Visualizing Contour Trees within Histograms

Kraus, Martin

Published in:
Computer Graphics and Imaging

Publication date:
2010

Document Version
Early version, also known as pre-print

VISUALIZING CONTOUR TREES WITHIN HISTOGRAMS

Martin Kraus
CVMT Laboratory
Aalborg University
Niels Jernes Vej 14,
DK-9220 Aalborg East, Denmark
email: martin@imi.aau.dk

ABSTRACT
Many of the topological features of the isosurfaces of a scalar volume field can be compactly represented by its contour tree. Unfortunately, the contour trees of most real-world volume data sets are too complex to be visualized by dot-and-line diagrams. Therefore, we propose a new visualization that is suitable for large contour trees and efficiently conveys the topological structure of the most important isosurface components. This visualization is integrated into a histogram of the volume data; thus, it offers strictly more information than a traditional histogram. We present algorithms to automatically compute the graph layout and to calculate appropriate approximations of the contour tree and the surface area of the relevant isosurface components. The benefits of this new visualization are demonstrated with the help of several publicly available volume data sets.

KEY WORDS
Information visualization; graph visualization; volume visualization; contour tree; topology; histogram

1 Introduction

The visualization of isosurfaces is of particular interest in scalar volume visualization as it is a well established and accepted technique in many sciences. However, it is not possible to visualize many isosurfaces in a single, comprehensible image. Fortunately, it is often sufficient to extract specific information about the structure or particular features of the scalar field. Some of the most important topological features of the set of all possible isosurfaces of a scalar volume field can be compactly represented by the contour tree. In fact, the contour tree has been successfully employed for a variety of tasks in volume visualization as discussed in Section 2.

Although the contour tree has been proven useful, the visualization of contour trees of real-world data sets is a challenging problem. While the contour tree is usually a compact representation of the scalar field in comparison to the size of the corresponding volume data set, it is in most cases still far too large to be visualized by dot-and-line diagrams; in particular if the volume data is noisy. Moreover, even for strongly simplified contour trees, it appears to be rather difficult for non-experts to match the edges of traditional graph drawings to connected components of isosurfaces.

Therefore, we propose a new visualization of the contour tree that resembles common visual abstractions of the volume data, namely histograms and isosurface statistics [1, 2]. It is straightforward to generalize traditional histograms of volume data to stacked bar charts representing the decomposition of isosurfaces into connected components. In this work, we are particularly interested in the continuous limit, in which these stacked bar charts correspond to stacked graphs. Furthermore, we combine this representation with the basic concepts of Sankey diagrams and flow maps to represent the edges of the contour tree as discussed in Section 3.

We also propose an algorithm to compute an appropriate approximation of the contour tree and all required isosurface statistics for a user-specified discretization of the data range in Section 4. As our visualization of the contour tree is inspired by a stacked bar chart of the sizes of connected components, we only need to sort these components appropriately to determine the basic graph layout. This sorting, however, is crucial to achieve a comprehensible visualization while preserving the most important topological structures even without crossing edges. We discuss the algorithm in Section 5 together with further details about the rendering. Results are summarized in Section 6.

The primary contributions of this work are:

- the design of a visualization of contour trees within histograms and
- the computation of appropriate graph layouts to preserve the most important edges in visualizations without crossing edges.

These contributions and our conclusions are summarized in Section 7.

2 Previous and Related Work

The contour tree (or the graph of a contour map) represents containing relationships between contours of a scalar field on a simply-connected domain. According to Freeman and Morse [3], each edge of this tree represents a contour and

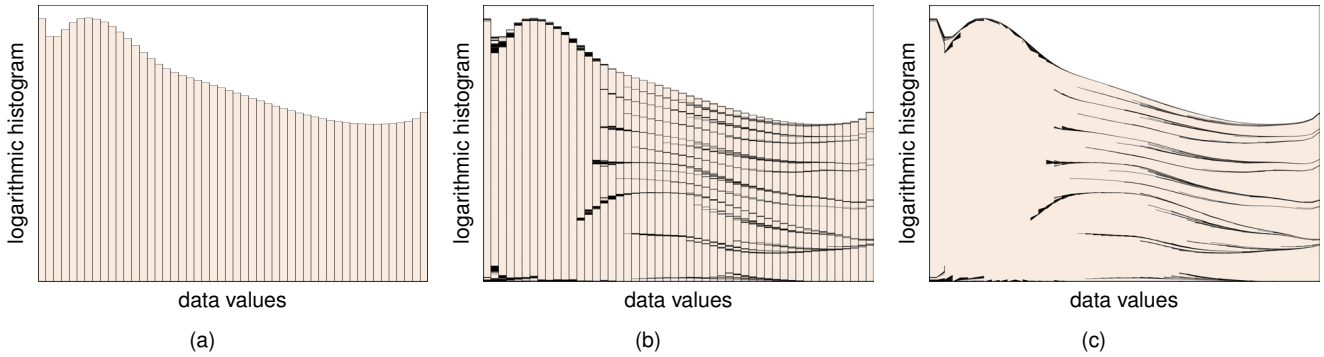

Figure 1. Illustration of the relation between (a) a volume data histogram for 50 intervals, (b) the same bars subdivided into contributions from different connected components of isosurfaces, and (c) our contour tree visualization for these 50 intervals. The data was obtained by a CT scan of the lower part of a foot; thus, the largest isosurface component corresponds to the whole foot. For larger isovalues, it splits up into the isosurface components that correspond to bones as illustrated in Figure 2.

