Computational Geometry 43 (2010) 42–58

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree

Hamish Carr [a,*], Jack Snoeyink [b], Michiel van de Panne [c]

[a] Dept. of Computer Science, University College Dublin, Belfield, Dublin, Ireland
[b] Dept. of Computer Science, University of North Carolina at Chapel Hill, NC, USA
[c] Dept. of Computer Science, University of British Columbia, Vancouver, BC, Canada

A R T I C L E   I N F O

Article history:
Received 1 February 2006
Accepted 1 May 2006
Available online 24 March 2009
Communicated by J. Iacono

Keywords:
Morse theory
Contour trees
Visualization
Isosurfaces

A B S T R A C T

The contour tree is an abstraction of a scalar field that encodes the nesting relationships of isosurfaces. We show how to use the contour tree to represent individual contours of a scalar field, how to simplify both the contour tree and the topology of the scalar field, how to compute and store geometric properties for all possible contours in the contour tree, and how to use the simplified contour tree as an interface for exploratory visualization.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

One of the fundamental tools for rendering and segmenting three-dimensional scalar fields is the *isosurface*. Isosurfaces, the three-dimensional analogue of contour lines on a topographic map, show the surface defined by a specified value of the scalar field, called the *isovalue*. An isosurface may consist of several connected components: we refer to the connected components of the isosurface as *contours*.

Isosurfaces can be rendered directly, or used to define transfer functions for volume rendering. In either case, the first task is to establish a suitable isovalue for which the isosurface captures all of the information of interest in the data set. In many applications, a suitable isosurface is already known, especially when the task has been performed many times.

In other applications, although boundaries are still of interest, the specific isovalue of interest is unknown. An example of this is the boundary between the heart and the rest of the body in a medical data set. In some cases, it is possible to select a suitable isovalue automatically, based on the gradient of the field [31]. In other cases, user interaction is required to find a suitable isovalue.

In comparison, computational fluid dynamics and molecular modelling often give rise to data sets where the specific isovalue of interest is not immediately obvious. Where no *a priori* isovalue is known, successful visualization usually involves human-directed exploration of the data. The user tries different isovalues interactively, until a suitable one is found. As a result, interfaces that give cues to suitable isovalues can aid the human's exploration of the data.

Specifying a single isovalue gives a set of contours that may either enclose or occlude each other. While it is convenient to have a single-parameter control, if the enclosed or occluded contour represents the principal object of interest, the remaining contours actually impede a user in their desire to understand the data. Moreover, the relationships between features defined at different isovalues may also be of interest.

* Corresponding author.
E-mail addresses: hamish.carr@ucd.ie (H. Carr), snoeyink@cs.unc.edu (J. Snoeyink), van@cs.ubc.ca (M. van de Panne).

0925-7721/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comgeo.2006.05.009

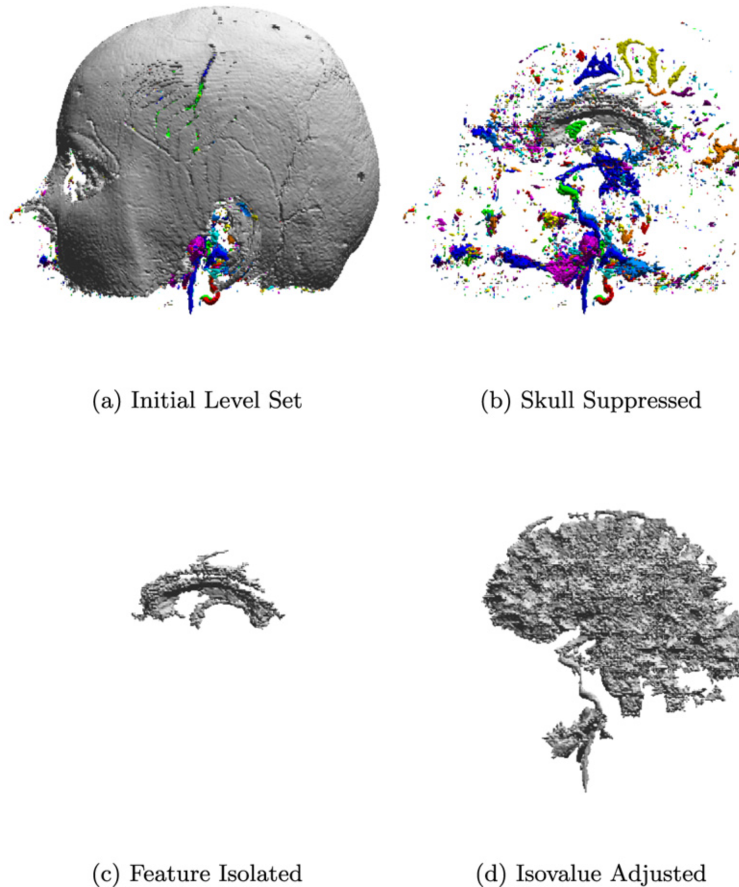

Fig. 1. Example of flexible isosurface editing to isolate the brain.

This paper, therefore, generalizes from a fixed set of contours sharing a single isovalue to a *flexible isosurface*, or arbitrary set of contours, and shows how to use topological analysis, in the form of the *contour tree*, to keep rigorous track of the contours in the set, and how to construct an interface based on the principle that the user should be given both direct control of the contours and their isovalues, and indirect control through the contour tree.

In essence, this paper develops the *contour spectrum* of Bajaj, Pascucci and Schikore [1] from an essentially passive display of the topology of a data set to a control paradigm for actively manipulating individual contours. Although significant portions of this work have been presented previously at conferences [4,6], each presents only a portion of the complete interface, and this paper consolidates this work and provides further details of the semantic and topological infrastructure necessary to achieve this.

Fig. 1 depicts images for a scenario in which a radiologist wishes to segment the brain in an MRI scan for further processing. Using conventional isosurfaces, the radiologist might get the image in Fig. 1(a), in which the skull occludes the brain isosurface. In our interface, the radiologist can suppress the display of this skull isosurface and make objects inside the skull visible (Fig. 1(b)). With the occluding surface removed, the radiologist could select a particular object, such as the one shown in grey, suppress all other surfaces (Fig. 1(c)), then vary the isovalue of the chosen surface (Fig. 1(d)).

There are many scenarios in which a user is interested in seeing some, but not all, of the features in the data. For exploring scalar fields, our work proceeds from a basic assumption:

Assumption 1. Some interesting features of a scalar field are represented by connected components of level sets (contours), but not all contours are interesting or important.

Because the contour tree captures the evolution of all contours for different isovalues, and their changes in topology, this assumption leads in a natural way to employing the contour tree as part of a visualization interface. As we will see below, the contour tree can be used to focus attention on one contour at a time or as a cue to locate features of specific interest.

In addition to using the contour tree indirectly, as in the radiology scenario just cited, we also provide the ability to use the contour tree directly as a visual interface to specify contours of interest. Unfortunately, large experimentally acquired data sets typically have extremely complex topology – in the case of Fig. 1, over a million edges, most of which repre-

sent extremely small-scale topological features due to noise. Providing abstract control over the contours to be displayed therefore relies on simplifying the contour tree in a consistent and rigorous fashion. This in turn depends on the ability to quantify the importance of individual topological features.

The principal contributions of this paper are therefore:

1. To use the *contour tree* as an index structure to keep track of contours and their relationships.
2. To use *path seeds* to support extraction of individual contours, without extracting other contours at the same isovalue.
3. To quantify feature importance with *local geometric measures*.
4. To *simplify* the contour tree, guided by these local geometric measures.
5. To define an *interface* by which a user can specify which contours are interesting in a principled fashion.

We also note that although path seeds, topological simplification and local geometric measures were developed to support the flexible isosurface interface, they do not depend on it, and can be used independently to produce contour trees at any desired level of simplification or any desired bound on the geometry of the simplification.

2. Morse theory and the contour tree

For a scalar field $f : \mathcal{R} \subseteq \mathbb{R}^d \to \mathbb{R}$, the *level set* of an *isovalue* h is the set $L(h) = \{x \in \mathcal{R}\colon\ f(x) = h\}$. A *contour* is a connected component of a level set. For $d = 1$, calculus students are familiar with the principle that the slope of a curve must be 0 at maxima and minima, but can also be 0 at points of inflection. For scalar fields defined over arbitrary manifolds, Morse theory [24,25] generalizes this analysis by considering the relationships between *critical points*: points at which the gradient of the function is 0. These critical points are classified as *peaks* or local maxima, *pits* or local minima, and *saddle points*. All other points are called *regular points*. Note that saddle points may correspond either to changes in the connectivity of the contours, changes in the genus of the contours, or in general, to changes in the Betti numbers of the contours [30].

To avoid degenerate conditions, however, Morse theory assumes that all critical points have unique (i.e. different) isovalues, and that there are no flat regions – i.e. regions of measure greater than zero with 0 gradient. Computationally, these requirements are usually guaranteed by simulation of simplicity [10], which tilts the entire manifold by an infinitesimal amount to ensure that all critical points and mesh vertices have unique isovalues.

