this quantization process limits potential simplification to at most as many levels as there are bits in each input sample. Finally, this method is relatively slow: 15s is claimed for a 2-manifold input mesh with 10,000 vertices: extensions to 10,000,000+ sample volumetric data have not yet been published.

Work also exists on computing geometric measures efficiently in large data sets. [Bentley 1979] defined problems to be *decomposable* if their solution could be assembled from the solutions of an arbitrary decomposition into subproblems. Decomposability has been used for a variety of problems, including computation of geometric properties of level sets [Bajaj et al. 1997] and extraction of isosurfaces [Lorenson and Cline 1987]. We use decomposability in Section 4 to compute local geometric measures.

3 Contour Tree Simplification

Given a contour tree and a scalar field, we apply graph simplification to the contour tree. This simplification can then be carried back to simplify the input data. Alternately, we can use the simplified contour tree to extract the reduced set of isosurfaces that would result if we had simplified the data. In this section, we describe the contour tree structure, the simplification operators, and the algorithms for simplification and isosurface extraction.

3.1 Contour Tree Structure

A contour tree is the result of contracting every contour to a point. We use a simple tree structure in which every vertex is assigned a y-coordinate, and every edge is associated with the set of contours between its vertices. We store *path seeds* for generating individual contours, as in [Carr and Snoeyink 2003]. That is, we store a pointer to a monotone path that intersects all contours along the edge, which then serves as a seed to generate any given contour. In this section, we assume that each edge has a simplification value (weight) that indicates the edge's priority. Low priority edges are good candidates for simplification.

3.2 Basic Simplification Operations

We simplify the contour tree with two operations: *leaf pruning* and *vertex reduction*. Leaf pruning removes a leaf of the tree, reducing the complexity of the tree, as shown in Figure 3, where vertex 80 is pruned from the tree on the left to produce the tree in the middle. Vertex reduction chooses a vertex with one neighbor above and one below, and deletes the vertex without changing the essential structure of the contour tree. This is also illustrated in Figure 3, where vertex 50 has been removed from the tree in the middle to produce the tree on the right. Since vertex reductions do not change the essential structure of the contour tree, we prefer them to leaf prunes. Also, pruning the only up- or down- edge at a saddle is prohibited to preserve the edge for a later vertex reduction. It is clear that these operations can simplify the tree to any desired size.

We can also think of these operations as having well-defined effects on the underlying scalar field: pruning a leaf corresponds to levelling off a maximum or minimum, while vertex reduction requires no changes.

As an example, in Figure 3 we show the result of leaf-pruning vertex 80 and edge $80-50$ from the tree. Since $80-50$ represents the left-hand maximum, pruning it flattens out the maximum, as shown in the middle terrain. Similarly, the right-hand image shows the results of reducing vertex 50 after the leaf prune. The edges incident to vertex 50 in the tree correspond to the regions above and below the contour through vertex 50. Removing vertex 50 merely combines these two regions into one.

The fact that simplification operations can be interpreted as modifying the scalar field suggests that one way to assess the cost of an operation is to measure geometric properties of the change. We show how this can be done efficiently in Section 4.

3.3 Simplification Algorithm

To simplify the contour tree, we apply the following rules:

1. Always perform vertex reduction where possible.
2. Always choose the least important leaf to prune.
3. Never prune the last up- or down- leaf at an interior vertex.

We implement this with a priority queue to keep track of the leaves of the tree with their associated pruning cost. We assume that for each edge e of the tree, we know two costs: $up(e)$ for pruning the edge from the bottom up: i.e. collapsing the edge to its upper vertex, and $down(e)$ for the cost of pruning the edge from the top downwards. We add each leaf to the priority queue, with priority of $up(e)$ for a lower leaf and $down(e)$ for an upper leaf. We then repeatedly remove the lowest cost leaf edge from the priority queue and prune it. If this pruning causes a vertex to become reducible, we do so immediately.