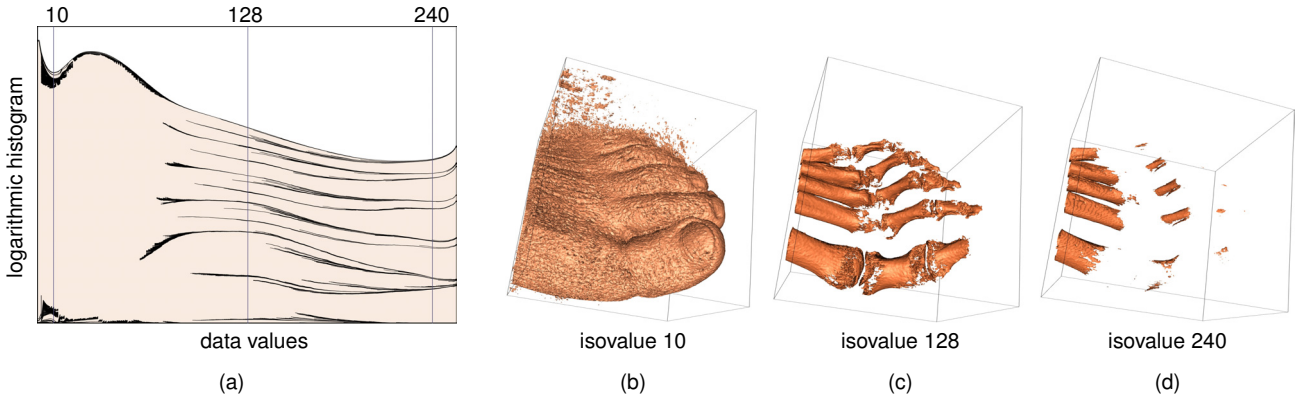

Figure 2. (a) Visualization of the contour tree of the foot data set with 200 intervals and renderings of isosurfaces corresponding to the isovalues (b) 10, (c) 128, and (d) 240.

each node represents an inter-contour region between two contours. Note that we will refer to contours as connected components of isosurfaces in this work.

In volume visualization, the contour tree has been used to accelerate the extraction of isosurfaces, for example by van Kreveld et al. [4], to generate transfer functions for direct volume rendering, for example by Takahashi et al. [5], and to segment volume data based on connected components, for example by Carr et al. [6], Takeshima et al. [7], and Weber et al. [8].

Contour trees themselves have been visualized automatically with the help of dot-and-line diagrams, for example by Bajaj et al. [1], Pascucci et al. [9, 10], and Carr et al. [6]. Shinagawa et al. [11] employed icons to represent topological relations in such diagrams. However, contour trees of real-world volume data sets are too complex for this kind of visualization; therefore, algorithms for a strong simplification have been suggested by Carr et al. [6], Takahashi et al. [5, 12], Weber et al. [8] and Pascucci et al. [10]. As it is still difficult to match the resulting graph drawing to the scalar field, the use of colors has been proposed to identify connected components [6, 8, 13]. Unfortunately, this approach is rather limited as humans cannot distinguish many colors. An alternative are three-dimensional dot-and-line diagrams [10] and the embedding of critical points and connecting lines in the three-dimensional space of the data set [5, 12, 10]. However, this can result in complex images with occlusions and line crossings, which we want to avoid in this work.

Klemelä [14] has proposed one- and two-dimensional representations of "level set trees", which are however not identical to contour trees. Nonetheless, it should be possible to apply our visualization technique to level set trees as well. Weber et al. [13] proposed "topological landscapes." The three-dimensional nature of these visualizations results in occlusions and difficulties to compare the data value of critical points, i.e., it is more difficult to compare the heights of two critical points in Weber et al.'s work than the x coordinates of two critical points in our visualization. Furthermore, some topological details are not well represented in the visualizations presented by Weber et al.; e.g., the topological noise in the engine data set (compare Figure 7(c) with Figure 10 in [13]).

