# Flexible and Local Isosurfaces - Using Topology for Exploratory Visualization

Hamish Carr*

Dept. of Computer Science
University of British Columbia
Vancouver, BC, Canada.

Jack Snoeyink†

Department of Computer Science
University of North Carolina - Chapel Hill
Chapel Hill, NC, USA

February 5, 2003

**Abstract**

The contour tree is a topological abstraction of a scalar field, used to accelerate isosurface extraction and to represent the topology of the scalar field visually. We simplify the minimal seed sets of van Kreveld et al. by extracting isosurface seeds directly from the contour tree at run-time, and by guaranteeing that no redundant seeds are generated. We then extend the contour spectrum of Bajaj et al. as an interface for *flexible isosurfaces*, in which multiple individual contour surfaces with different isovalues can be displayed, manipulated and annotated. Finally, we show that the largest contour segmentation of Manders et al. , in which separate surfaces are generated for each local maximum of the field, is in fact a special case of the flexible isosurface.

## 1 Introduction

One of the fundamental techniques 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 multiple connected components: we refer to the connected components of the isosurface as *contour surfaces*.

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 cases, a suitable isosurface is already known, especially when the task has been performed many times.

*hcarr@cs.ubc.ca, http://www.cs.ubc.ca/

†snoeyink@cs.unc.edu, http://www.cs.unc.edu/ snoeyink

In other applications, the specific isovalue of interest is unknown, but it is known that boundaries are of particular interest. An example of this is the boundary between the heart and the rest of the body in a medical data set. Where boundaries are desired, it is possible to select a suitable isovalue automatically, based on the gradient of the field [16].

Not all applications focus on boundaries: computational fluid dynamics and molecular modelling, for example, give rise to data sets where the specific isosurface of interest is not immediately obvious. In applications such as these, where no *a priori* information is available, the general approach is exploratory. The user tries different isovalues interactively, until a suitable one is found. Tools that give cues to interesting isovalues are important, and we review these in Section 2.

One technique for efficient isosurface extraction uses the *contour tree*: a topological abstraction of the development of isosurfaces as the isovalue is varied. In Section 3, we give a brief description of this structure. Then, in Section 4, we describe *path seeds*: an improved and simplified technique for generating isosurface seeds from the contour tree. We show how to integrate the computation of these seeds with the contour tree algorithm of Carr, Snoeyink & Axen [4]. We also show how to dispense with the seed sets described by van Kreveld et al.. Instead, we show how to generate seeds directly from the contour tree, guaranteeing a 1-1 correspondence between contour surfaces, isosurface seeds, and superarcs of the contour tree.

This 1-1 correspondence makes it easy to identify individual contour surfaces, and to track individual surfaces as the isovalue varies. In Section 5, we use this to extend and improve the contour spectrum of Bajaj, Pascucci & Schikore [1]. We call this approach the *flexible isosurface*, and use it to manipulate individual contour surfaces independently, to annotate these surfaces by storing the annotations on the contour tree, and to allow visual selection of which contour surfaces to display.

Since isosurfaces are restricted to single isovalues, it can be quite difficult to select an isosurface that captures all, or most, of the interesting information. One existing method, the local contour segmentation of Manders et al. [13] is shown in Section 6 to be a special case of the flexible isosurface.

We make some comments on implementation issues in Section 7, then state our conclusions in Section 8, and discuss some possible further extensions of this tool in Section 9.

## 2 Previous Work

Previous work from several areas is relevant to a discussion of flexible isosurfaces. These areas include automatic isovalue selection [16, 25], isosurface interfaces [1, 10], interactive transfer function design [11], isosurface generation [1, 4, 26], feature tracking [17, 19, 20], and segmentation [13].

There are several ways to locate an isosurface that displays interesting information. The simplest approach is to display the isosurface interactively. The user interacts with the software, trying different isovalues until a suitable isosurface is found. This is faster when cues are available that guide the choice of isovalues. Bajaj, Pascucci & Schikore [1] display spatial and topological information in a separate display called the *contour spectrum*. This displays information such as surface area, field gradient, and isosurface connectivity. The connectivity, in particular, is displayed by drawing the contour tree in the contour spectrum.

Kettner, Rossignac & Snoeyink [10] introduced a related interface, called SAFARI. This interface displays the number of contour surfaces in each isosurface in a panel similar to the contour spectrum. Since the isosurface can vary by time or by isovalue, this results in a panel encoding the variation of topology over these two dimensions. The contour trees for individual times steps are used to extract this information efficiently without explicitly constructing isosurfaces.

Instead of interactivity, some authors use statistical approaches to determine suitable isovalues. Pekar, Wiemker & Hempel [16] use a histogram of gradient information to determine the isovalues for which the gradient is steepest, assuming that steep gradients mark significant boundaries. Tenginakai, Lee & Machiraju [25] use statistical signatures of the local distribution of voxel values to determine interesting isovalues. This information is used to construct a transfer function for volume rendering, which maps isovalues to opacity.

These approaches all assume that a given isovalue has uniform importance throughout the data set. Kniss, Kindlmann & Hansen [11] note that this assumption causes problems, and construct transfer functions interactively, adding the gradient as an input parameter. While this successfully adds some spatial information to the task of designing transfer functions, the authors note that this is less useful where boundaries are not sharply defined.

The contour tree has also been used as an index structure for extracting contour lines [3, 26] and as an abstract description of a landscape [5, 12, 21, 22]. We will describe the contour tree in more detail in Section 3. For now, we observe that the contour tree encodes the connectivity of the field for all isovalues, and that it can be used to generate contours or displayed directly as an abstraction of the data.