The branch of Morse theory of interest to us in this work deals with the Reeb graph [32,33] – the graph of topological relationships between contours that is obtained by contracting each contour to a single point while preserving continuity between contours – i.e. any continuous path in the domain contracts to a continuous path in the Reeb graph. For simple scalar fields whose domain is homologous to $\mathbb{R}^d$, this graph is a tree called the *contour tree* [2]. By convention, edges in the contour tree are referred to as *superarcs* and vertices as *supernodes*. Since the tree only forks where the contour connectivity changes, the supernodes of the contour tree are all critical points, but not vice versa. In particular, critical points at which the connectivity of the contours does not change need not be supernodes. Where this distinction is important, we will use *connectivity critical points* to refer to critical points at which connectivity changes and *connectivity regular points* to refer to all other critical and regular points.

As an example, Fig. 2 shows a scalar field in $f : \mathbb{R}^2 \to \mathbb{R}$ representing a volcanic crater lake with a central island. The contour tree of f depicts the relationships of all local maxima, minima, and saddle points; individual contours are represented uniquely as points on contour tree edges. For example, contours c_1, c_2, and c_3 share a single isovalue but each has a unique location in the contour tree.

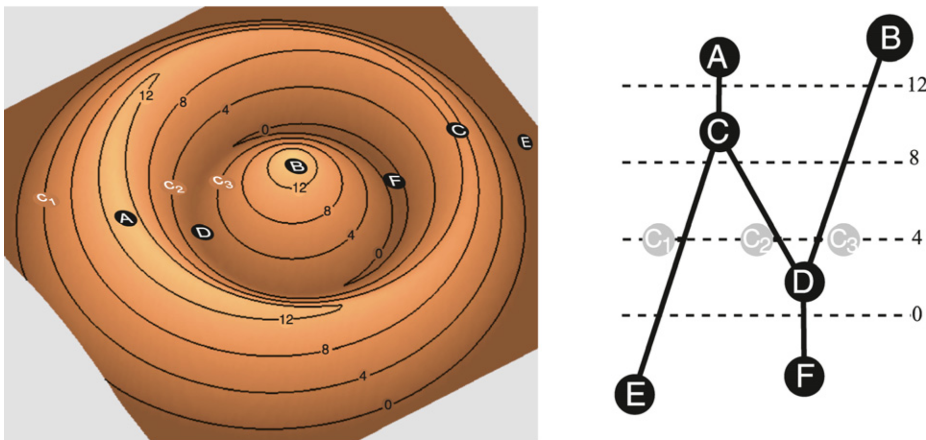

Fig. 2. Surface rendering and contour tree of a volcanic crater lake with a central island. A: maximum on crater edge; B: maximum on central peak; C and D: saddle points; E,F: minima.

This one-to-one correspondence between the contour tree and individual contours of the underlying scalar field is the basis for the *flexible isosurface* interface, in which individual contours are manipulated visually by a user. We therefore represent a single contour symbolically as follows:

Convention 1. A contour c is a single connected component of a level set at isovalue h, and can be denoted by a pair $c = (s, h)$ consisting of a superarc s in the contour tree and an isovalue h. Correspondingly, we use $c \subset s$ to indicate that a contour c *belongs* to a superarc s.

At a supernode, the corresponding contour can be expressed as more than one pair $c_1 = (s_1, h)$, $c_2 = (s_2, h), \ldots$, each of which corresponds to a specific superarc incident to the supernode. These representations are, however, redundant, as there is only contour exactly at the isovalue of the supernode. We will use this redundant representation in Section 5, but will otherwise assume that contours are never to be extracted exactly at the supernodes. This assumption is a form of *symbolic perturbation* and is common to all isosurface extraction methods, including marching cubes [22] in which vertices are classified as above or below the isosurface, never exactly at the isosurface.

Regardless of our representation of contours at supernodes, the 1–1 correspondence between contours and points in the tree also implies a useful property:

Property 1. A monotone path in the domain $\mathcal{R} \subseteq \mathbb{R}^d$ of f maps to a monotone path in the contour tree; any monotone path in the contour tree is the image of at least one monotone path in $\mathcal{R}$.

3. Previous work

This paper deals with the application of the contour tree in an exploratory interface. Thus, relevant previous work includes isosurface extraction, contour tree algorithms, interfaces for scalar field exploration, local surface extraction, automated or guided isovalue selection, and transfer function design.

Isosurface extraction Isosurface extraction techniques start with two fundamental papers: *marching cubes* [22], and *continuation* [44], both of which assume that a polyhedral *mesh* underlies the scalar field. In each case, the isosurface is separately approximated in each cell of the mesh with a triangulated surface. *Marching cubes* tests every cell of the mesh but does not in general extract one contour at a time, instead triangulating individual cells in arbitrary order. *Continuation*, on the other hand, propagates outwards from an initial *seed* to generate the isosurface. With minor modifications, this can be performed one contour at a time. Since we are interested in extracting single contours, we use the *continuation* method instead of marching cubes. Subsequent researchers [14,15,42] have used the contour tree or variants thereof to generate a list of seed cells that intersects every possible isosurface, but have not done so in a way that provides a correspondence between individual seed cells and individual contours. We will give more details of the actual mechanics of this work in Section 4.

Contour tree The contour tree has been used in two and three dimensions to index contours [2,4,42], to describe terrain [11,20,35,37] or volumetric data [1,33], to detect features [39,45], to extract isosurfaces [4,42], to simplify data [6,8], to design transfer functions [39] and to extract contour properties [6,17]. Algorithms for computing the contour tree have been described by Takahashi et al. [37], van Kreveld et al. [42], Carr, Snoeyink and Axen [5], Pascucci and Cole-McLaughlin [30], Takahashi et al. [39] and by Chiang et al. [7]. Pascucci [28] and Pascucci and Cole-McLaughlin [30] have also computed topological indices called the Betti numbers while computing the contour tree. For the purposes of this paper, we will assume that the contour tree has been computed for a simplicial mesh with one of these algorithms, although this assumption is not necessary.

Local surface choice Silver [34] introduced *object-oriented visualization*, in which individual contours were defined by expanding isovalued surfaces from local maxima. Manders et al. [23] described *Largest Contour Segmentation* in which a feature was defined to be the largest surface containing only one local maximum. Shinagawa, Kunii and Kergosien [33] gave a visual code for contour changes but did not use it to build a user interface. Thus, although isolation of individual contours was feasible, the user was given little or no control over the set of contours chosen.

Isosurface selection Important isovalues can be chosen automatically or semi-automatically. Bajaj, Pascucci and Schikore [1] described the *Contour Spectrum*, drawing the contour tree as a visual cue to the user that showed which isovalues corresponded to topological changes in the data. Variations of this have been described by Takahashi et al. [38,39]. Kettner, Rossignac and Snoeyink [17] modified the Contour Spectrum in an interface called SAFARI, which used a color map to represent isosurface connectivity with respect to time and isovalue, using the contour tree as an indirect cue to interesting regions in the data. Pekar, Wiemker and Hempel [31] used a gradient histogram for automated detection of isosurface boundaries with high gradient. All of these approaches, however, assume that the same isovalue is uniformly significant over the entire scalar field.

Transfer function design Volume rendering [21] visualizes fields as if the underlying physical system is partially opaque to visible light. A *transfer function* is defined that maps isovalues to color and opacity: rendering then integrates the light projecting to the eye through each pixel of the image. Details of the volume rendering are controlled by a *transfer function* which maps isovalue to color and opacity. Kniss, Kindlmann and Hansen [19] noted that the common definition of a transfer function assumes that a given isovalue has uniform meaning throughout the data set, and added gradient information as a parameter. Kindlmann et al. [18] later added curvature as another parameter, but no information as to spatial or topological locality. Similarly, Tenginakai, Lee and Machiraju [41] use statistical signatures of the local distribution of voxel values to define a transfer function.

With few exceptions, therefore, previous authors have applied the same map of isovalue to color and opacity uniformly across an entire dataset. As we will see later, we relax this assumption both by permitting selection of individual contours and by providing a wide choice of geometric measures of feature importance.

4. Path seeds for single contour extraction

As we have seen above, the *continuation* method can be used to extract individual contours one at a time, provided that a suitable source of *seed cells* is available. In this section, we will see how to replace the *minimal seed sets* of van Kreveld et al. [42] with *path seeds*, which generate individual seeds for each contour rather than a set of seeds for an entire isosurface.

Since each cell in a simplicial mesh corresponds to a monotone path in the contour tree, van Kreveld et al. [42] selected a *minimal seed set* of cells whose monotone paths covered the entire contour tree. Because a minimal seed set is guaranteed to intersect every contour at least once, a minimal seed set then provides a sufficient set of isosurface seeds for the continuation method.