Our visualization is based on histograms of volume

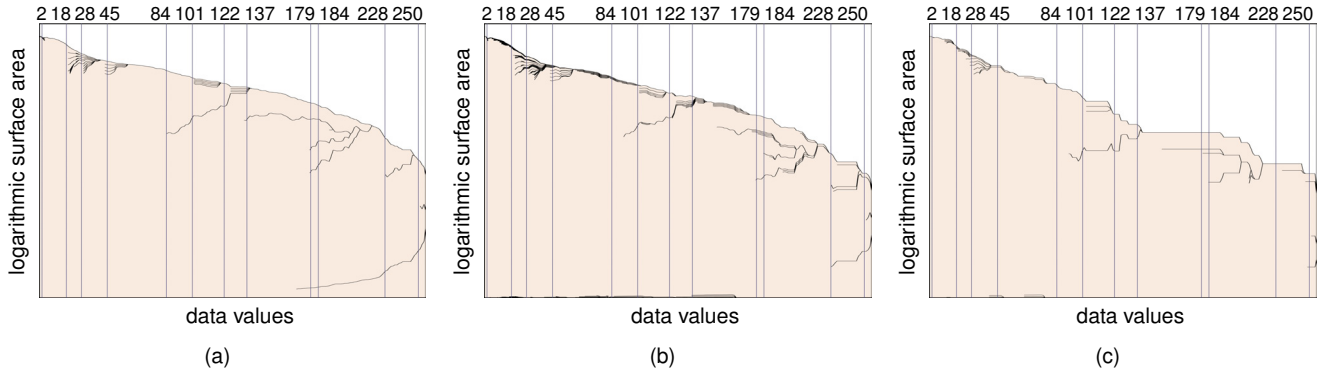

Figure 3. (a) Visualization of the contour tree of the fuel data set with markers for the 12 isosurfaces shown in Figure 4 using 200 intervals. (b) Same as (a) but for the first level of the min-max octree. (c) Same as (a) for the second level.

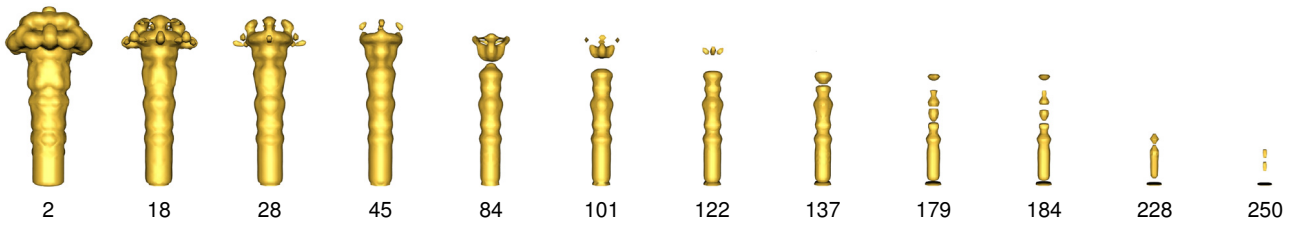

Figure 4. Isosurfaces of the fuel data set for the isovalues indicated in Figure 3.

data and employs the results of Scheidegger et al. [2] about the relation between histograms and isosurface statistics. Furthermore, the design of the proposed visualization was inspired by stacked bar charts and stacked graphs (see the works by Havre et al. [15] and Byron and Wattenberg [16] and references therein), Sankey diagrams [17] and flow maps [18].

One of the new contributions of our work is the combination of stacked graphs and Sankey diagrams. Related combinations of stacked bar charts and Sankey diagrams were published by Fry [19, section 4.6] and by Rosvall and Bergstrom [20]. In contrast to these visualizations, however, we consider approximations to continuous stacked graphs and we avoid crossing edges in order to reduce the visual complexity.

3 Design of the Proposed Visualization

Due to the wide-spread use of histograms of volume data and their close relation to isosurface statistics [2], our visualization of the contour tree resembles a stacked graph of connected isosurface components. Figure 1 illustrates the relation between histograms and the proposed visualization while the relation to isosurfaces is illustrated in Figure 2. More examples are depicted in Figures 3, 5, and 7. Figure 4 shows isosurfaces for 12 isovalues, which are indicated by vertical lines in Figure 3. Note that isosurface components in Figure 4 split into multiple components when the isovalue is equal to the data value of a saddle point. This corresponds to the start of one (or more) new separating line in Figure 3a, for example at the isovalue 84. In terms of the contour tree, this corresponds to a node with multiple children.

Apart from this basic concept of stacked graphs, there are two important design decisions. The first one concerns the displayed size of the connected components. We note that for isosurface statistics of the area of isosurfaces [1, 2], the value of each isosurface is equal to the sum of the values of all its connected components. As demonstrated in Figure 3 these isosurface statistics can be used for our visualization, too. Scheidegger et al. [2] have discussed the mathematical relation between histograms and isosurface statistics, which allows us to compute histograms from the isosurface statistics of the isosurface area. In both cases the total height of the diagram represents a sum over all connected components; thus, it is intuitive to choose the displayed size of each connected component as a fraction of the total height proportionally to the component's contribution to this sum. Strictly speaking, this is only true for linear diagrams. However, linear histograms (and isosurface statistics) are unsuitable to show all of the topologically interesting structures of the contour tree; thus, a logarithmic scale is preferable.