Van Kreveld et al. [26] gave an algorithm for computing the contour tree on simplicial meshes, and noted that the contour tree could be used to extract isosurfaces from volume data. Tarasov & Vyalyi [23] and Carr, Snoeyink & Axen [4] extended and improved this algorithm in three and higher dimensions. Pascucci [14] independently extended the algorithm of van Kreveld et al. [26] to track topological genus, as well as connectivity. Subsequently, Pascucci & Cole-McLaughlin [15] added topological genus to the algorithm of Carr, Snoeyink & Axen [4], and adapted the algorithm to handle cubic cells with a trilinear interpolation function.

The contour tree is not the only topological structure that has been used for isosurfaces. Itoh & Koyamada [7, 8] use a related heuristic structure called the *extrema graph*. Itoh, Yamaguchi & Koyamada [9] extended this to *volume thinning*, in which a set of seeds for extracting contours is constructed by discarding redundant samples until no more can be discarded. This gives a volumetric skeleton for the data set, which encodes the same connectivity as the contour tree does. Bajaj, Pascucci & Schikore [2] used a similar thinning technique to construct the contour tree. Gagvani, Kenchammana-Hosekote & Silver [6] used almost the same volumetric skeleton to animate volumetric data.

Another related structure is the Reeb graph, which tracks the development of arbitrary surfaces as some parameter varies. The Reeb graph has been used for surface reconstruction from two-dimensional slices by Shinagawa & Kunii [18], and to track the development of isosurfaces over time by Samtaney et al. [17] and by Silver and Wang [19, 20]. Since the contour tree tracks the development of isosurfaces as the isovalue varies, the contour tree is precisely the Reeb graph with respect to isovalue.

Work related to contour surfaces can also be found in cell biology. Manders et al. [13] use *largest contour segmentation* to detect features called organelles in individual cells. They note that boundaries between organelles and the surrounding cytoplasm are usually not sharply defined. Instead of finding sharp boundaries, largest contour segmentation uses maximal contour surfaces which contain exactly one local maximum. These *largest contours* are extracted from the data by growing regions independently from each local maximum until a saddle point is detected. As we will see in Section 6, this is a special case of the flexible isosurfaces which we introduce in this paper.

The common thread here is identification of important features, either automatically or interactively. Some approaches use gradient information: others use isovalue information. However, with the exception of the largest contour segmentation of Manders et al. [13], all of them use global properties of the isovalue: properties consistent throughout the data set. We observe that the contour tree combines this global information with spatially local information, allowing us greater control over what is displayed on the screen. Before turning to this, we review the contour tree in more detail in the next section.

## 3 The Contour Tree

The *contour tree* is a structure that represents the topological evolution of a data set as the isovalue varies. It is best described by showing an example.

Figures 1 (a) - (f) show six isosurfaces of a small data set in order from high to low. Figure 1(g) shows the corresponding contour tree: the horizontal lines correspond exactly to the isosurfaces shown in Figure 1. If we imagine a sweep from high isovalues to low isovalues, we see four small surfaces appear in sequence (10, 9, 8, 7): these correspond to the four leaves at the top of the contour tree. The surfaces join (6, 5) in pairs, forming larger surfaces, which quickly become rings. These rings then flatten out into cushions, which join (4) to form a single surface. This surface gradually wraps around a hollow core, and pinches off at (3), splitting in two: one surface faces inwards, the other outwards. The inward surface contracts until it disappears at (2): the outward contour expands until it reaches the global minimum (1).

More formally, the contour tree is defined in terms of equivalence classes of contours [4]. A simple, yet accurate description is that the contour tree is the result of contracting each possible contour surface to a single point [14]. A contour surface that passes through a local extremum, or through a saddle point at which the connectivity of the contour surfaces changes, always contracts to a vertex of the contour tree: the extremum or saddle is called a *critical point*. Otherwise, the contour surface contracts to a point on one of the edges of the tree, and is called a *regular point*. For example, in Figure 2(b), vertex 5 is a critical point: vertex 6.5 is a regular point.

Van Kreveld et al. noted that, for simplicial meshes, critical points can only occur at vertices of the mesh. As a result, computing contour trees for simplicial meshes can be computed combinatorially.