This algorithm has several undesirable properties. First, although the minimal seed set can be computed in polynomial time, van Kreveld et al. [42] do not specify the polynomial. Instead, they present an algorithm which requires linear storage and $O(n\log^2 n)$ time in two dimensions, linear storage and $O(n^2)$ time in higher dimensions, and generates at most twice the minimum number of seed cells. Second, the size of the seed set is at least $\Omega(t)$, and potentially as much as $\Theta(n)$ for a t-supernode contour tree in a n-vertex mesh. Third, more than one seed cell can intersect a given contour, requiring additional processing to avoid redundant contours. And fourth, while only one contour can intersect a given seed cell in a simplicial mesh, multiple contours can intersect a given seed cell for other interpolants, making it difficult to isolate a single contour.

Itoh, Yamaguchi and Koyamada [15] generate *extrema graphs* as seed sets. These graphs start with the entire mesh as a seed set and discarding redundant cells until no more could be discarded. While not provably optimal, this algorithm is effective and has since been improved [13]. Still, this approach stores more seed cells than a minimal seed set. It also fails to provide individual seed cells for each contour, and may deliver multiple seed cells for a given contour.

To provide contour-specific seeds, we generate seed edges that intersect only the desired contour instead of seed cells. We choose these seed edges so that they form a set of monotone *seed paths* starting at connectivity critical points. We use paths similar to those that Takeshima et al. [37] and Chiang et al. [7] use to compute the contour tree.

For a given contour $c = (s, h)$ at an isovalue h on a superarc s, we know from Property 1 that any monotone path in f that passes through c in the field maps to a monotone path in the tree that passes through c in the tree, and vice versa. We will construct a monotone path in f that intersects c based on a monotone path in the tree that starts at v, one of the supernodes on the superarc. It is clear that the portion of the superarc from v to c is a monotone path, and we need only construct a corresponding monotone path in the field.

We assume that the field is defined over a simplicial mesh, and that we wish to construct an ascending path to contour c using edges of the mesh. Provided that the first edge in our path maps to superarc s, subsequent edges in the path are guaranteed to map to s as well, until the path ascends past the supernode w at the upper end of the superarc. Thus, in order to initialize our path, we need only associate a single mesh edge e with each superarc s in the contour tree as a representative of all the monotone paths that map to s.

As an example, consider Fig. 3, in which the edges with black arrows mark monotone seed paths starting at connectivity critical points. Each path maps to a single superarc in the contour tree shown in the figure. To generate seed cells for contours corresponding to a superarc, we follow the corresponding path through the mesh until we reach the desired isovalue. For example, to generate the contour at isovalue 49 along the superarc 30–50, we start with the edge H, then ascend along the marked seed path until we reach the isovalue 49. Conveniently, except for the first edge in each path, we can follow any ascending path and still generate valid seed cells. We call the first edge the *path seed*, and store it in the contour tree, as shown in the right-hand figure.

Since at most two path seeds are stored for each superarc, the path seeds take $\Theta(t)$ storage for a tree of size t and are minimal in the sense that no stored set of seed cells can be smaller, at least in a simplicial mesh, where at most one minimum and one maximum can belong to each simplex. This being the case, in a tree of size t, there must be at least $\Omega(t)$ simplices in the stored set.

At run-time, contour extraction using path seeds is slightly more expensive than minimal seed sets because the algorithm has to follow the seed path through the mesh. Although this additional cost can be $\Omega(n)$ in the worst case, in practice it is much smaller, as shown in Table 1.

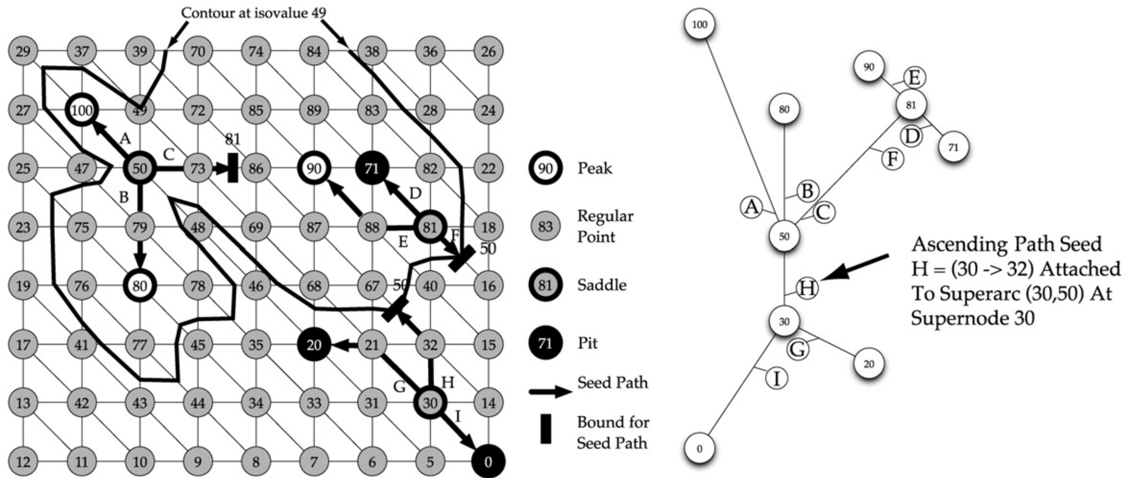

Fig. 3. Seed paths in a small simplicial mesh in 2-D.

Conveniently, these path seeds can be obtained during the construction of the contour tree. Current algorithms [5,7,30] merge two partial structures called the *join tree* and *split tree* to obtain the contour tree. The join and split trees are constructed by two isovalue-order sweeps through the data, detecting joins with Tarjan's union-find structure [40]. Since a join or split occurs when edges are added to the union-find structure, if there are two distinct directions of descent or ascent, the very edge that is used to detect a union of two component also serves as a path seed, and is stored accordingly. In later stages, the contour tree is computed by transferring arcs from the join tree and split tree, then collapsing arcs into superarcs. In these steps, superfluous path seeds are discarded.

The modification also applies to optimized versions of the algorithm, and leaves the asymptotic running time unchanged at $O(n \log n + t\alpha(t))$. Since there is exactly one path seed for each of t superarcs, it follows that path seeds are cheaper to compute and store than the minimal seed sets of van Kreveld et al. [42], especially in dimensions higher than two.

While it is possible for a given superarc to have one path seed at the upper supernode and one at the lower supernode, these are redundant. Therefore, to generate a seed for any contour on this superarc, we choose any path seed on the superarc and use it to generate *exactly* one seed per contour. This is fundamental to our interface, which tracks individual contours as separate objects. Before discussing the interface, however, we show how to compute geometric properties of contours and how to use them to simplify the contour tree in a meaningful fashion.

5. Local geometric measures

As we saw in the introduction, path seeds are not the only innovation required for the flexible isosurface interface. We also require topological simplification of the contour tree and the data set, and *local geometric measures* for guiding the simplification. In this section we introduce these measures before dealing with simplification in the next section. These measures will include contour length or surface area, cross-sectional area or volume, contained volume or hypervolume, and statistical measures such as mean, standard deviation and root mean squared error of samples in regions bounded by the contour.

In their contour spectrum paper, Bajaj, Pascucci and Schikore [1] considered regions of the form $\{x \in \mathbb{R}^d\colon\ f(x) \geqslant h\}$ in simplicial meshes, and showed that geometric properties such as contour size and volume were piecewise-polynomial functions of the isovalue h. Edelsbrunner, Harer and Zomorodian [9] computed the *persistence* of a topological feature: the difference between the isovalue at which the feature is *created* and that at which it is *destroyed*, based on pairing saddle points with local extrema. Pascucci and Cole-McLaughlin [30], Takahashi, Takeshima and Fujishiro [39] and Bremer, Edelsbrunner and Hamann [3] have all used this measure of feature importance. Zhang, Bajaj and Baker [45] approximate region size by counting samples along the arcs of the contour tree, as do Mizuta and Matsuda [26]. Pascucci and Cole-McLaughlin [30] also compute Betti numbers along the arcs of the contour tree to determine topological genus of individual contours.

We generalize these approaches by computing arbitrary geometric properties efficiently for all contours in the data set. We call these properties *local geometric measures* since they are local to a given contour, geometric in nature and useful for measuring the importance of features.

We must however be careful with terminology. *Above* and *below* do not apply to the region inside c_1 in Fig. 2, which is partly above and partly below the isovalue of c_1. *Inside* and *outside* lose their natural meaning for contours that intersect the boundary. We instead subtract the contour from the domain of f, and identify the connected components of its complement according to whether one reaches them by ascending or descending from the contour. Since monotone paths in the contour tree correspond to monotone paths in the space, our definition starts from the contour tree.

For a given contour $c = (s, h)$, we divide the inverse image of its superarc s into *upper* and *lower* regions: $R^+(c) = \bigcup\{c^+ = (s, h^+)\colon\ c^+ \subset s \text{ and } h^+ > h\}$ and $R^-(c) = \bigcup\{c^- = (s, h^-)\colon\ c^- \subset s \text{ and } h^- < h\}$.