Another important design decision concerns crossing edges. As discussed in Section 5, we try to avoid crossing edges by sorting the connected components appropriately. However, for real-world data sets, it is impossible to avoid all crossings of straight edges if the isovalue is used as one of the coordinates of the nodes for the graph layout.

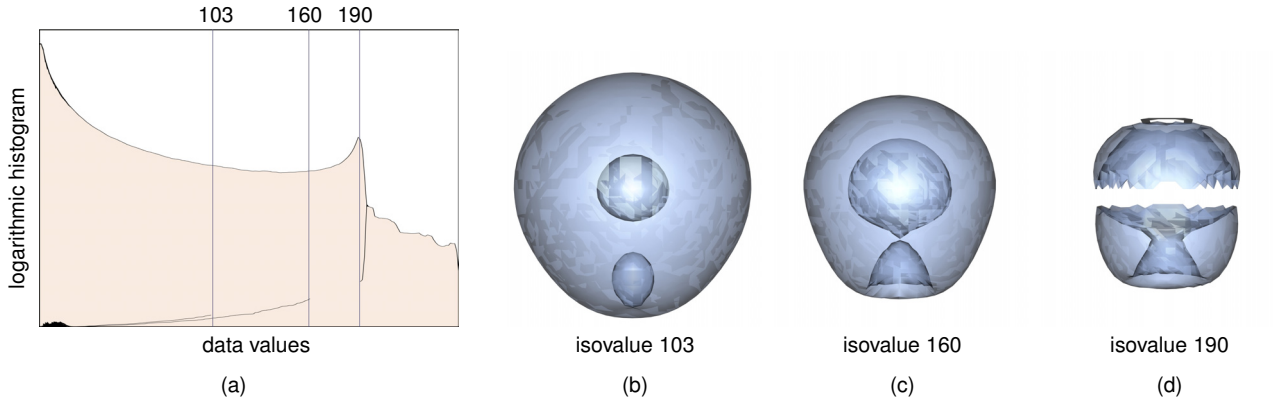

Figure 5. (a) Visualization of the contour tree of the nucleon data set with 200 intervals and renderings of isosurfaces corresponding to the isovalues (b) 103, (c) 160, and (d) 190.

(See Figure 7 in [10] for an example. Without these restrictions, crossing edges could be avoided since the contour tree is a planar graph.) Crossing edges result in occlusions and therefore increase the visual complexity of visualizations significantly. Therefore, Fry [19] and Rosvall et al. [20] employ gray shades and colors when rendering crossing edges. In the case of our visualization—and in general for stacked graphs [16]—the problem of crossing edges is even worse since the crossings might have to be visualized within a very small horizontal distance—possibly just one pixel.

On the other hand, we can take advantage of a particular feature of visualizations of complex contour trees: we cannot expect to accurately visualize the actual contour tree. In fact, our choice of the displayed sizes of connected components already results in a strong simplification since the representation of many components is too small to be visually relevant. Obviously it is not necessary to visualize edges to or from these small components. This motivates our approach to remove the less important edge of any pair of crossing edges as explained in detail in Section 5.

Thus, this simplification of the contour tree is based on the requirements of the chosen visualization instead of geometric or topological properties of the contour graph, which have been employed in previous works [6, 5, 12, 8]. We also apply this approach to the computation of approximations of contour trees as discussed next.

4 Computation of Contour Trees

There are several efficient algorithms for the computation of contour trees and their simplification [6, 9, 10], which could be employed for our visualization. Alternatively, we present an algorithm that employs a decomposition of the data range to avoid the computation of the full contour tree. This decomposition can be adjusted by the user to trade computation time for accuracy or to match the decomposition employed by a histogram.

We consider the definition of contour trees employed by Freeman and Morse [3], i.e., nodes of the contour tree represent inter-contour regions and edges represent the contours separating two inter-contour regions. Thus, each inter-contour region corresponds to a specific range of scalar data values $[r_{\min}, r_{\max}]$.

In our algorithm, we decompose the whole data range into an appropriate number of uniform intervals. These intervals are processed one-by-one in ascending order. For each interval we compute connected components of inter-contour regions, which are also known as "interval volumes" or "volume intervals" in volume visualization (see [21] and references therein). Since an approximation is sufficient, we only count the intersected cells for each connected component. To this end, we compute the minimum value $v_{\min}$ and the maximum value $v_{\max}$ of the eight vertices of each hexagonal cell in structured volume data sets. If the intervals $[r_{\min}, r_{\max}]$ and $[v_{\min}, v_{\max}]$ have a non-empty intersection, the cell is considered part of the inter-contour region. Intersected cells that share a face are considered part of the same connected component if the interval $[f_{\min}, f_{\max}]$ of data values on the face is also intersected by $[r_{\min}, r_{\max}]$. To trace connected components in regular grids, we employ a depth-first search along shared faces of cells. This approach is conservative in the sense that a component is never split into multiple components unless it actually consists of multiple, well separated components. An exhaustive depth-first search in a min-max octree is used to find seed cells for all connected components in the data set.