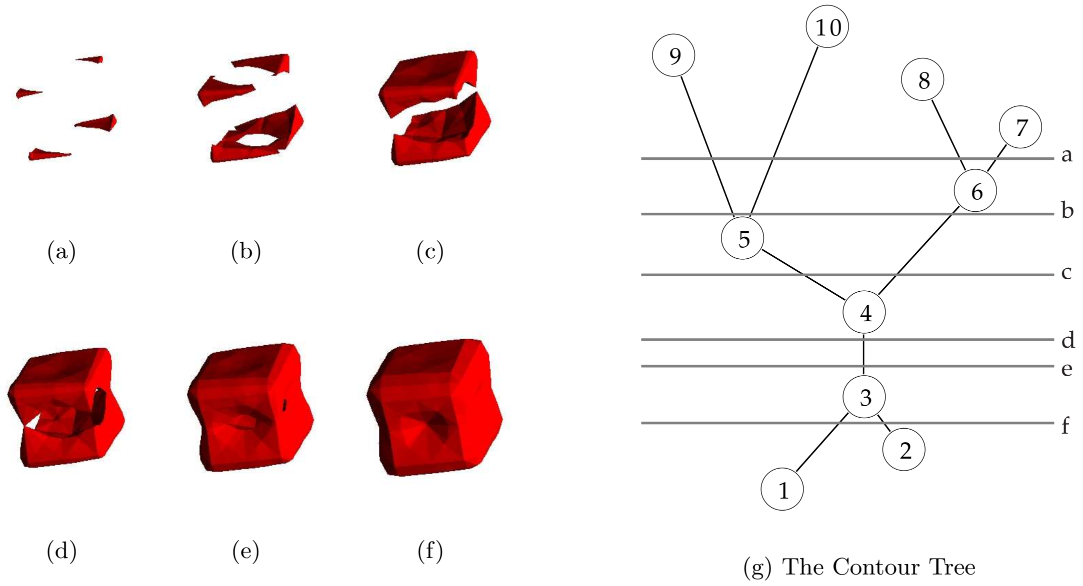


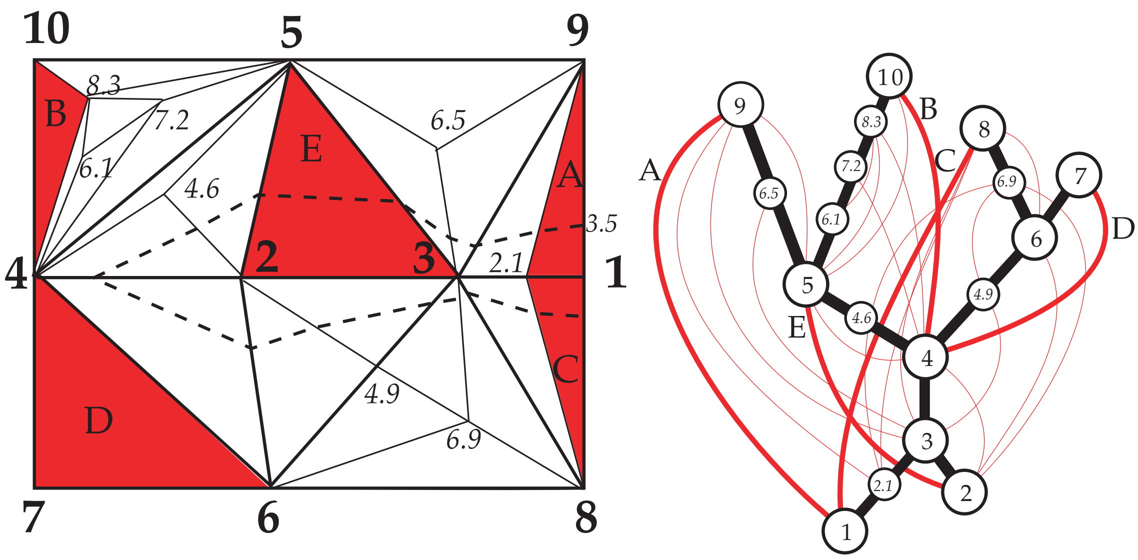

Figure 1: Sample Development of Level Sets in 3-D

They gave an algorithm to construct the contour tree, and showed how to use it to generate a *minimal seed set*: a set of seed cells guaranteed to intersect each possible contour surface. Figure 2 shows a small example in two dimensions, based on a triangulation (Figure 2(a)) with the same contour tree (Figure 2(b)) as that used in Figure 1. In Figure 2(b), the edges of the contour tree are shown as thick black lines. For each cell (i.e. triangle) in Figure 2(a), an edge is added between the highest- and lowest-isovalued vertices. This thin red edge represents the path through the contour tree between these two vertices of the cell. Every contour that intersects the cell is represented in the tree by a point on this path.

For example, consider cell A. The extreme vertices of this cell are 1 and 9, so we add a red edge from 1 to 9 in the contour tree. Every contour represented in the tree by a point on the path 1 - 2.1 - 3 - 4 - 4.6 - 5 - 6.5 - 9 intersects cell A. This implies that A is a sufficient seed cell for all of these cells.

By representing each cell as a path in the contour tree, van Kreveld et al. reduced the problem to that of choosing a minimal set of red paths that cover all the edges of the contour tree. The cells corresponding to these paths then intersect all contours, and form a sufficient set of seeds to generate all isosurfaces. In this case, one such minimal seed set is the set of five medium-weight red edges marked A-E. The corresponding triangles A-E in the triangulation are shown in red in Figure 2(a).

Several drawbacks to this approach can be noted. First, although the minimal seed set can be computed in polynomial time, the time and storage requirements are excessive. Van Kreveld et al. [26] do not specify the polynomial, but give an approximation 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, where $n$ is the size of the inpu mesh. This approximation guarantees that no more than twice the minimum number of seed cells are chosen.




(a) A Small Triangulation in Two Dimensions

(b) Using the Contour Tree to Find Seeds

Figure 2: A Two-Dimensional Example Showing How to Find Minimal Seed Sets

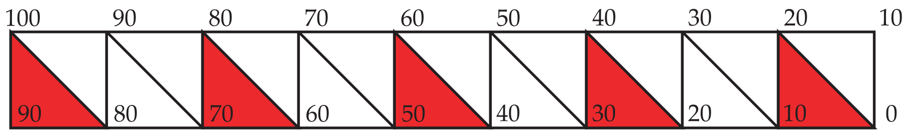
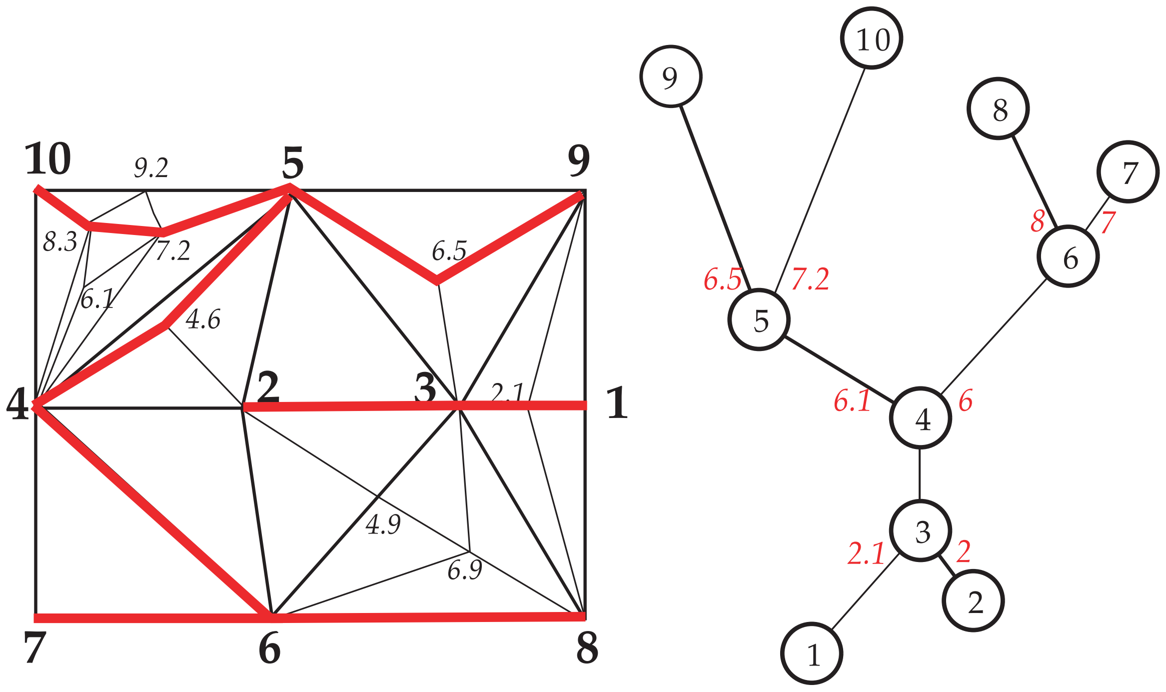