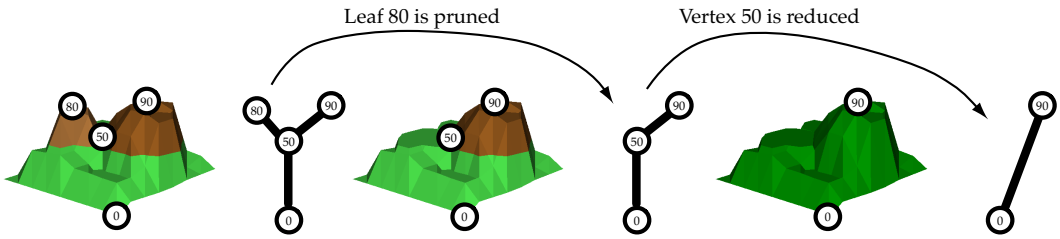


Figure 3: Leaf Pruning Levels Extrema; Vertex Reduction Leaves Scalar Field Unchanged

When a vertex is reduced, two edges e_1 and e_2 are merged into a simplified edge d. The cost of pruning d is based on the costs of the two reduced edges. Since $up(d)$ is the cost of pruning d upwards, we set it to $up(e_1)$, the cost of pruning the upper edge upwards. Similarly, we set $down(d)$ to $down(e_2)$, the cost of pruning the lower edge downwards. If d is a leaf edge, we add it to the priority queue. To simplify queue handling, we mark the reduced edges for lazy deletion. When a marked edge reaches the front of the priority queue, we discard it immediately. Similarly, when the edge removed from the queue is the last up- or down- edge at its interior vertex, we discard it, preserving it for a later vertex reduction.

A few observations on this algorithm: First, any desired level of simplification of the tree can be achieved in a number of queue operations linear in t, the size of the original tree. Since at least half the nodes are leaves, this bound is tight. And if the contour tree is stored as nodes with circular linked lists of upwards and downwards edges, every operation except (de)queueing takes constant time. As a result, the asymptotic cost of this algorithm is dominated by the $O(t \log(t))$ cost of maintaining the priority queue.

Second, the simplified contour tree can still be used to extract isosurface contours. Vertex reductions build monotone paths corresponding to the simplified edges, while leaf prunes discard entire monotone paths. Thus, any edge in a simplified contour tree corresponds to a monotone path through the original contour tree. To generate the contour at a given isovalue on a simplified edge, we perform a binary search along the contour tree edges that make up the monotone path for that simplified edge. This search identifies the unique contour tree edge that spans the desired isovalue, and we use the path seed associated with that edge to generate the contour.

Third, we extract contours from seeds as before. Instead of simplifying individual contours, we reduce the set of contours that can be extracted. Surface simplification of contours is a separate task.

Finally, $up(e)$ and $down(e)$ actually need not be set except at leaves of the tree. As a leaf is pruned and vertex reduced, new values can be computed using information from the old nodes and edges. It is not hard to show by induction that any desired level of simplification of the tree can be achieved. And, since leaf pruning and vertex reduction are the only two operations, the net result can also be a meaningful simplification of the underlying scalar field, assuming that a reasonable geometric measure is used to guide the simplification. We therefore next discuss geometric measures.

4 Local Geometric Measures

[Bajaj et al. 1997] compute global geometric properties, and display them alongside the contour tree in the *contour spectrum*. [Pascucci and Cole-McLaughlin 2002] propagate topological indices called the *Betti numbers* along branches of the contour tree, based on previous work by [Pascucci 2001]. We bring these two ideas together to compute *local geometric measures* for individual contours.

In 2D scalar fields, the geometric properties we could compute include the following contour properties: line length (perimeter), cross-sectional area (area of region enclosed by the contour), volume (of the region enclosed), and surface area (of the function over the region). In 3D scalar fields, there are analogous properties that include isosurface area, cross-sectional volume (the volume of the region enclosed by the isosurface), and hypervolume (the integral of the scalar field over the enclosed volume).