A cell intersected by the interval $[r_{\min}, r_{\max}]$ is not necessarily intersected by the isosurfaces for all isovalues in this interval. Therefore, a weighting factor is necessary to take the probability into account that a cell with data values between $v_{\min}$ and $v_{\max}$ is intersected by an isosurface for a random isovalue between $r_{\min}$ and $r_{\max}$. Specifically, we employ this factor:

$$\frac{\min(v_{\max}, r_{\max}) - \max(v_{\min}, r_{\min})}{r_{\max} - r_{\min}}$$

As proposed by Scheidegger et al. [2, section 5], we estimate the area of an isosurface within a cell by a constant

and approximate the gradient magnitude by the difference $v_{\max} - v_{\min}$.

There can be many thousands of connected components in noisy data sets for each interval $[r_{\min}, r_{\max}]$. To reduce the memory and time requirements of our implementation, we keep only the largest components (in our current implementation the 250 largest) since very small components are not relevant for the visualization. Thus, most of the required memory can be released before the next interval is processed.

Apart from the nodes corresponding to connected components, we also have to compute edges between these nodes, i.e., we have to determine which connected components are separated by contours. To this end, the overlap of cells intersected by adjacent intervals is recorded while the components are traced. More precisely spoken, for each interval we record the overlap of its components with components of the previously computed interval. To simplify the implementation, for each component we only record overlaps with the n largest components of the previous interval (in terms of cells), where n is a small number (6 in our current implementation).

5 Computation of the Graph Layout

5.1 Sorting of Nodes

As discussed in Section 3, crossing edges are not included in the visualization; therefore, the sorting of components is particularly important to reduce the number of crossing edges. On the other hand, it is crucial to include the most relevant edges. To this end, our algorithm assigns an "importance" to nodes of the contour tree, which is determined by the size (in cells) of the corresponding component. Furthermore, the "importance" of an edge is determined by the *size of the smaller of the two components* that correspond to the nodes of the edge. This definition of the importance of edges is plausible if noisy data is considered, which usually leads to many, very small components. The corresponding nodes are connected by edges to relevant components of all sizes, Thus, only the smaller component of each edge should be considered to determine whether the edge exists due to noise.

The sorting of the components is performed one-by-one for all data intervals $[r_{\min}, r_{\max}]$ in ascending order. Thus, the order of the components for the previous interval is known and fixed while the order of the components for the next interval is unknown. Therefore, we take only the edges to nodes of the previous interval into account. Note that there are no edges between any two nodes of the same interval.

The initial order of the nodes is determined by a projection of the centers of mass of the components onto a line. Our first attempt was to determine this line by an eigenvector analysis of the covariance matrix of the component's centers weighted with their importance. Unfortunately, this approach is only useful to compute an initial order. For this initial order, however, any line can be used. In praxis, it is reasonable to offer a few options to users, e.g., the axes of the data set.

The sorting is performed similarly to a bubble sort where pairs of nodes are swapped when their most important edges cross each other. The algorithm performs the following steps:

1. Iterate over all nodes of the current interval in their current order.

2. For each node search for a crossing edge: For a node a consider its most important edge to a node, called b, of the previously processed interval. Search for a node c of the current interval that currently precedes node a and that is connected by its most important edge to a node d of the previous interval that is preceded by node b.

3. If such a pair of crossing edges is found, swap the nodes a and c to resolve it. Continue the iteration in 1. at the new position of a. Otherwise continue at the next position after a or terminate the iteration if the end of the list has been reached.

Since this algorithm has a quadratic time complexity, it is important to limit the number of components per interval to a sufficiently small number. As mentioned in Section 4, our current implementation considers the 250 largest components per interval. It should be noted that it is rather easy to construct synthetic data sets for which this algorithm does not produce satisfactory results. However, it worked very well for all real-world data sets that we tested (see also Figure 7).

After the sorting of nodes of one interval, their edges (not only the most important ones) to the nodes of the previous interval are either accepted or rejected. In a first pass over all nodes and their edges, we accept all edges that are not crossed by a more important edge. In this pass many edges will not be accepted because of more important crossing edges. However, some of these more important edges will also fail to be accepted because of yet more important edges crossing them. Therefore, a second pass is performed over all nodes (in order of descending size of the corresponding component) and their edges. In this second pass, all edges are accepted that are not crossed by already accepted edges. While this procedure guarantees that no crossing edges are accepted, it also tries to accept as many edges as possible.