Figure 3: A Simple Mesh with Large Minimal Seed Set

Second, the size of this seed set can be significant Since at least half of the supernodes of the tree are local extrema, and no cell can contain more than one minimum and one maximum, it follows that the seed set must be $\Omega(t)$, where $t$ is the number of supernodes in the contour tree. A trivial upper bound is $O(n)$, as each red edge covers at least one arc of the contour tree: thus, there is always a seed set of size $n$. It is not difficult to construct worst case examples for which the minimal seed set is significantly larger than $t$. Consider an inclined triangle strip, as shown in Figure 3. In this case, the contour tree will consist of a single superarc (i.e. $t = 1$), but the minimal seed set always uses $n/4$ cells. In $d$ dimensions, this construction can be extended to give cell strips requiring $\frac{n}{2d}$ seed cells.

Third, although the contour tree is used to construct the seed set, the correspondence between its superarcs and individual contours in the output is lost. This is because each contour may have more than one seed cell. In Figure 2(a), the contour at 3.5 is shown as a dotted line. This contour intersects three of the red seed triangles: A, C, and E. This many-to-one correspondence makes it difficult to relate contours in the output to the superarcs to which they correspond.

This also makes it necessary to mark which cells have been visited, so that the same contour is not extracted multiple times. This is particularly onerous for two-dimensional data, which does not need such flags for contour extraction. Starting at the seed cell, we follow the contour in either




(a) Using Monotone Paths as Seeds

(b) Storing Path Seeds in the Contour Tree

Figure 4: Generating Path Seeds Directly From the Contour Tree

direction until we return to the seed cell, or we reach the boundary of the data set. If we reach the boundary, we return to the seed cell and extract the other half-contour by travelling in the opposite direction. For three- and higher-dimensional data, however, we always have to store flags for each cell in the mesh.

Having noted these problems with minimal seed sets, we next describe an improvement to the seed set approach: path seeds.

# 4 Path Seeds

> The distance is nothing: it is only the first step that is difficult.
>
> *The Marquise du Deffand (1697-1780), on the legend that St. Denis walked two leagues carrying his head after it was cut off*

In the previous section, we reviewed the construction of minimal seed sets from the contour tree. In this section, we show how to bypass this step, and generate seeds directly from the contour tree as needed. Instead of generating seed cells, we generate seed edges (i.e. edges that intersect the desired contour). In practice, either seed edges or seed cells suffice, as any cell including the edge is a seed cell, and there is always at least one edge in a seed cell that can be used as a seed edge.

Moreover, we will choose our seed edges so that they form a set of monotone paths starting at critical points.

Consider Figure 4(a), in which the heavy red edges mark monotone paths starting at critical points. Each path corresponds to a superarc in the contour tree shown in Figure 4(b). 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 8.9 along the superarc $5-10$, we start along the edge $5-7.2$, then ascend along edges $7.2-8.3$ and $8.3-10$ until we reach the isovalue 8.9.

But, once we have departed in the right direction from 5 going in the right direction, it is trivial to choose a suitable path. After following the edge $5-7.2$, we have a choice: to follow $7.2-8.3$ or $7.2-9.2$. Since 7.2 is not a critical point, all ascending edges pass through the same set of contours. The same is true at 8.3 or 9.2, until we reach the isovalue of the critical point 10. All we need to store to generate a seed edge is this critical first step: $5-7.2$. We call these "first steps" *path seeds*, as they provide seeds for us to construct a path which generates the desired seed cells or seed edges.

Storing this additional information on the contour tree takes $\Theta(t)$ space. Using it to compute contour seeds on the fly does add an additional cost: this can be as much as $\Omega(n)$ for the inclined strip referred to in the previous section. In most data sets of practical interest, however, this cost is dwarfed by the cost of the actual contour extraction, estimated to be $O(N^{(d-1)/d})$ [8].

How do we compute these path seeds? In [4], Carr, Snoeyink & Axen describe how to merge two partial structures called the *join tree* and *split tree* to obtain the contour tree. The join tree is constructed by sweeping from high to low isovalues, incrementally using Tarjan's union-find structure [24] to determine connected components. When vertex 5 is added to the union-find, so are the edges from 5 to 7.2, 9.2, and 9. Immediately before adding 5, these belong to two different union-find components: one containing 9, the other containing 6.1, 7.2, 8.3, 9.2, and 10. This merge in the union-find structure is precisely what identifies 5 as a join in the contour tree. At this point, we know that $5-7.2$ and $5-9$ are edges ascending from 5 into the two connected components of $\{x : f(x) \geq h\}$ that join at 5. We store this information in the join tree, and transfer it to the contour tree during the merge step of the algorithm. Similarly, we store $3-2.1$ and $3-1$ in the split tree when we identify 3 as a split, and transfer these path seeds to the contour tree.