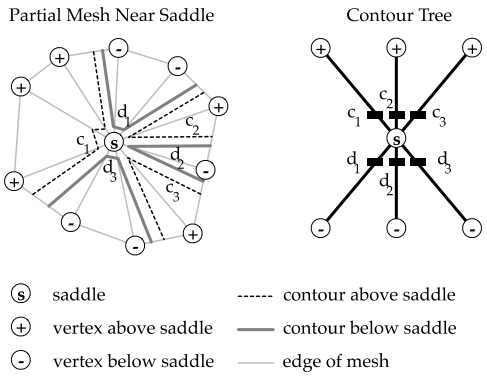


Figure 4: Contours Sweeping Past a Saddle Point

Consider a plane sweeping through the field in Figure 2 from high to low isovalues. At any isovalue h, the plane divides the field into regions above and below the plane. As the isovalue decreases, the region above the plane grows, sweeping past the vertices of the mesh one at a time. Geometric properties of this region can be written as functions of the isovalue h. Such properties are decomposable over the cells of the input data – for each cell we compute a piecewise polynomial function, and sum them to obtain a piecewise polynomial function for the entire region. [Bajaj et al. 1997] compute these functions by sweeping through the isovalues, altering the function as each vertex is passed. Figure 4 illustrates this process, showing the contours immediately above and below a vertex s. As the plane sweeps past s, the function is unchanged in cells outside the neighbourhood of s, but changes inside the neighbourhood of s. This sweep computes global geometric properties for the region above the sweep plane. Reversing the direction of the sweep computes global geometric properties for the region below the sweep plane.

In Figure 2, the region above the sweep plane at 2000m consists of two connected components, one defined by contours c_1 and c_2, the other by c_3. To compute properties for these components, we sweep along an edge of the contour tree, representing a single contour sweeping through the data. This lets us compute functions for

the central maximum at B. For the crater rim defined by contours c_1 and c_2, we use inclusion/exclusion. We sweep one contour at a time, computing properties for the region inside the contour, *including* regions above and below the isovalue of the contour. The area of the crater rim can then be computed by subtracting the area inside contour c_2 from the area inside contour c_1.

We define *local geometric measures* to be geometric properties of regions bounded by a contour. We compute these measures in a manner similar to the global sweep of [Bajaj et al. 1997], but by sweeping contours along contour tree edges.

4.1 Local Geometric Measures

To define local geometric measures attached to contour tree edges, we must be careful with terminology. *Above* and *below* do not apply to the region inside c_1 in Figure 2, since part of the region is above the contour and part is below. Nor do *inside* and *outside*, which lose their meaning for contours that intersect the boundary. We therefore define *upstart* and *downstart* regions of a contour. An upstart region is a region reachable from the contour by paths that initially ascend from the contour and never return to it. For contour c_1, there is one upstart region (inside) and one downstart region (outside). At saddles such as D, there may be several upstart regions. Since each such region corresponds to an edge in the contour tree, we refer, for example, to the upstart region at D for arc CD.

We now define *upstart* and *downstart functions*: functions computed for upstart or downstart regions. Note that the upstart and downstart functions do not have to be the same. For example, the length of a contour line is independent of sweep direction, so the upstart and downstart functions for contour length in 2D are identical. But the area enclosed by a contour depends on sweep direction, so the upstart and downstart functions will be different.

Since upstart and downstart functions describe geometric properties local to a contour, we refer to them collectively as *local geometric measures*. These measures are piecewise polynomial since they are piecewise polynomial in each cell. Because we need to track connectivity for inclusion/exclusion, they are not strictly decomposable. Stated another way, in order to make them decomposable, we need to know the connectivity during the local sweep. We are fortunate that the contour tree encodes this connectivity.

For regular data, we approximate region size with vertex count as in [Takahashi et al. 2004b]. For the integral of f over region R, we sum the sample values to get $\Sigma_{x\in R} f(x)$: the correct integral is the limit of this sum as sample spacing approaches zero. When we prune a leaf to a saddle at height h, the integral over the region flattened is $\Sigma_{x\in R}(f(x)-h) = (\Sigma_{x\in R} f(x)) - Ah$ where A is the area of region R.

In three dimensions, vertex counting measures *volume*, and summing the samples gives *hypervolume*. This geometric measure is quite effective on the data sets we have tested in Section 6.

4.2 Combining Local Geometric Measures

To compute local geometric measures, we must be able to *combine* upstart functions as we sweep a set of contours past a vertex. In Figure 4, we must combine the upstart functions for contours c_1, c_2 and c_3 before sweeping past s. We must then *update* the combined upstart function as we sweep past the vertex.

After sweeping past s, we know the combined upstart function d for contours d_1, d_2 and d_3. We *remove* the upstart functions for d_1 and d_2 from d to obtain the upstart function for d_3.

We assume that we have recursively computed the upstart functions for d_1 and d_2 by computing the downstart functions and then *inverting* them. Let us illustrate inversion, combination and removal for two local geometric measures in two dimensions.