5.2 Rendering of Edges

For the visualization, each interval $[r_{\min}, r_{\max}]$ corresponds to one x coordinate. To determine the y coordinates of the separating lines, we first compute the total height of the graph based on the total size of all components of the interval. Each node is assigned a certain height determined as a fraction of the total height proportionally to

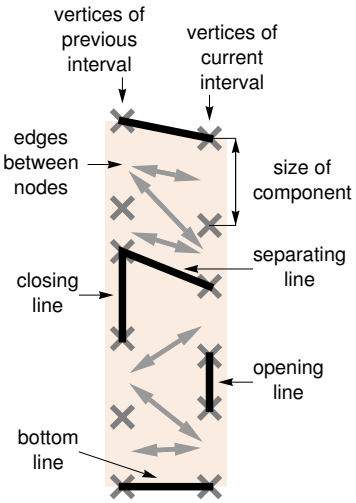

Figure 6. Illustration of the rendering of separating lines in the visualization.

the size of the corresponding component. Then, y coordinates for the sorted nodes are computed using partial sums of these heights. As illustrated in Figure 6, the resulting two-dimensional grid of vertices is used to construct lines, which separate nodes that are not connected by accepted edges.

We generate these lines by a sweep-line algorithm from smaller to larger x coordinates (i.e., left to right). Thus, data intervals are processed one-by-one analogously to the algorithms in Sections 4 and 5.1. Four different kinds of separating lines are required:

- Nodes of the previously processed interval that have no edges to the current interval have to be "closed"; e.g., by a vertical line along the height of the corresponding component.
- Nodes of the currently processed interval that have no edges to nodes of the previous interval have to be "opened"; e.g., by a vertical line along the height of the corresponding component.
- The nodes of the current interval are processed in order of ascending y coordinates. Their edges to nodes of the previous interval are sorted with respect to the y coordinates of those nodes. If the first edge of a node a of the current interval links to a node b that has no edge to any of the nodes that precede a in the current interval then a separating line is drawn below the components corresponding to a and b.
- Two lines are rendered at the top and the bottom of the graph.

This basic rendering has been further refined to reduce the number of vertical lines; however, these improvements become visually less important as the number of intervals is increased (and their size is decreased).

Table 1. Timings for various data sets [22, 23].

		time in seconds		
data set	size	intervals: 50	100	200
nucleon	$41 \times 41 \times 41$	0.3	0.6	1.0
silicium	$98 \times 34 \times 34$	1.2	2.1	4.0
fuel	$64 \times 64 \times 64$	0.6	0.7	1.0
neghip	$64 \times 64 \times 64$	1.1	1.9	3.3
lobster	$301 \times 324 \times 56$	17.7	25.6	41.1
MR brain	$256 \times 256 \times 109$	32.2	53.3	92.7
CT head	$256 \times 256 \times 113$	32.6	51.8	88.7
engine	$256 \times 256 \times 128$	34.2	53.4	91.4
leg of statue	$341 \times 341 \times 93$	27.8	35.2	51.1
teapot	$256 \times 256 \times 178$	32.9	44.4	66.7
aneurism	$256 \times 256 \times 256$	40.4	49.4	67.4
bonsai	$256 \times 256 \times 256$	56.8	82.8	134.8
foot	$256 \times 256 \times 256$	79.9	127.4	215.1
skull	$256 \times 256 \times 256$	81.7	129.3	223.3

6 Results

6.1 Performance

Our prototypical implementation employs VTK [24] to load various publicly available data sets [22, 23] and to visualize the isosurfaces in Figures 2, 4, and 5. The C++/OpenGL implementation of the algorithms described in Sections 4 and 5 was tested on a PC with two 3.6 GHz Intel Pentium 4 CPUs (only one of them was used for our program) and 2 GB RAM running Microsoft Windows XP. Table 1 shows the resulting timings for 50, 100, and 200 data intervals. All visualizations presented in this work employ 200 data intervals, except for Figure 1, which employs 50 intervals.

Note that the timings in Table 1 include the computation of the surface area of all isosurface components and the graph layout. On the other hand, only an approximation to the actual contour tree is computed. Thus, a comparison to previously reported results is difficult. However, the results show that we can effectively trade accuracy for performance by reducing the number of data intervals.

6.2 Downsampling

To obtain a fast preview, the data set can be downsampled before the contour tree is computed. To this end, the min-max octree can be employed, which is also used to find seed cells of connected components. Thus, contour trees can be computed for any level of this min-max octree. Note that our algorithm requires the min-max intervals of all faces of each cell. Thus, these intervals have to be computed for the downsampled cells in addition to the min-max intervals of the cells themselves. Figures 3b and 3c show the results for the 1st and 2nd level. While there are strong deviations from the original computation (compare with Figure 3a), the most important structures of the contour tree are preserved in Figure 3b.