Extracting the path seeds in this way adds constant cost to generating each edge of the join or split tree, and constant cost to transferring edges to the contour tree. Thus, the asymptotic cost is exactly the same as for the contour tree algorithm: $O(n \log n + N + t\alpha(t))$ [1]. It follows that path seeds are cheaper than the minimal seed sets of van Kreveld et al. [26], especially in dimensions higher than two.

Finally, we note that this approach generates one and only one seed for each contour. This allows us to keep track of individual contours as separate objects. In the next section, we show how this can be used to generalize the concept of an isosurface, and to use the contour tree as a control for manipulating or annotating individual contour surfaces.

[1] See Appendix A for additional details on the analysis

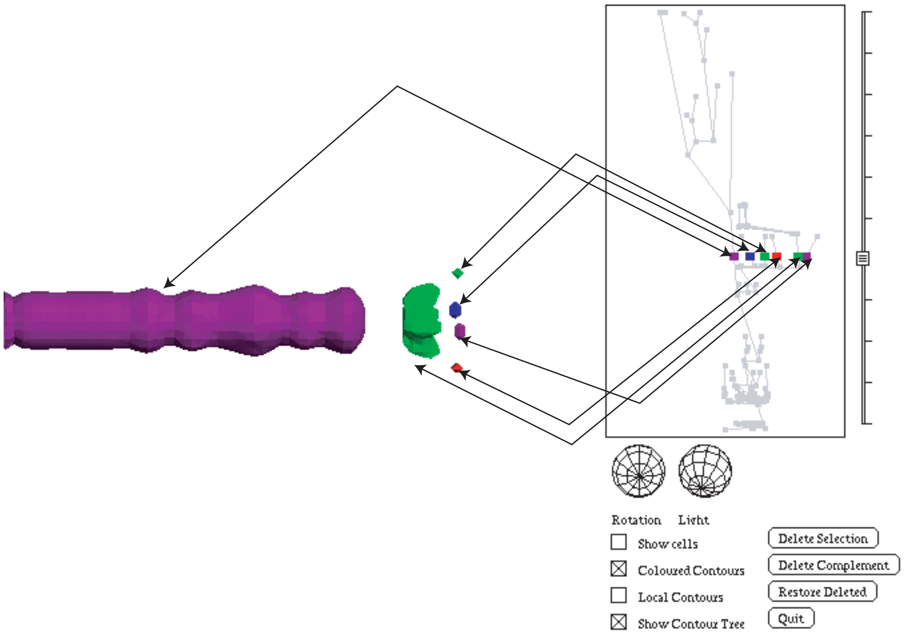



Figure 5: A Level Set, with Colour-Coded Tags

## 5 Flexible Isosurfaces

Bajaj, Pascucci & Schikore [1] showed the contour tree alongside the isosurface in a display that they called the *contour spectrum*. The contour spectrum is, however, a static display, and cannot be used to single out individual contours, or to isolate them in the displayed isosurface. In this section, we show how to use the correspondence between superarcs and contours to do so.

In the preceding section, we showed how to use path seeds to guarantee that, for any given isovalue, exactly one seed is generated for each contour. From this, if we identify a particular superarc in the contour tree, we can also identify it in the isosurface display, and vice versa. Not only can we do so, we can use the superarc in the contour tree as a proxy for the entire contour. To illustrate, consider Figure 5, in which the display is divided into the *contour display* on the left, and the *contour tree* on the right.

Each contour in the contour display is arbitrarily assigned a colour to distinguish it from other contours. In the contour tree, the corresponding superarc is marked by a tag of the same colour, as shown by the arrows in Figure 5. By colour-coordinating, we explicitly show the relationship between the contour tree and the individual contours in the contour display. At present, the colours are drawn from a small palette of easily distinguished colours, and are based on an ID number for the superarc. In future, we hope to assign colours based on the structure of the tree.

Moreover, since the vertical axis of the contour tree is the isovalue, we place the tag so that it intersects the superarc at the isovalue of the contour. A level set, or conventional isosurface, then corresponds to the set of all possible tags at that height in the contour tree. The flexible isosurface will consist of a more generalized set of tags, not all of which need be at the same isovalue.