Contour Length: Contour length is independent of sweep direction, so these operations are simple: Inversion is the identity operation, combination sums the lengths of the individual contours, and contours are removed by subtracting their lengths.

Area: Area depends on sweep direction, so inversion subtracts the function from the area of the entire field. Combining upstart functions at a saddle depends on whether the corresponding edges ascend or descend from the saddle. For ascending edges the upstart regions are disjoint, and the upstart functions are summed. For descending edges the upstart regions overlap, and the upstart functions are combined by inverting to downstart functions, summing, and re-inverting. Removing upstart functions reverses combination.

Consider Figure 4 once more. The upstart region of d_1 contains s, as well as contours c_1, c_2 and c_3. Similarly, the upstart regions of d_2 and d_3 contain s and contours c_1, c_2 and c_3. However, the downstart regions of d_1, d_2 and d_3 are disjoint, and can be summed, then inverted to obtain the combination of the upstart regions.

In general, measures of contour size are independent of sweep direction and their computation follows the pattern of 2D contour length. Such measures include surface area in three dimensions, and hypersurface volume in four dimensions. Measures of region size depend on sweep direction and their computation follows the pattern of 2D cross-sectional area. Such measures include surface area and volume in two dimensions, and isosurface cross-sectional volume and hypervolume in three dimensions.

Input : Fully Augmented Contour Tree C
A local geometric measure f with operations
Combine$(f_1,\ldots,f_m)$ local geometric measures
Update(f,v) that updates f for sweep past v
Remove$(f,f_1,\ldots,f_m)$ local geometric measures
Invert() from *down*(e) to *up*(e) or vice versa
Output : *down*(e) and *up*(e) for each edge e in C

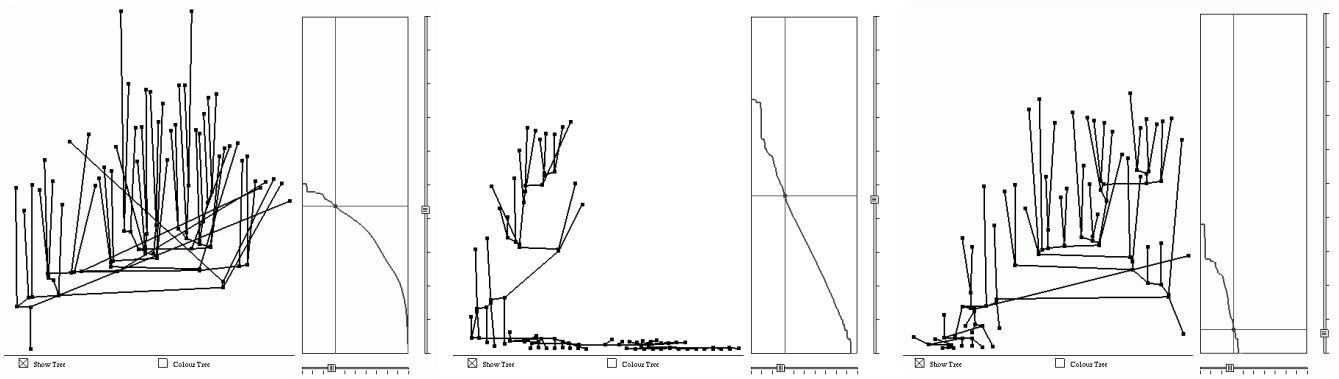
Make a copy C' of C
for each vertex v **do**
  If v is a leaf of C, enqueue v

while *NumberOfArcs*$(C') > 0$ **do**
  Dequeue v and retrieve edge $e = (u,v)$ from C'
  Without loss of generality, assume e ascends from v
  Let $d_1,\ldots,d_k$ be downward arcs at v in C
  Let *upBelow* = *Combine*(*down*$(d_1),\ldots,$*down*(d_k)
  Let *upAbove* = *Update*(*upBelow*, v)
  Let $e_1,\ldots,e_m$ be upwards arcs at v in C, with $e_1 = e$
  Let f_i = *Invert*(*down*(e_i)) for $i = 2,\ldots,m$
  Let *up*(e) = *Remove*(*upAbove*, $f_2,\ldots,f_m$)
  Let *down*(e) = *Invert*(*up*(e))
  Delete e from C'
  If u is now a leaf of C', enqueue u