We then observe that, except for contours through connectivity critical points, the entire domain can be divided into two regions – one which can only be reached by paths that intersect the upper region, and one which can only be reached by paths that intersect the lower region, leading to the following definition:

Definition 5.1. Let $c = (s, h)$ be a contour of f. Define the *upstart* region of s at h to be: $R_s^+(h) = \{x \in \mathcal{R} - c\colon\ \forall$ paths P in $\mathcal{R}$ from c to x, $P \cap R^+(c) \neq \emptyset\}$ and the *downstart* region of s at h to be: $R_s^-(h) = \{x \in \mathcal{R} - c\colon\ \forall$ paths P in $\mathcal{R}$ from c to x, $P \cap R^-(c) \neq \emptyset\}$.

Contours not belonging to supernodes thus each partition the domain into two regions; contours belonging to supernodes divide the domain into exactly one upstart or downstart region for each incident superarc. For example, in Fig. 2, contour c_1 has an upstart region inside and a downstart region outside. At saddle D, however, there are several upstart regions, one for each ascending superarc. We can refer to these regions by their superarcs, such as the upstart region at D for arc CD.

Upstart and *downstart functions* are functions computed over upstart or downstart regions. Examples include the area of the region or the length of the contour line that bounds it. Note that the first is dependent on the sweep direction, so is different for upstart and downstart regions, whereas the second is independent of sweep direction, so the upstart and downstart functions for contour length are identical.

5.1. Local geometric measures in the contour tree

A local geometric measure is an upstart or downstart function of a given contour. For simplicial meshes, Bajaj, Pascucci and Schikore [1] showed that measures such as contour length, area, volume, etc. are piecewise polynomial since they are piecewise polynomial in each cell. Moreover, these polynomials can be tracked efficiently during an isovalue sweep by storing the coefficients of the polynomial and changing the coefficients as we sweep past each vertex in the mesh. We modify this approach by using locally isovalued sweeps which follow the edges of the contour tree.

For a single isovalued sweep in a consistent direction (i.e. either up or down), updating local geometric measures is straightforward. For a sweep inwards through the arcs of the contour tree, however, we continually reverse the direction of the sweep. If we are measuring the boundary size of the sweeping contour, this is trivial, but for other measures, it is more complex.

At a saddle point, we must be able to *combine* upstart functions as we sweep a set of contours past a vertex. In Fig. 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 several local geometric measures.

Contour length (2D): 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.

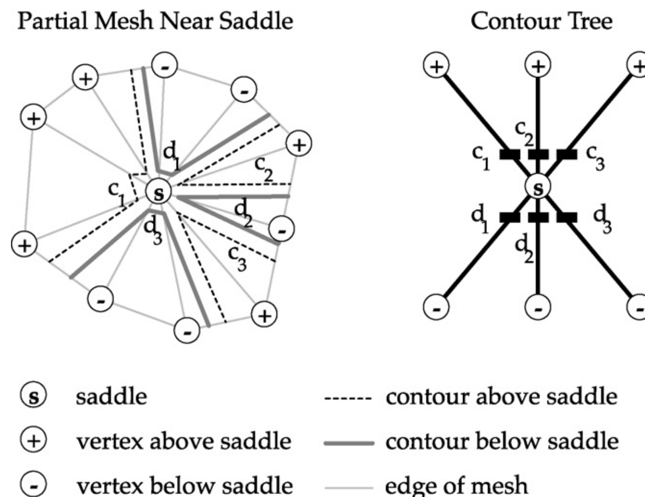

Fig. 4. Contours sweeping past a saddle point.

Contour surface area (3D): This is analogous to contour length, and follows the same rules.

Cross-sectional area (2D): 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.

Surface area (2D): If we consider a function over a two-dimensional domain as a terrain, it is clear that there are two possible measures of area for a region bounded by a given contour: the area on the ground plane (cross-sectional area), or the surface area on the manifold embedded in three dimensions. In the latter case, the surface area is the integral of the contour length with respect to the isovalue and follows the same rule as cross-sectional area.

Volume (3D): In three dimensions, the analogue to area is the volume of the upstart region. As with area, we can have either the cross-sectional volume or the hypersurface volume (the integral of surface area with respect to isovalue). This also follows the same rule as area in 2-D: inversion is achieved by subtracting from the volume for the entire domain, while combination requires summing the functions for the regions being combined.

Volume (2D): In two dimensions, we can also compute the volume between the surface of the function and the isovalue cutting plane over the upstart or downstart region. For a given isovalue h, the volume between f and h over an upstart region R is:

$$\int_A \big(f(x) - h\big)\, dh = \int_R f(x)\, dh - Ah \tag{1}$$

where A is the area of region R.

If f is defined over a simplicial mesh, then the first term of this is a piecewise-polynomial function and the second term is the cross-sectional area multiplied by the isovalue of interest. Over a downstart region, the terms are reversed. Formally, the reversal rule is that the value is multiplied by -1, and the combination rule uses addition. In practice, it is easier to track the two terms separately, computing the volume from them at need.

Hypervolume (3D): In three dimensions, the analogue of volume in two dimensions is *hypervolume*. Instead of treating the contours as individual surfaces, we view them as cross-sections of a hypersurface, and compute the hypervolume between that hypersurface and a hyperplane of the form $f = h$. This uses an integral of the same form just shown for volume under a terrain, and is computed in exactly the same way.

Since these computations involved local updates during contour sweeps, they are easy to incorporate in the merge phase of the contour tree construction algorithm [5]. However, since these computations use the fully augmented contour tree, they cannot be computed using the optimized algorithms of [29] and [7], which discard connectivity regular points before the merge phase.

5.2. Approximate local geometric measures

Local geometric measures are easy to compute for simplicial meshes. For meshes with bilinear or trilinear interpolants, contours are hyperbolic in nature [27], and the integrals required for exact evaluation involve exponential terms. Moreover, the bounds for evaluation depend on detailed analysis of the particular interpolant. And, even where these integrals are soluble, they do not reduce to polynomials, and the sweep strategy identified by Bajaj, Pascucci and Schikore BPS97 cannot be applied.

Since exact evaluation of these integrals is a significant task in its own right, we leave consideration of them for future work, and instead compute approximate local geometric measures, including several measures that are common in digital imaging: we note in passing that it is possible to compute contour trees using digital imaging connectivity [26].

Contour area (2D)/volume (3D): As we saw in Section 5.1, these measures are analogous, and measure the size of the region in the upstart or downstart region. For regular datasets, we approximate this by counting N_R, the number of sample vertices in the region, an approximation backed by the definition of area of an irregular region as the Riemann sum of the areas of regular patches inside the region.

$$N_R = \sum_{x \in R} 1 \approx \int_R 1\, dh = Area(R) \tag{2}$$

Volume (2D)/hypervolume (3D): Again, these measures are analogous, and measure the size of the region between the function and a cutting plane. For regular data, we sum the sample values inside the region to approximate the correct value, again noting the similarity to the Riemann sum definition of the integral:

$$Size_R = \int_R \big(f(x) - h\big)\, dh \approx \left(\sum_{x \in R} f(x)\right) - N_R h \tag{3}$$

Mean, standard deviation and RMS error: In addition to approximations of geometric measures, we can also compute statistical properties of the sample values inside the upstart or downstart region. We can for example compute μ_R, the

mean value, σ_R, the standard deviation, or RMS_R, the root mean squared error for pruning the branch of the contour tree corresponding to the upstart or downstart region at h:

$$\mu_R = \frac{\sum f(x)}{\sum 1} \tag{4}$$

$$\sigma_R = \sqrt{\frac{\sum f^2(x) - 2\mu_R \sum f(x) + \mu_R^2}{N_R}} \tag{5}$$

$$RMS_R = \sqrt{\frac{\sum f^2(x) - 2h \sum f(x) + \sum h^2}{N_R}} \tag{6}$$

where all summations are over all $x \in R$.

6. Contour tree simplification

Although the contour tree represents the component connectivity of the entire scalar field, it is vulnerable to noise, which adds small scale topology, especially in empirically-acquired data. In order for the contour tree to be useful as a user interface, it is desirable to simplify both the contour tree and the underlying topology in a meaningful fashion.

The basic mechanism for contour tree simplification has also been reported by Takahashi et al. [36,39] and by Pascucci and Cole-McLaughlin [30]. These authors assume that all saddles are simple (i.e. degree 3) and enforce this assumption by symbolic perturbation of the saddles. Under this assumption, simplification was performed by choosing the Y-shaped structure adjacent to a leaf of the tree and replacing it with a straight edge between the two non-leaf vertices of the Y. We present the mechanism in a more general form that permits multiple saddles, does not require symbolic perturbation, and allows simplification to be guided by measures other than *persistence*.

We assume that the contour tree is stored as a set of edges with path seeds attached, although branch decomposition [30] could also be employed. We also assume that each edge has a simplification value (weight) that indicates the edge's importance. We use the local geometric measures of Section 5 for this purpose, but any other basis for computing weights could be used.

We simplify the contour tree principally by applying two primitive operations: *leaf pruning* and *vertex reduction*. Leaf pruning removes topology from the field (and the tree) by selecting a leaf node of low importance and removing it, while vertex reduction eliminates connectivity regular points from the contour tree, leaving the topology unchanged.