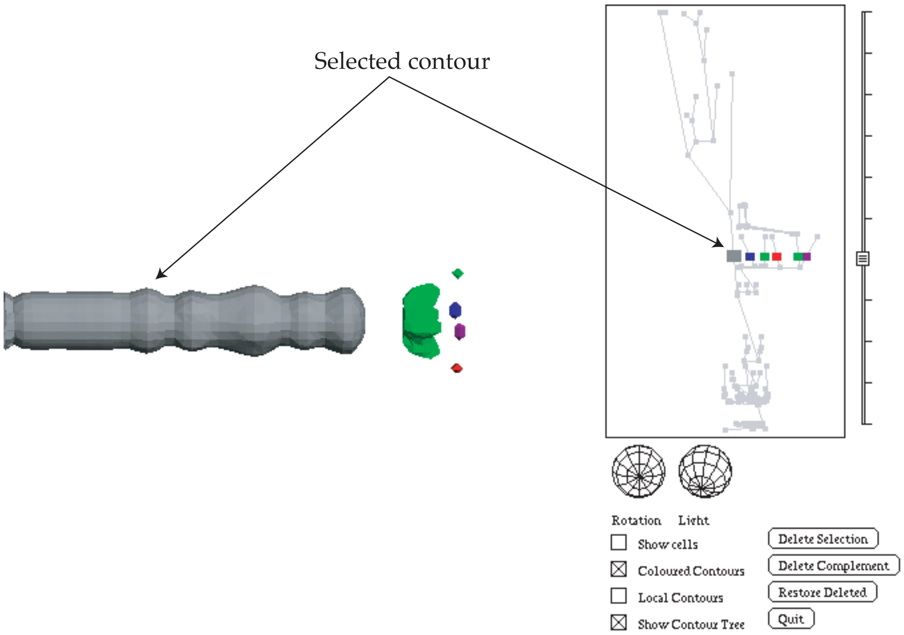



Figure 6: Selecting The Leftmost Contour

Returning to the interface, the first operation that we might wish to perform is to select an individual contour. This can be done either by clicking on the contour in the contour display, or clicking on the corresponding tag in the contour tree. Following the usual metaphor of user interfaces, we highlight both the contour and the tag to indicate which have been selected. In Figure 6, we show the effect of doing this. In this case, since we are using colours to code individual contours, we use grey for the highlighted contour, and slightly increase the size of the corresponding tags in the contour tree.

Once a contour has been selected, it can be adjusted by manipulating the isovalue slider. As the isovalue is adjusted, the contour is adjusted, both in the image and in the contour tree. In Figure 7, we have adjusted the isovalue upwards. As the isovalue increases, the contour shrinks, and breaks into two pieces. In the contour tree, the tag slides up the superarc until it reaches the join, then separates into two tags. This allows us to adjust an individual contour, and keep track of all of the pieces into which it breaks.

As we continue dragging in Figure 8, the contour breaks into more pieces, all of which are tracked by their relationship in the contour tree. Finally, we deselect the contour, and the new contours remain and are assigned arbitrary colours themselves, as shown in Figure 9. We now have, not a level set, but a set of contours at mixed isovalues: this is what we call a *flexible isosurface*.

In addition to cross-referencing the contour display and the contour tree, the colours serve two extra purposes. First, they increase the visual contrast between different contours in intricate situations, such as that in Figure 12. Second, simply by assigning the colour "invisible", we can suppress individual contours which block the view: Figure 10 shows an example, using the UNC "3dhead" MRI data set.