Algorithm 1: Computing Local Geometric Measures

4.3 Computing Local Geometric Measures

Algorithm 1 shows how to compute edge priorities *up*(e) and *down*(e) for a given local geometric measure. This algorithm relies on *Combine*(), *Update*(), *Invert*(), and *Remove*() having been



(a) Reduced by Height (Persistence) (b) Reduced by Volume (Vertex Count) (c) Reduced by Hypervolume (Riemann Sum)

Figure 5: Comparison of Simplification Using Three Local Geometric Measures. In each case, the UNC Head data set has been simplified to 92 edges using the specified measure. Each trees were laid out using the *dot* tool, with no manual adjustment.

suitably defined, and can be integrated into the merge phase of the contour tree algorithm in [Carr et al. 2003].

The algorithm builds a queue of leaf edges in Step 2, then works inwards, pruning edges as it goes. At each vertex, including regular points, the computation described in Section 4.2 is performed, and the edge is deleted from the tree. In this way, an edge is processed only when one of its vertices is reduced to a leaf: i.e. when all other edges at that vertex have already been processed.

Unlike simplification, Algorithm 1 requires the *fully augmented* contour tree, which is obtained by adding every vertex in the input mesh to the contour tree. This makes the algorithm linear in the input size n rather than the tree size t: it cannot be used with the algorithms of [Pascucci and Cole-McLaughlin 2002] and [Chiang et al. 2002], which reduce running time by ignoring regular points.

4.4 Comparison of Local Geometric Measures

In Figure 5, we show the results of simplifying the UNC Head data set with three different geometric measures: height (persistence), volume, and hypervolume. In each case, the contour tree has been reduced to 92 edges and laid out using *dot* with no manual intervention.

In the left-hand image, height (persistence) is used as the geometric measure. All of the edges shown are tall as a result, but on inspection, many of these edges are caused by high-intensity voxels in the skull or in blood vessels. Most of the corresponding objects are quite small, while genuine objects of interest such as the eyes, ventricular cavities and nasal cavity have already been suppressed, because they are defined by limited ranges of voxel intensity. Also, on the corresponding simplification curve, we observe that there are a relatively large number of objects with large intensity ranges: again, on further inspection, these tended to be fragments of larger objects, particularly the skull.

In comparison, the middle image shows the results of using volume (i.e. vertex count) as the geometric measure. Not only does this focus attention on a few objects of relatively large spatial extent, but the simplification curve shows a much more rapid drop-off, implying that there are fewer objects of large volume than there are of large height. Objects such as the eyeballs are represented, as they have relatively large regions despite having small height. However, we note that there are a large number of small-height edges at the bottom of the contour tree. These edges turn out to be caused by noise and artifacts outside the skull in the original CT scan, in which large regions are either slightly higher or lower in isovalue than the surrounding regions.

Finally, the right-hand image shows the results of using hypervolume (the sum of sample values, as discussed above). In this case, we see a very rapid dropoff of importance in the simplification curve, with only 100 or so regions having significance. We note that this measure preserves small-height features such as the eyeballs, while eliminating most of the apparent noise edges at the bottom of the tree, although at the expense of representing more skull fragments than the volume measure. In general we have found that this measure is better for data exploration than either height or volume, since it balances representation of tall objects with representation of large objects.

We do not claim that this measure is universally ideal: the choice of simplification measure should be driven by domain-dependent information. However, no matter what measure is chosen, the basic mechanism of simplification remains.

5 Implementation

We have combined simplification with the flexible isosurface interface of [Carr and Snoeyink 2003], which uses the contour tree as a visual index to contours. The interface window, shown in Figures 1, 6, and 7, is divided into data, contour tree, and simplification curve panels. The data panel displays the set of contours marked in the contour tree panel. Contours can be selected in either panel, then deleted, isolated, or have their isovalue adjusted. The simplification curve panel shows a log-log plot of contour tree size against "feature size": the highest cost of any edge pruned to reach the given level of simplification. Selecting a point on this curve determines the detail shown in the contour tree panel.

For efficiency, we compute contour trees for the surfaces given by the Marching Cubes cases of [Montani et al. 1994] instead of a simplicial or trilinear mesh, because these surfaces generate roughly 60% fewer triangles than even a minimal simplicial subdivision of the voxels, with none of the directional biases identified by [Carr et al. 2001], and because they are significantly simpler to

compute than the trilinear interpolant used by [Pascucci and Cole-McLaughlin 2002]. There is a loss of accuracy, but since our simplification discards small-scale details of the topology anyway, little would be gained from more complex interpolants.