6.1. Leaf pruning

Our principal contour tree simplification operation is *leaf pruning*, which removes a leaf edge of the tree. This reduces the tree size by one vertex and one edge, deleting the leaf vertex l and retaining the interior (saddle) vertex s. In Fig. 5, vertex 80 on the left is pruned to produce the tree in the middle. This implicitly simplifies the topology of the underlying scalar field as well. Recall that each point on the pruned leaf edge corresponds to a single contour of the scalar field. Leaf pruning therefore discards all contours corresponding to the pruned edge. Let A refer to the region that is the union of all these contours. Then pruning the leaf edge is equivalent to any of the following topological operations:

1. removing A from the domain of the scalar field,
2. contracting A to the saddle s,
3. setting $f(a) = f(s)$ for all $a \in A$, or
4. setting $f(v) = f(s)$ for all vertices $v \in A$ in a simplicial mesh with barycentric interpolation.

Although removal and contraction of A are easy to define topologically, they cause regular meshes to become irregular either due to holes in the domain (removal) or spatial distortion (contraction). The other two operations flatten the ex-

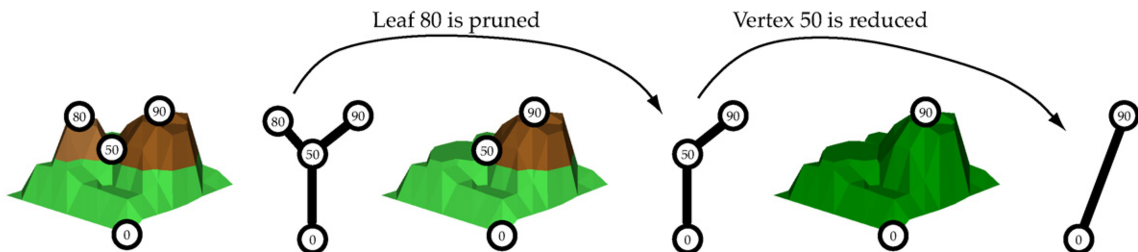

Fig. 5. Leaf pruning levels extrema; Vertex reduction leaves scalar field unchanged.

tremum corresponding to the leaf, resulting in a flat spot in the data, as shown in Fig. 5. The last of the four operations, while restricted to simplicial meshes, is trivial to implement, although the boundary contour of A will deform slightly as a result.

We note that these last two operations cause the function to become non-Morse. However, as Pascucci and Cole-McLaughlin [30] have shown, this can be remedied by an additional perturbation step, which raises the saddle by a symbolic epsilon and tilts the flat spot to regain Morseness.

As a side effect, leaf pruning may cause saddles to become connectivity regular points at which no topological change occurs. These points can then be removed from the contour tree by a second simplification operation: *vertex reduction*, which does not modify the field.

6.2. Vertex reduction

Vertex reduction deletes a connectivity regular point in the contour tree: a vertex with one neighbor above and one below, replacing the two incident edges with a single edges. In Fig. 5, vertex 50 is reduced from the tree in the middle to produce the tree on the right. Note that this does not alter the field, since any point in the unsimplified tree is still present in the simplified tree. Because of this, we prefer vertex reductions to leaf prunes when they are available.

Pruning a leaf connected to a saddle of degree three will normally reduce the saddle to a connectivity regular point which can then be reduced, as shown in Fig. 5. The sole exception to this is that if the domain of the scalar field is not homeomorphic to a disk, in which case a vertex may have two upwards edges and no downwards edge, or vice versa. These meshes are not commonly seen in practice, but even on such meshes, leaf pruning is still possible at the vertex, so this does not hamper our simplification method.

To maximize the number of vertex reductions, we protect the last upwards or downwards edge at saddle from leaf prunes. For example, in Fig. 5, pruning vertex 0 would needlessly flatten the lowest region upwards to the saddle value of 50, and vertex 50 would never be reduced. If we prevent 0 from being pruned, however, we will be able to reduce 50 at a later date, as shown.

In the worst case, it might seem that all leaves could be protected in this way, causing the algorithm to stall. We show that this cannot happen with a counting argument based on the fact that any tree must have at least one more leaf edge than it has interior points (saddles). Since we only protect the last edge upwards or downwards, it follows that at most one upwards and one downwards edge can be protected at a given saddle. Because each of these is the last edge in its direction, if a saddle has protected edges in both directions, it has degree two and may be reduced, which we have assumed to have been done already. Thus, each saddle may protect at most one edge. Since there are more leaf edges than saddles, it follows that there is always a leaf available for pruning.

6.3. Simplification algorithm

Implementing simplification can be done efficiently by maintaining a priority queue to store 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.

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. We note that *up(d)* is stored at the *top* supernode of the edge, and represents the cost of pruning the edge upwards – i.e. the local geometric measure for the downstart region of the top supernode. Similarly, $down(d)$ is stored at the *bottom* supernode of the edge and represents the cost of pruning the edge downwards.

Since vertex reduction merges two edges, the downstart region at the top supernode of the new edge d is the same as the downstart region at the top supernode of the e_1, the upper of the two edges that are merged. It follows that $up(d) = up(e_1)$, while $down(d) = down(e_2)$, the lower of the two edges merged. Having determined the pruning costs for d, 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)queuing 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 find path seeds. Vertex reduction builds *branch decompositions* – monotone paths corresponding to the simplified edges, while leaf pruning discards entire monotone paths. Thus, any edge in a simplified contour tree corresponds to a monotone path through the original contour tree. To generate a contour at a given isovalue on a simplified edge, we perform a 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.

Finally, $up(e)$ and $down(e)$ need only be computed at leaves. As a leaf is pruned or a vertex reduced, new values can be computed using information from the old nodes and edges.

6.4. Simplification and perturbation

In Section 2, we stated that we would apply perturb the data using simulation of simplicity to ensure that the function f was a Morse function. For user interface purposes, however, we wish to make sure that the simplified tree represented removes this perturbation.

To achieve this in practice requires simplifying the contour tree using leaf pruning, vertex reduction and *edge reduction*. In the same way that vertex reduction deletes a vertex and combines the two incident edges, edge reduction deletes an edge and combines the two incident vertices. While it is possible to apply edge reduction to the tree generally, it generates large flat spots in the data, which is why we avoid it for simplification. Where, however, a flat spot exists in the data in the first place and the contour tree shows an edge of zero height, it is entirely appropriate to reduce the edge.

7. The flexible isosurface interface

Having dealt with the necessary algorithmic underpinnings, we now turn to the question of how we can use the contour tree to provide finer control over contour extraction and display. We will use the path seeds in the contour tree to extract individual contours, not all of which need have the same isovalue, and define a *flexible isosurface*:

Definition 7.1. A *flexible isosurface* is a set $\{c_1, \ldots, c_\tau\}$, of contours $c_i = (s_i, h_i)$ of f.

This definition generalizes isosurfaces, *object-oriented visualization* [34] and *largest contour segmentation* [23], while also permitting manipulations such as that described in Section 1. We permit at most one contour on each superarc to belong to the flexible isosurface, to prevent confusion in the use of the interface.

Fig. 6 shows the flexible isosurface interface, which consists of panels displaying the contours, the contour tree, and the level of simplification, as well as miscellaneous controls. Although superficially similar to the Contour Spectrum [1], there are some important differences. The most fundamental difference is that instead of a passive display of a single level set, the main panel in this interface allows direct user manipulation of the visible objects. We employ the conventional *select and operate* metaphor for modeless user interfaces, in which the user uses the mouse to select an object, then applies a desired operation to that object. We also support *undoing* and *redoing* operations.