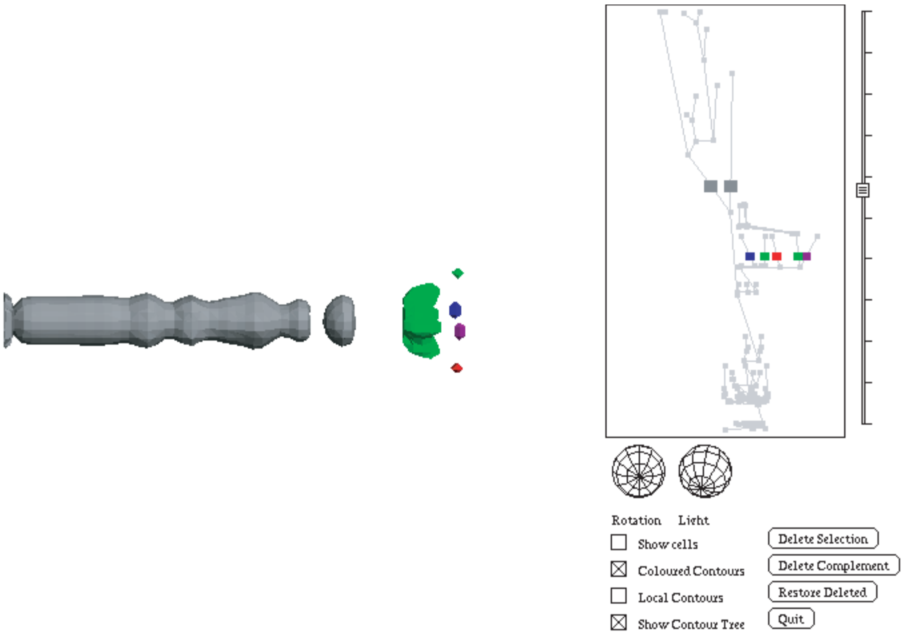



Figure 7: Adjusting The Isovalue of the Leftmost Component of Figure 5

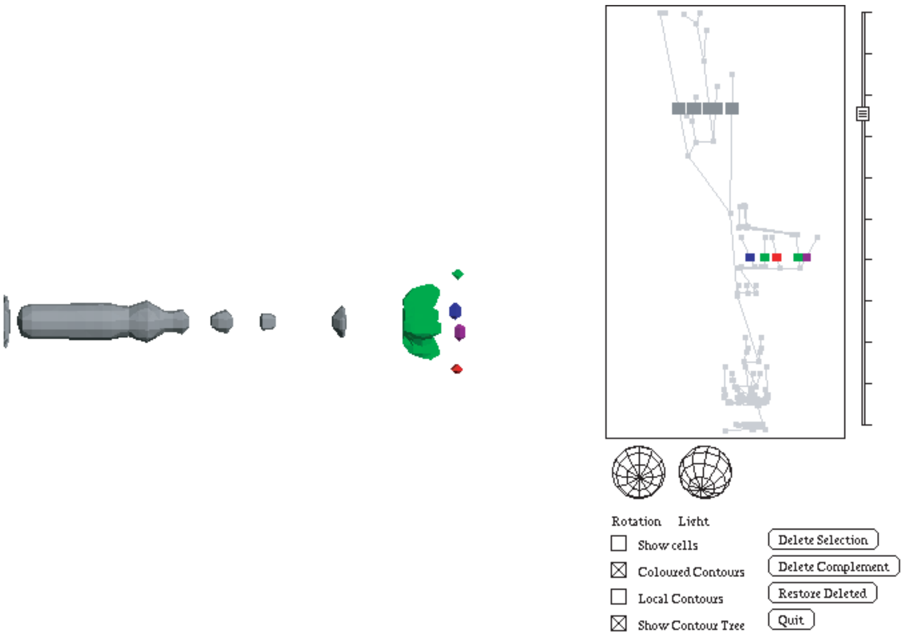



Figure 8: Final Isovalue of the Leftmost Component of Figure 5

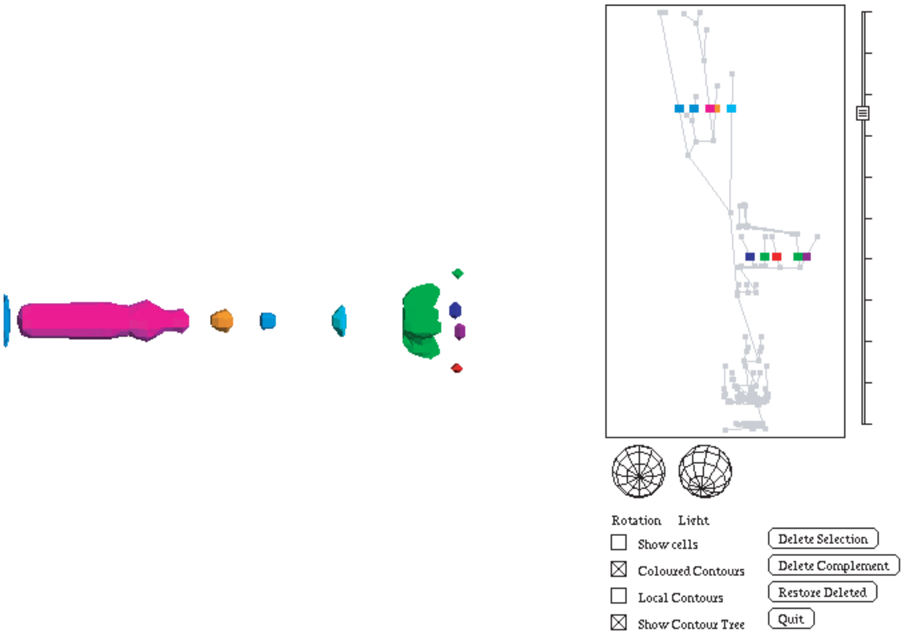



Figure 9: Effect of Unselecting Contours in Figure 8

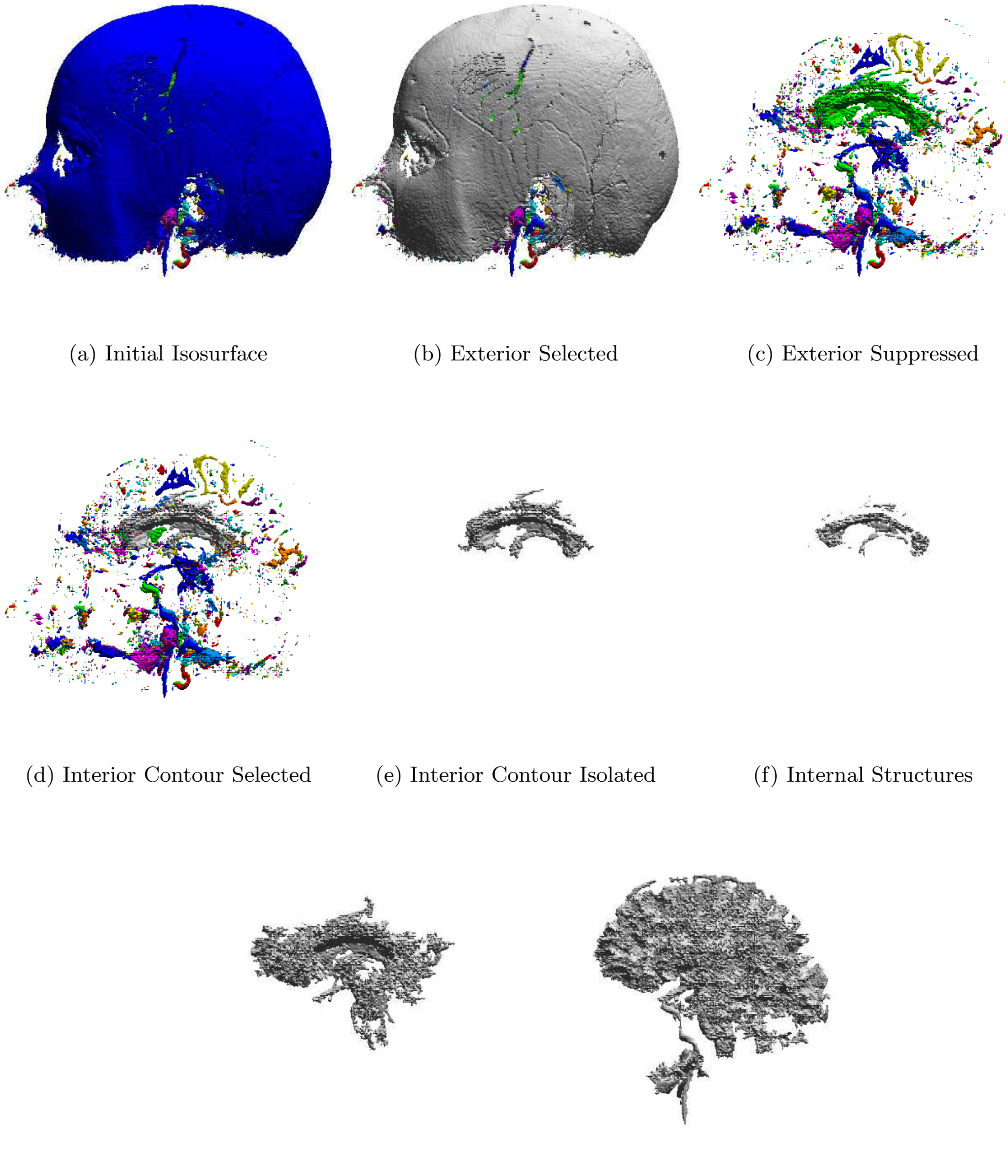
In this example, we start with the isosurface in Figure 10(a). This isosurface has the disadvantage that the exterior contour prevents us from seeing the contours inside. We select the exterior contour, as shown in Figure 10(b), then suppress it in Figure 10(c).