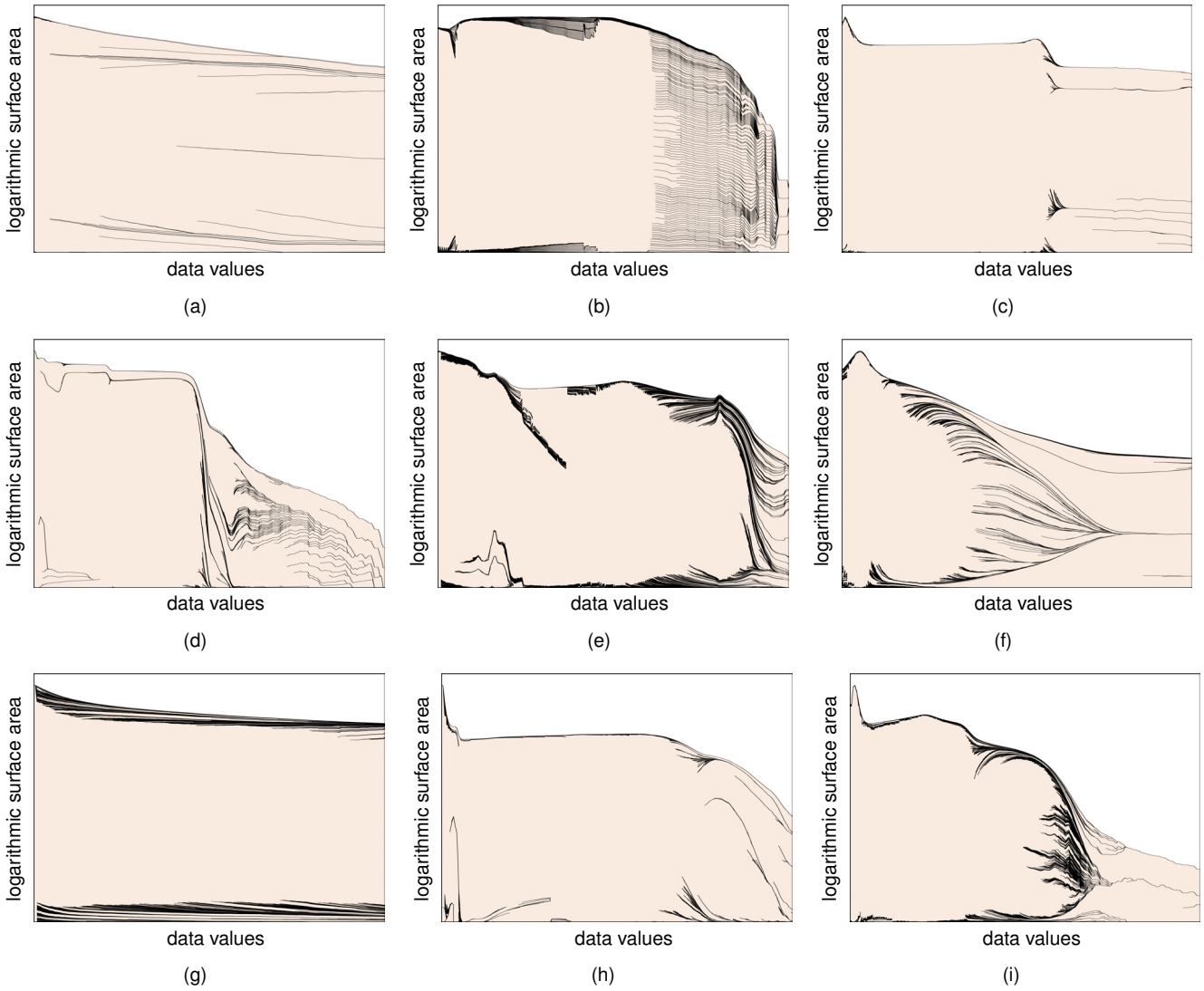

Figure 7. Visualizations of the contour trees of various well-known data sets, each using 200 data intervals: (a) the neghip data set, (b) the silicium data set, (c) the engine data set, (d) the teapot data set, (e) the bonsai data set, (f) the skull data set, (g) the aneurism data set, (h) the leg of statue data set, and (i) the MR brain data set.

7 Conclusion

We have presented a visualization of contour trees, which is particular well suited to convey the most important—i.e., largest—components of isosurfaces even in noisy data sets. Since this visualization is integrated in a histogram, it is likely to be more comprehensible to many users than previously published representations. Moreover, it offers strictly more information than a histogram; therefore, we assume that it is at least as useful as a histogram. In fact, we assume that the visual representation of splitting and joining isosurface components is comprehensible even to users who are not familiar with contour trees. Therefore, we hope that this representation will also help to popularize the contour tree and its applications in volume graphics.