Finally, as in [Carr et al. 2003; Pascucci and Cole-McLaughlin 2002; Chiang et al. 2002], we use simulation of simplicity [Edelsbrunner and Mücke 1990] to guarantee uniqueness of isovalues, then collapse zero-height edges in the tree. Implementation details can be found in [Carr 2004].

6 Results and Discussion

We used a variety of data sets to test these methods, including results from numerical simulations (Nucleon, Silicium, Fuel, Neghip, Hydrogen), analytical methods (ML, Shockwave), CT-scans (Lobster, Engine, Statue, Teapot, Bonsai), and X-rays (Aneurysm, Foot, Skull). Table 1 lists the size of each data set, the size of the unsimplified contour tree, the time for constructing the unsimplified contour tree, and the simplification time. Times were obtained using a 3 GHz Pentium 4 with 2 GB RAM, and the *hypervolume* measure.

Data Set	Data Size	Tree Size	CT (s)	ST (s)
Nucleon	41× 41× 41	49	0.28	0.01
ML	41× 41× 41	695	0.25	0.01
Silicium	98× 34× 34	225	0.41	0.01
Fuel	64× 64× 64	129	0.72	0.01
Neghip	64× 64× 64	248	0.90	0.01
Shockwave	64× 64×512	31	5.07	0.01
Hydrogen	128×128×128	8	5.60	0.01
Lobster	301×324× 56	77,349	19.22	0.10
Engine	256×256×128	134,642	31.51	0.18
Statue	341×341× 93	120,668	32.20	0.15
Teapot	256×256×178	20,777	33.14	0.02
Aneurysm	256×256×256	36,667	41.83	0.04
Bonsai	256×256×256	82,876	49.71	0.11
Foot	256×256×256	508,854	67.20	0.74
Skull	256×256×256	931,348	109.73	1.47
CT Head	106×256×256	92,434	21.30	0.12
UNC Head	109×256×256	1,573,373	91.23	2.48
Tooth	161×256×256	338,300	39.65	0.48
Rat	240×256×256	2,943,748	233.33	4.97

Table 1: Data sets, unsimplified contour tree sizes, and contour tree construction time (CT) and simplification time (ST) in seconds.

The size of the contour tree is proportional to the number of local extrema in the input data. For analytic and simulated data sets, such as the ones shown in the upper half of Table 1, this is much smaller than the input size. For noisy experimentally acquired data, such as the ones shown in the lower half of Table 1, the size of the contour tree is roughly proportional to the input size. The time required to simplify the contour tree using local geometric measures is typically less than one percent of the time of constructing the original contour tree, plus the additional cost of pre-computing these measures during contour tree construction.

6.1 Examples of Data Exploration

Figure 1 shows the result of exploring of the UNC Head data set using simplified contour trees. An appropriate level of simplification was chosen on the simplification curve and individual contours explored until the image shown was produced. Surfaces identifiable as part of the skull were not chosen because they occluded the view of internal organs, although two contours for the ventricular system were chosen despite being occluded by the brain surrounding them. The flexible isosurface interface is particularly useful in this context because it lets one manipulate a single contour at a time, as shown in the video submitted with this paper.

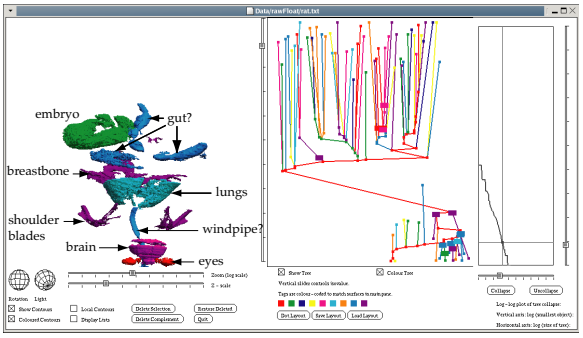


Figure 6: A Pregnant Rat MRI ($240 \times 256 \times 256$). Despite low quality data, simplifying the contour tree from 2,943,748 to 125 edges allows identification of several anatomical features.