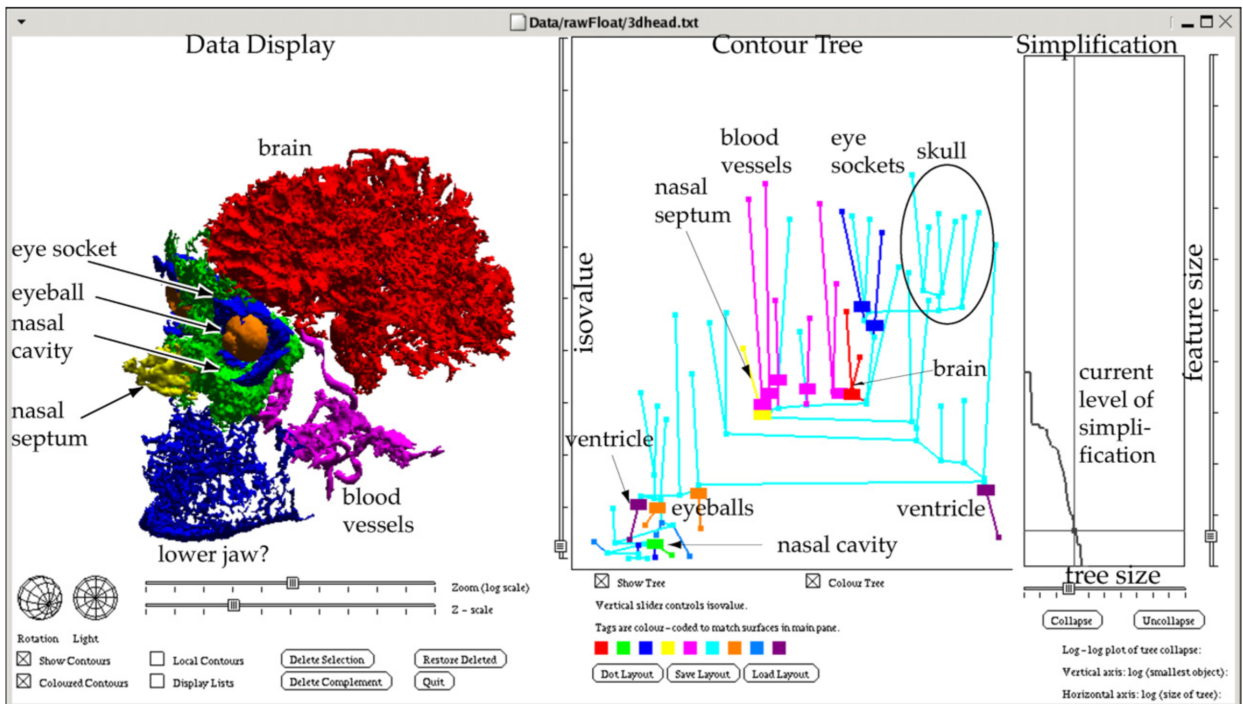

Fig. 6. The flexible isosurface interface.

In the contour display panel, we render the current flexible isosurface: we also render it in the contour tree panel. In this panel, the contour tree is laid out either manually or with the *dot* package [12], with the vertical dimension representing isovalue. Each contour $c = (s, h)$ can therefore be represented by a rectangular *tag* on the superarc s at isovalue h. Each tag is assigned the same color as the corresponding contour in the contour display to further aid the user in relating the topology (the contour tree) with the geometry (the contours).

Since we have assumed that objects are single contours, we support the following operations: *selecting* a contour, *deselecting* a selected contour, *adding* a contour to a flexible isosurface, *deleting* a contour, *isolating* a contour by deleting all other contours, *applying* attributes to a contour, *evolving* the isovalue of a contour, and *initializing* a flexible isosurface. Two additional operations can be applied to the contour tree as a whole: *simplification* and *unsimplification*.

Selection A contour may be *selected* visually with the mouse: the object under the mouse pointer can be identified with any standard picking algorithm, provided that the objects have some unique identifier. Since we use path seeds to extract contours, we use the ID of the superarc to which each contour belongs as the unique identifier. If we have simplified the contour tree, we use the ID of the simplified arc – as noted above, we will use the branch decomposition to descend to an unsimplified superarc when we need to generate a path seed.

We also permit contours to be selected in the contour tree panel: if the mouse pointer is over a colored tag, the corresponding contour is selected. In either case, we *highlight* the contour by changing its color. Although multiple selection is feasible, the semantics of contour evolution become unclear, so we disallow it.

Deselection A contour is *deselected* when another contour is chosen, or when the user depresses the mouse button over a region of the contour display panel or the contour tree panel which does *not* select a contour.

Adding Since we can only apply an operation to a contour we can see, it is not possible to *add* a contour to the flexible isosurface in the contour display panel. Instead, if the mouse pointer is over a superarc s in the contour tree at isovalue h but not over a colored tag, we add $c = (s, h)$ to the flexible isosurface and select c immediately.

Deletion and isolation A contour may be *deleted* by selecting it, then using one of the buttons beneath the contour display panel to delete it from the current flexible isosurface. A second button allows the contour to be *isolated* by deleting all other contours in the current flexible isosurface.

Applying attributes In general, we can store arbitrary information for each contour in the flexible isosurface. This attribute information can include color, texture, display list IDs for accelerated rendering, text, or any other information desired. At present, we support display list IDs and color information: the latter can be changed with the small color palette beneath the contour tree pane.

Evolution One of the commonest isosurface operations is to vary the isovalue of an entire level set. *Evolution* varies the isovalue of a single contour, either by selecting the contour and manipulating its isovalue with the vertical slider beside the contour tree panel, or by selecting a contour in the contour tree pane and dragging the mouse with the button depressed, in which case the y-coordinate of the mouse pointer is used instead of the slider. We describe the semantics of contour evolution below, as they can be fairly complex.

Initialization A flexible isosurface can be initialized to a conventional isosurface (level set) or to a largest contour segmentation, controlled by a check box below the contour display pane. If no contour is currently selected, we assume that the user may wish to re-initialize the flexible isosurface by changing the isovalue of the vertical slider next to the contour tree panel. Moving this slider vertically thus generates conventional isosurfaces if the check box has not been checked, and a parameterized largest contour segmentation if it has, mapping the isovalue proportionally onto each of the upward leaves in the tree.

Simplification and unsimplification The rightmost panel is used to control the level of simplification of the contour tree, and plots the size of the contour tree horizontally against the measure of the largest leaf pruned vertically, in a log–log plot. A group of radio buttons beneath the panel sets which local geometric measure to apply, while sliders are provided for major changes in simplification level and buttons for minor changes. This simplification is performed on the fly using the simplification algorithm discussed above.

7.1. Evolving a contour

With conventional isosurfaces, changing the isovalue and observing the corresponding changes in the level set is one of the commonest operations, since it allows the user to gain insight into the overall structure of the data. Correspondingly, we permit *evolution* of a single contour:

Definition 7.2. A contour c at isovalue h evolves into the set of all contours $c_1, \ldots, c_m$ at isovalue h' for which at least one f-monotone path P_i exists in $\mathcal{R}$ from c to each c_i.

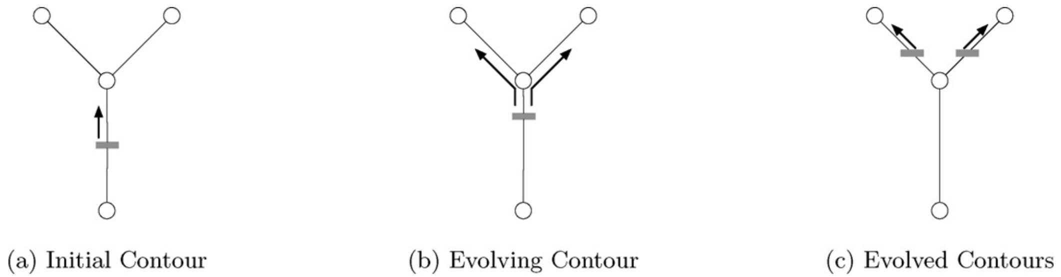

Fig. 7. Simple example of an evolving contour. (a) As the contour evolves upwards, we generate contours on the same superarc at progressively higher isovalues. (b) As the contour evolves past a saddle, it splits into one contour for each of the upwards superarcs at the saddle. (c) Once past the saddle, each contour continues to evolve upwards on its own superarc.

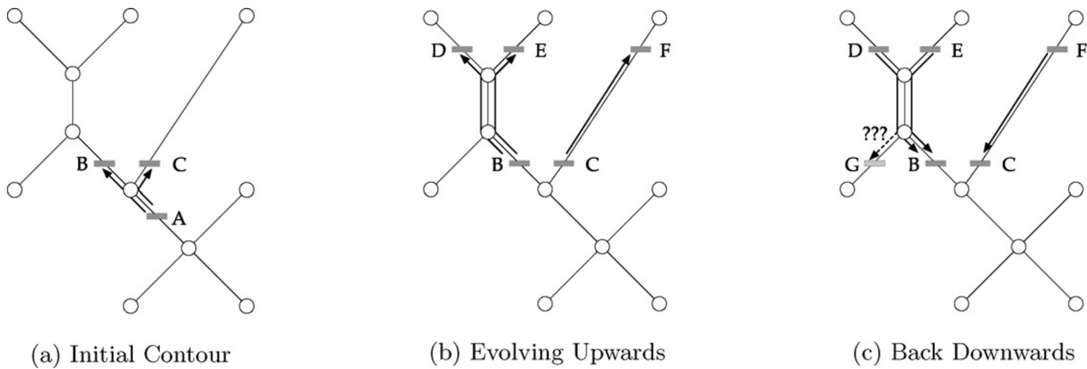

Fig. 8. Continuous contour evolution: Upwards, then downwards. (a) A evolves upwards into contours B and C. (b) B and C evolve into D, E and F. (c) When the parameter is returned to a smaller isovalue of B and C, should G be shown?