Figure 7 suggested that there is a wide range of visually very different results even for this small set of data sets. The variety of visual characteristics will hopefully encourage more detailed explorations of the contour trees of specific kinds of data sets.

There are many avenues for future work: isosurface components could be selected by clicking; rendering and editing of selected isosurface components could be integrated; zooming, panning, and further filtering options could be integrated; etc.

8 Acknowledgements

The author would like to thank all reviewers. The volume data sets appear courtesy of the Stanford Volume Data Archive [23] and the Volvis Homepage [22]. This work has been supported by the German Research Council (DFG) under grant no. KR2948/2-2.

References

[1] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. The contour spectrum. In *Proceedings of the conference on Visualization '97*, pages 167–ff., 1997.

[2] C.E. Scheidegger, J.M. Schreiner, B. Duffy, H. Carr, and C.T. Silva. Revisiting histograms and isosurface statistics. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1659–1666, Nov.-Dec. 2008.

[3] S. Freeman and S. P. Morse. On searching a contour map for a given terrain elevation profile. *Journal of the Franklin Institute*, 284(1):1–25, July 1967.

[4] Marc van Kreveld, René van Oostrum, Chandrajit Bajaj, Valerio Pascucci, and Dan Schikore. Contour trees and small seed sets for isosurface traversal. In *SCG '97: Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 212–220, New York, NY, USA, 1997. ACM.

[5] Shigeo Takahashi, Yuriko Takeshima, and Issei Fujishiro. Topological volume skeletonization and its application to transfer function design. *Graph. Models*, 66(1):24–49, 2004.

[6] Hamish Carr, Jack Snoeyink, and Michiel van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *Proceedings of the conference on Visualization '04*, pages 497–504, 2004.

[7] Y. Takeshima, S. Takahashi, I. Fujishiro, and G. M. Nielson. Introducing topological attributes for objective-based visualization of simulated datasets. *Volume Graphics, 2005. Fourth International Workshop on*, pages 137–236, June 2005.

[8] Gunther H. Weber, Scott E. Dillard, Hamish Carr, Valerio Pascucci, and Bernd Hamann. Topology-controlled volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 13(2):330–341, 2007.

[9] V. Pascucci and K. Cole-McLaughlin. Efficient computation of the topology of level sets. In *Proceedings of the conference on Visualization '02*, pages 187–194, 2002.

[10] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli. Multi-resolution computation and presentation of contour trees. In *Proceedings IASTED Conference Visualization, Imaging, and Image Processing*, pages 452–290, 2004.

[11] Y. Shinagawa, T.L. Kunii, and Y.L. Kergosien. Surface coding based on morse theory. *IEEE Computer Graphics and Applications*, 11(5):66–78, Sep 1991.

[12] S. Takahashi, Y. Takeshima, G. M. Nielson, and I. Fujishiro. Topological volume skeletonization using adaptive tetrahedralization. *Geometric Modeling and Processing, 2004. Proceedings*, pages 227–236, 2004.

[13] G.H. Weber, P.-T. Bremer, and V. Pascucci. Topological landscapes: A terrain metaphor for scientific data. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1416–1423, Nov.-Dec. 2007.

[14] Jussi Klemelä. Visualization of multivariate density estimates with level set trees. *Journal of Computational and Graphical Statistics*, 13(3):599–620, 2004.

[15] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. Themeriver: Visualizing thematic changes in large document collections. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):9–20, Jan/Mar 2002.

[16] Lee Byron and Martin Wattenberg. Stacked graphs — geometry & aesthetics. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1245–1252, 2008.

[17] Patrick Riehmann, Manfred Hanfler, and Bernd Froehlich. Interactive Sankey diagrams. In *Proceedings of 2005 IEEE Symposium on Information Visualization*, page 31, 2005.

[18] Doantam Phan, Ling Xiao, Ron Yeh, Pat Hanrahan, and Terry Winograd. Flow map layout. In *Proceedings of the 2005 IEEE Symposium on Information Visualization*, page 29, 2005.

[19] Benjamin Jotham Fry. *Computational Information Design*. PhD thesis, 2004. Supervisor: John Maeda.

[20] M. Rosvall and C. T. Bergstrom. Mapping change in large networks, 2008. URL: http://arxiv.org/abs/0812.1242v1; last accessed March 12, 2009.

[21] P. Bhaniramka, Caixia Zhang, Daqing Xue, R. Crawfis, and R. Wenger. Volume interval segmentation and rendering. In *Volume Visualization and Graphics, 2004 IEEE Symposium on*, pages 55–62, Oct. 2004.

[22] D. Bartz. Volren and volvis homepage, 2005. URL: http://www.volvis.org/; last accessed March 20, 2009.

[23] M. Levoy. The stanford volume data archive, 2001. URL: http://graphics.stanford.edu/data/voldata/; last accessed March 20, 2009.

[24] Kitware, Inc. *The Visualization Toolkit User's Guide*, 2006.