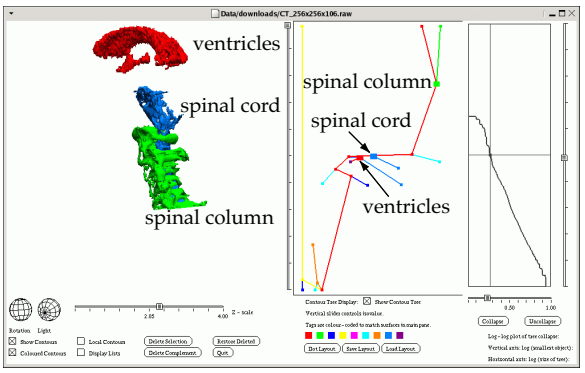


Figure 7: CT of a Skull ($256 \times 256 \times 106$). Simplification of the contour tree from 92,434 to 20 edges isolates the ventricular cavity, spinal cord and spinal column.

Similarly, Figure 6 shows the result of a similar exploration of a $240 \times 256 \times 256$, low-quality MRI scan of a rat from the *Whole Frog Project* at `http://www-itg.lbl.gov/ITG.hm.pg.docs/Whole.Frog/Whole.Frog.html`. Again, simplification reduces the contour tree to a useful size. Figure 7 shows a spinal column, spinal cord and ventricular cavity identified in a $256 \times 256 \times 106$ CT data set from the University of Erlangen-Nuremberg. Other examples may be seen on the accompanying video.

Each of these images took less than 10 minutes to produce after all pre-processing, using the *dot* tool from the graphviz package (`http://www.research.att.com/sw/tools/graphviz/`) to lay out the contour tree: we generally then made a few adjustments to the node positions for clarity. Although *dot* produces reasonable layouts for trees with 100 – 200 nodes, it is slow, sometimes taking several minutes, and the layout computed usually becomes unsatisfactory as edges are added or subtracted from the tree.

Note that in none of these cases was any special constant embedded in the code – the result is purely a function of the topology of the isosurfaces of the input data.

7 Conclusions and Future Work

We have presented a novel algorithm for the simplification of contour trees based on local geometric measures. The algorithm is *online*, meaning that simplifications can be done and undone at any time. This addresses the scalability problems of the contour tree in exploratory visualization of 3D scalar fields. The simplification can also be reflected back onto the input data to produce an on-line simplified scalar field. The algorithm is driven by local geometric measures such as area and volume, which make the simplifications meaningful. Moreover, the simplifications can be tailored to a particular application or data set.

We intend to explore several future directions. We could compute a multi-dimensional feature vector of local geometric measures, and allow user-directed simplification of the contour tree, with different measures being applied in different regions of the function.

The simplified contour tree also provides a data structure for queries. With local feature vectors one could efficiently answer queries such as "Find all contours with volume of at least 10 units and an approximate surface-area-to-volume ratio of 5." If information about spatial extents (e.g., bounding boxes) is computed, then spatial constraints can also be included. Inverse problems could also be posed – given examples of a feature (e.g., a tumor), what should the query constraints be to find such features?

Some interface issues still need resolution, such as finding a fast contour tree layout that is clear over a wide range of levels of simplification but which also respects the convention that the y-position depends on the isovalue. We would also like to annotate contours using the flexible isosurface interface, rather than after the fact as we have done in Figure 1 and Figures 6 – 7, and to enable local simplification of the contour tree rather than the single-parameter simplification presented here.

Isosurfaces are not the only way of visualizing volumetric data. Other methods include boundary propagation using level set methods or T-snakes. We believe that simplified contour trees can provide seeds for these methods, either automatically or through user interaction. We are adapting the flexible isosurface interface to generate transfer functions for volume rendering. These transfer functions would add spatial locality to volume rendering, based on the regions corresponding to edges of the simplified contour tree.

Another possible direction is to develop more local geometric measure for multilinear interpolants. Lastly, the algorithms we describe work in arbitrary dimensions, but special consideration should be given to simplification of contour trees for time-varying data.

8 Acknowledgements

Acknowledgements are due to the National Science and Engineering Research Council of Canada (NSERC) for support in the form of post-graduate fellowships and research grants, and to the U.S. National Science Foundation (NSF) and the Institute for Robotics and Intelligent Systems (IRIS) for research grants. Acknowledgements are also due to those who made volumetric data available at volvis.org and other sites.

References

BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. 1997. The Contour Spectrum. In *Proceedings of IEEE Visualization 1997*, 167–173.