This definition emphasizes the expectation of the user that they are seeing the results of increasing (or decreasing) the isovalue *of a specified contour*. Since we use the contour tree as our index, we must convert this definition into one in terms of the contour tree, an easy task given the correspondence between contours and the contour tree:

Lemma 7.3. *Definition* 7.2 *is equivalent to the following formulation*: *A contour c at isovalue h evolves into the set of contours $c_1, \dots, c_m$ at isovalue h' such that for each c_i there exists a monotone path Q_i from c to c_i in the contour tree.*

Proof. For each monotone path in either $\mathcal{R}$ or the contour tree, Property 1 guarantees that a corresponding monotone path exists in the other. □

This result allows us to discuss contour evolution solely in terms of the contour tree, while still remaining true to the user's expectations. We show a simple example of contour evolution in Fig. 7. In this case, a single contour is chosen in a small contour tree. As the isovalue is increased using the isovalue slider, the contour tag slides up the superarc of the contour tree from Fig. 7(a) to Fig. 7(b). As the contour evolves past the isovalue of the join, the contour separates into two distinct contours, one for each branch of the contour tree at the join.

Semantically, the basic meaning of this evolution is clear: the user expects to see a smooth evolution of contours as the parameter is changed. As the parameter is changed repeatedly, however, several difficulties arise.

Continuous and reversible contour evolution For the user, contour evolution should be *continuous* by providing visual feedback as the isovalue is changed, and *reversible* to any previous stage in the evolution. This emphasizes the assumption that the evolving contours represent the same object at different isovalues. Consider Fig. 8, in which contour A evolves upwards to B and C, then to D, E, and F, then back downwards. If the evolution were a sequence of progressive evolutions, G would be included in the flexible isosurface. This is clearly not reversible, so we treat a sequence of evolutions as individual evolutions from a common initial contour.

Giving precedence to evolving contours Fig. 9 shows a contour, A evolving past a contour B that is already present in the contour tree. As the isovalue is increased, A evolves into A_1 and A_2, then into A_3, A_4 and A_5. If, as is often the case, contour A_5 is entirely inside contour B in the contour display pane, the user will be unable to see the effect of the evolution. We therefore assume that A_5 is more important to the user than B, and remove B from the flexible isosurface.

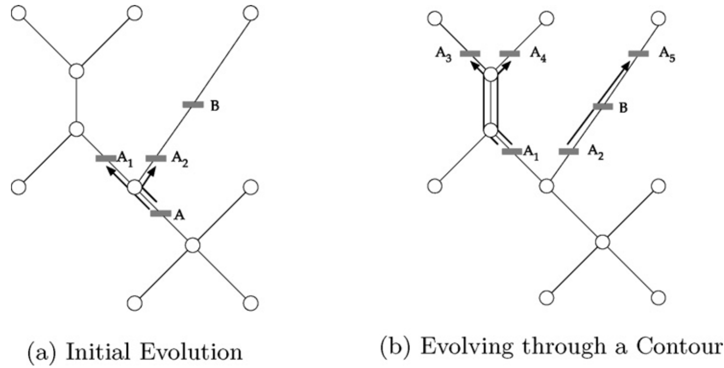

Fig. 9. Suppressing contours during evolution. As A evolves past B, B will normally occlude either A_2 or A_5 as A sweeps past B. We assume that the user is more interested in the evolution of A through A_2 and B to A_5 than in B itself, and suppress B so that the evolution of A is clear.

Table 1
Path seed isosurface extraction: Large contour trees tend to have shorter paths, and vice versa. Moreover, the path lengths are much smaller than the output cost, k.

Data set	Size (n) (samples)	Tree size (t) (supernodes)	Longest path (edges)	Isosurface size (k) (triangles)
f368	30,345	915	4–6	10K–11K
marlobb	68,921	912	1–23	10K–35K
fuel	262,144	227	2–3	3K–11K
hipiph	262,144	1360	5–29	2K–22K
neghip	262,144	2063	7–10	10K–30K
lobster	489,600	17,867	4	19K–96K
teddybear	1,015,808	245,588	4–5	19K–250K
3dhead	7,143,424	2,231,900	2–5	85K–500K
3dknee	8,323,072	2,751,506	1–6	325K–1634K

In order to make this reversible, we do not actually delete B until the sequence of evolutions is complete: thus, if the user evolves the contours back down, B will reappear.

8. Results

8.1. Path seed performance

In Table 1, we show some examples of path seed length and the corresponding size of isosurfaces. Broadly speaking, for a data set of size $n \times n \times n$, we might expect linear features such as seed paths to be of size $O(n)$, but areal features such as contours to be of size $O(n^2)$. To generate this table, we tested a range of isovalues to determine the longest path length needed to generate the contours, along with the corresponding total isosurface size. As we see, broadly speaking, the path length is in line with our expectations, and is certainly small enough that extraction time will be dominated by the size of the isosurfaces generated.

8.2. Comparison of local geometric measures

In Fig. 10, 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* [12] with no manual intervention. As we will see in Section 7, the superarcs in these simplified contour trees correspond to individual features in the data set shown in Fig. 6.

In the left-hand image, height (persistence) was used as the geometric measure. All of the edges shown are tall as a result, but on inspection, many of these edges were 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 have limited persistence. Moreover, on further inspection, the large number of objects with large persistence tended to be fragments of larger objects, particularly the skull.

In comparison, the middle image shows the results of using approximate volume. Not only does this focus attention on a few objects of relatively large spatial extent, but there were fewer objects of large volume than of large persistence. Important objects such as the eyeballs are retained because they have relatively large regions despite having low persistence. However, we note that there are also a large number of low-persistence 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.

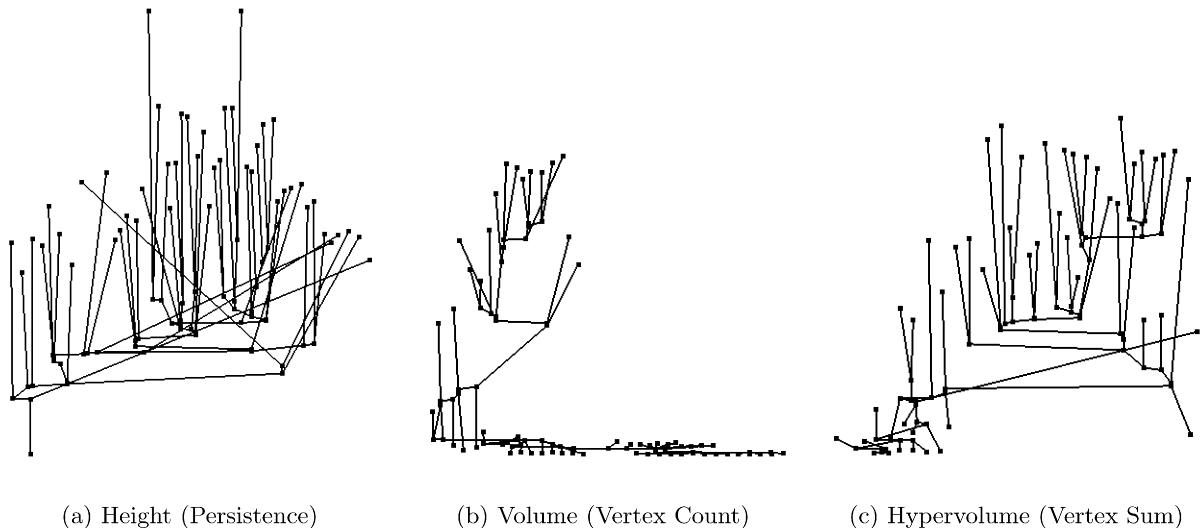

Fig. 10. Comparison of simplification using three local geometric measures. In each case, the UNC Head data set has been simplified to 92 edges using the specified measure. Trees were laid out using the *dot* tool [12], with no manual adjustment.

Finally, the right-hand image shows the results of using approximate hypervolume. In this case, we observed a very rapid dropoff of importance of features, with only 100 or so regions having significant hypervolumes. This measure preserves large low-persistence 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.

We examined a broad range of datasets, from medical datasets to synthetic and analytical functions, and found that in most circumstances, the hypervolume measure was better for data exploration than either height or volume, since it balances representation of high-persistence 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.

8.3. Simplification results

We used a variety of data sets to test our simplification 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 2 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.

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 2, this is much smaller than the input size. For noisy experimentally acquired data, such as the ones shown in the lower half of Table 2, the size of the contour tree is roughly proportional to the input size.

As Table 2 shows, we also computed the time taken to simplify the contour tree to a single edge, as this is an upper bound on the time taken to simplify the contour tree to any desired size. These times were obtained using a 3 GHz Pentium 4 with 2 GB RAM, and the *hypervolume* measure, and are shown in comparison to the time taken to compute the contour tree itselfs. As we see from the table, simplification typically costs less than one percent of the time of constructing the original contour tree, plus the additional cost of pre-computing local geometric measures during contour tree construction.

9. Conclusions and future work

We have shown how to use the contour tree to provide an innovative visual interface for exploring scalar data fields, with simplification of the contour tree guided by local geometric measures of feature importance, and individual contours extracted using path seeds.

There are a number of directions for extensions and future work. We believe that simplified contour trees can provide seeds for level set methods or for boundary propagation methods such as T-snakes [16]. Interfaces similar to the flexible isosurface interface have now been extended to interval volumes [36] and volume rendering [43], but further work is possible here.