BENTLEY, J. L. 1979. Decomposable searching problems. *Inform. Process. Lett. 8*, 244–251.

BREMER, P.-T., EDELSBRUNNER, H., HAMANN, B., AND PASCUCCI, V. 2003. A Multi-resolution Data Structure for Two-dimensional Morse-Smale Functions. In *Proceedings of IEEE Visualization 2003*, 139–146.

BRODLIE, K., AND WOOD, J. 2001. Recent advances in volume visualization. *Computer Graphics Forum 20*, 2 (June), 125–148.

CARR, H., AND SNOEYINK, J. 2003. Path Seeds and Flexible Isosurfaces: Using Topology for Exploratory Visualization. In *Proceedings of Eurographics Visualization Symposium 2003*, 49–58, 285.

CARR, H., MÖLLER, T., AND SNOEYINK, J. 2001. Simplicial Subdivisions and Sampling Artifacts. In *Proceedings of IEEE Visualization 2001*, 99–106.

CARR, H., SNOEYINK, J., AND AXEN, U. 2003. Computing Contour Trees in All Dimensions. *Computational Geometry: Theory and Applications 24*, 2, 75–94.

CARR, H. 2004. *Topological Manipulation of Isosurfaces*. PhD thesis, University of British Columbia, Vancouver, BC, Canada.

CHIANG, Y.-J., AND LU, X. 2003. Progressive Simplification of Tetrahedral Meshes Preserving All Isosurface Topologies. *Computer Graphics Forum 22*, 3, to appear.

CHIANG, Y.-J., LENZ, T., LU, X., AND ROTE, G. 2002. Simple and Output-Sensitive Construction of Contour Trees Using Monotone Paths. Tech. Rep. ECG-TR-244300-01, Institut für Informatik, Freie Universtät Berlin.

EDELSBRUNNER, H., AND MÜCKE, E. P. 1990. Simulation of Simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics 9*, 1, 66–104.

EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. 2002. Topological persistence and simplification. *Discrete Comput. Geom. 28*, 511–533.

EDELSBRUNNER, H., HARER, J., AND ZOMORODIAN, A. 2003. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete Comput. Geom. 30*, 87–107.

HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. 2001. Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH 2001*, 203–212.

KETTNER, L., ROSSIGNAC, J., AND SNOEYINK, J. 2001. The Safari Interface for Visualizing Time-Dependent Volume Data Using Iso-surfaces and Contour Spectra. *Computational Geometry: Theory and Applications 25*, 1-2, 97–116.

LORENSON, W. E., AND CLINE, H. E. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics 21*, 4, 163–169.

MATSUMOTO, Y. 2002. *An Introduction to Morse Theory*. AMS.

MILNOR, J. 1963. *Morse Theory*. Princeton University Press, Princeton, NJ.

MONTANI, C., SCATENI, R., AND SCOPIGNO, R. 1994. A modified look-up table for implicit disambiguation of Marching Cubes. *Visual Computer 10*, 353–355.

PASCUCCI, V., AND COLE-MCLAUGHLIN, K. 2002. Efficient Computation of the Topology of Level Sets. In *Proceedings of IEEE Visualization 2002*, 187–194.

PASCUCCI, V. 2001. On the Topology of the Level Sets of a Scalar Field. In *Abstracts of the 13th Canadian Conference on Computational Geometry*, 141–144.

REEB, G. 1946. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus de l'Acadèmie des Sciences de Paris 222*, 847–849.

TAKAHASHI, S., FUJISHIRO, I., AND TAKESHIMA, Y. 2004. Topological volume skeletonization and its application to transfer function design. *Graphical Models 66*, 1, 24–49.

TAKAHASHI, S., NIELSON, G. M., TAKESHIMA, Y., AND FUJISHIRO, I. 2004. Topological Volume Skeletonization Using Adaptive Tetrahedralization. In *Geometric Modelling and Processing 2004*.

TARASOV, S. P., AND VYALYI, M. N. 1998. Construction of Contour Trees in 3D in $O(n \log n)$ steps. In *Proceedings of the 14th ACM Symposium on Computational Geometry*, 68–75.

VAN KREVELD, M., VAN OOSTRUM, R., BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. 1997. Contour Trees and Small Seed Sets for Isosurface Traversal. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, 212–220.