Further work is required on the best geometric measures for tree simplification, on the use of local geometric measures for automatic feature recognition, on improved methods for displaying the contour tree visually, and on methods for simultaneous simplification of the contour tree and the contours extracted.

Table 2
Data sets, unsimplified contour tree sizes, and contour tree construction time (CT) and simplification time (ST) in seconds.

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

Finally, the continuation method used to extract single contours is inherently serial in nature, and we intend to investigate methods for parallel or distributed methods for single contour extraction.

Acknowledgements

We gratefully acknowledge Alan Ableson of Queen's University and the contributors to volvis.org for data, and the National Science and Engineering Research Council of Canada (NSERC), the Institute for Robotics and Intelligent Systems (IRIS), and the US National Science Foundation (NSF) for post-graduate fellowships and research grants.

References

[1] C.L. Bajaj, V. Pascucci, D.R. Schikore, The contour spectrum, in: Proceedings of Visualization, 1997, pp. 167–173.
[2] R.L. Boyell, H. Ruston, Hybrid techniques for real-time radar simulation, in: Proceedings of the 1963 Fall Joint Computer Conference, IEEE, 1963, pp. 445–458.
[3] P.-T. Bremer, H. Edelsbrunner, B. Hamann, V. Pascucci, A topological hierarchy for functions on triangulated surfaces, IEEE Transactions on Visualization and Computer Graphics 10 (2004) 385–396.
[4] H. Carr, J. Snoeyink, Path seeds and flexible isosurfaces: Using topology for exploratory visualization, in: Proceedings of Eurographics Visualization Symposium, 2003, pp. 49–58, 285.
[5] H. Carr, J. Snoeyink, U. Axen, Computing contour trees in all dimensions, Computational Geometry: Theory and Applications 24 (2) (2003) 75–94.
[6] H. Carr, J. Snoeyink, M. van de Panne, Simplifying flexible isosurfaces with local geometric measures, in: Proceedings of Visualization, 2004, pp. 497–504.
[7] Y.-J. Chiang, T. Lenz, X. Lu, G. Rote, Simple and optimal output-sensitive construction of contour trees using monotone paths, Computational Geometry: Theory and Applications 30 (2005) 165–195.
[8] Y.-J. Chiang, X. Lu, Progressive simplification of tetrahedral meshes preserving all isosurface topologies, Computer Graphics Forum 22 (3) (2003) 493–504.
[9] H. Edelsbrunner, J. Harer, A. Zomorodian, Hierarchical Morse–Smale complexes for piecewise linear 2-manifolds, Discrete and Computational Geometry 30 (2003) 87–107.
[10] H. Edelsbrunner, E.P. Mücke, Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms, ACM Transactions on Graphics 9 (1) (1990) 66–104.
[11] H. Freeman, S. Morse, On searching a contour map for a given terrain elevation profile, Journal of the Franklin Institute 284 (1) (1967) 1–25.
[12] E.R. Gansner, E. Koutsofios, S.C. North, K.-P. Vo, A technique for drawing directed graphs, IEEE Transactions on Software Engineering 19 (1993) 214–230.
[13] C.-H. Hung, C.-k. Yang, A simple and novel seed-set finding approach for isosurface extraction, in: Proceedings of Eurographics – IEEE Symposium on Visualization, 2005, pp. 125–132.
[14] T. Itoh, K. Koyamada, Automatic isosurface propagation using an extrema graph and sorted boundary cell lists, IEEE Transactions on Visualization and Computer Graphics 1 (4) (1995) 319–327.
[15] T. Itoh, Y. Yamaguchi, K. Koyamada, Fast isosurface generation using the volume thinning algorithm, IEEE Transactions on Visualization and Computer Graphics 7 (1) (2001) 32–46.
[16] M. Kass, A. Witkin, D. Terzopoulos, Snakes: Active contour models, in: Proceedings of the 1st International Conference on Computer Vision, IEEE, 1987, pp. 259–268.
[17] L. Kettner, J. Rossignac, J. Snoeyink, The safari interface for visualizing time-dependent volume data using iso-surfaces and contour spectra, Computational Geometry: Theory and Applications 25 (1–2) (2001) 97–116.
[18] G. Kindlmann, R. Whitaker, T. Tasdizen, T. Möller, Curvature-based transfer functions for direct volume rendering: Methods and applications, in: Proceedings of Visualization, 2003, pp. 513–520.
[19] J. Kniss, G. Kindlmann, C.D. Hansen, Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets, in: Proceedings of Visualization, 2001, pp. 255–262, 562.
[20] I.S. Kweon, T. Kanade, Extracting topographic terrain features from elevation maps, CVGIP: Image Understanding 59 (1994) 171–182.

[21] M. Levoy, Volume rendering: Display of surfaces from volume data, IEEE Computer Graphics and Applications 8 (3) (1988) 29–37.
[22] W.E. Lorenson, H.E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, Computer Graphics 21 (4) (1987) 163–169.
[23] E.M.M. Manders, R. Hoebe, J. Strackee, A. Vossepoel, J. Aten, Largest contour segmentation: A tool for the localization of spots in confocal images, Cytometry 23 (1996) 15–21.
[24] Y. Matsumoto, An Introduction to Morse Theory, AMS, 2002.
[25] J. Milnor, Morse Theory, Princeton University Press, Princeton, NJ, 1963.
[26] S. Mizuta, T. Matsuda, Description of the topological structure of digital images by region-based contour tree and its application, Technical report, Institute of Electronics, Information and Communication Engineers, 2004.
[27] G.M. Nielson, B. Hamann, The asymptotic decider: Resolving the ambiguity in marching cubes, in: Proceedings of Visualization, IEEE, 1991, pp. 83–91.
[28] V. Pascucci, On the topology of the level sets of a scalar field, in: Abstracts of the 13th Canadian Conference on Computational Geometry, 2001, pp. 141–144.
[29] V. Pascucci, K. Cole-McLaughlin, Efficient computation of the topology of level sets, in: Proceedings of Visualization, 2002, pp. 187–194.
[30] V. Pascucci, K. Cole-McLaughlin, G. Scorzell, Multi-resolution computation and presentation of contour trees. in: Proceedings of the IASTED Conference on Visualization, Imaging and Image Processing (VIIP 2004), 2004, pp. 452–290.
[31] V. Pekar, R. Wiemker, D. Hempel, Fast detection of meaningful isosurfaces for volume data visualization, in: Proceedings of Visualization, 2001, pp. 223–230.
[32] G. Reeb, 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 (1946) 847–849.
[33] Y. Shinagawa, T.L. Kunii, Y.L. Kergosien, Surface coding based on Morse theory, IEEE Computer Graphics and Applications 11 (1991) 66–78.
[34] D. Silver, Object-oriented visualization, IEEE Computer Graphics and Applications 15 (3) (1995) 55–62.
[35] J.K. Sircar, J.A. Cebrian, Application of image processing techniques to the automated labelling of raster digitized contour maps, in: Proceedings of the 2nd International ACM Symposium on Spatial Data Handling, 1986, pp. 171–184.
[36] S. Takahashi, I. Fujishiro, Y. Takeshima, Interval volume decomposer: A topological approach to volume traversal, in: R.F. Erbacher, K. Börner, M. Gröhn, J.C. Roberts, (Eds.), Visualization and Data Analysis 2005 (Proceedings of the SPIE), 2005.
[37] S. Takahashi, T. Ikeda, Y. Shinagawa, T.L. Kunii, M. Ueda, Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data, Computer Graphics Forum 14 (3) (1995) C-181–C-192.
[38] S. Takahashi, G.M. Nielson, Y. Takeshima, I. Fujishiro, Topological volume skeletonization using adaptive tetrahedralization, in: Geometric Modelling and Processing, 2004.
[39] S. Takahashi, Y. Takeshima, I. Fujishiro, Topological volume skeletonization and its application to transfer function design, Graphical Models 66 (1) (2004) 24–49.
[40] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, Journal of the ACM 22 (1975) 215–225.
[41] S. Tenginakai, J. Lee, R. Machiraju, Salient iso-surface detection with model-independent statistical signatures, in: Proceedings of Visualization, 2001, pp. 231–238.
[42] M. Van Kreveld, R. Van Oostrum, C.L. Bajaj, V. Pascucci, D.R. Schikore, Contour trees and small seed sets for isosurface traversal, in: Proceedings of the 13th ACM Symposium on Computational Geometry, 1997, pp. 212–220.
[43] G. Weber, S. Dillard, H. Carr, V. Pascucci, B. Hamann, Topology-controlled volume rendering, IEEE Transactions on Visualization and Computer Graphics 13 (2) (2007) 330–341.
[44] G. Wyvill, C. McPheeters, B. Wyvill, Data structure for soft objects, Visual Computer 2 (1986) 227–234.
[45] X. Zhang, C.L. Bajaj, N. Baker, Application of new multi-resolution methods for the comparison of biomolecular electrostatic properties in the absence of global structural similarity, SIAM Journal on Multiscale Modelling & Simulation (2006) 1196–1213.