Now that we have removed the exterior, we can see numerous contours inside the head, of which only one interests us: the brain. We select what appears to be the brain in Figure 10(d), then delete all the other contours in Figure 10(e). After we have reduced our field of interest to this single contour, we now adjust its isovalue upwards in Figure 10(f) to see the structures inside it, and adjust the isovalue downwards to see structures outside it in Figure 10(g), and Figure 10(h), in which the brain and spinal cord are visible unimpeded by any other structure.

Third, colours are a form of annotation. We could instead annotate contours with arbitrary information, such as text labels or textures. One particularly useful variant of this is that we store each individual contour in a display list on the video card, then annotate the corresponding edge of the tree with the display list ID. As we manipulate one contour, we only need to update the affected display lists: the remainder are unchanged, reducing the cost of updating the contour display.

Fourth, the scene is further accelerated when we suppress surfaces: in Figure 10, suppressing the exterior contour effectively doubles the frame rate of the display, since roughly half of the primitives no longer need to be drawn.

Although the flexible isosurface interface has the merits of simplicity and a direct correspondence between the contour tree and the surfaces, several problems arise. Choosing a suitable layout can be difficult, because the y coordinate is fixed by the isovalue. In fact, some contour trees cannot be laid out in this fashion without their edges crossing. Figure 11 shows an example of a contour tree that does not have a suitable layout. Even when a suitable layout is possible, larger contour trees



(a) Initial Isosurface (b) Exterior Selected (c) Exterior Suppressed

(d) Interior Contour Selected (e) Interior Contour Isolated (f) Internal Structures

(g) External Structures (h) Brain and Spinal Cord

Figure 10: Removing Unwanted Surfaces and Adjusting the Isovalue of a Desired Surface

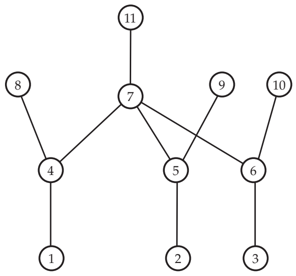



Figure 11: A Contour Tree that is Hard to Display in Two Dimensions

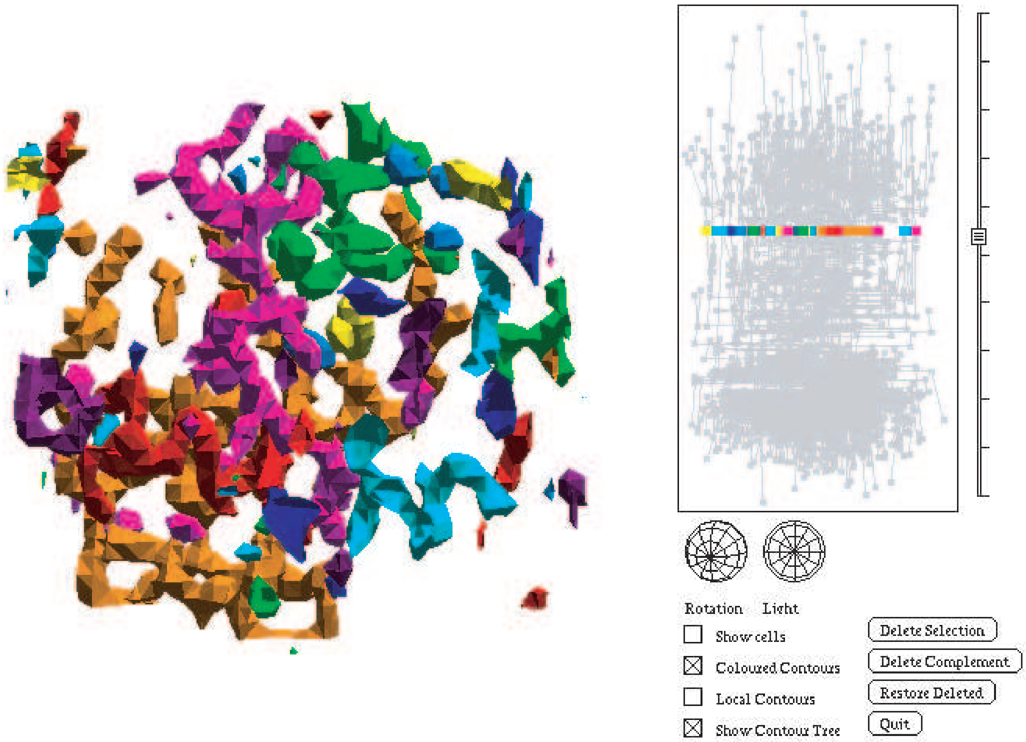



Figure 12: A Molecule with a Complex Contour Tree

become impractical to display due to the sheer number of edges. As the number of edges increases, the contour tree display becomes visually saturated, as in Figure 12, and it becomes difficult to manipulate.

Some of these problems can be addressed by embedding the contour tree in three dimensions rather than two, but this requires that further controls be added to manipulate the contour tree as well as the contour display. It is questionable whether doing so is worth the effort. Instead of this, we note that the operations described above of selection, suppression, and isovalue adjustment can be done directly from the contour display. In fact, we can dispense with the contour tree display, but still use the contour tree behind the scenes to control the display. A better long-term solution is to simplify the contour tree visually: this is under investigation.

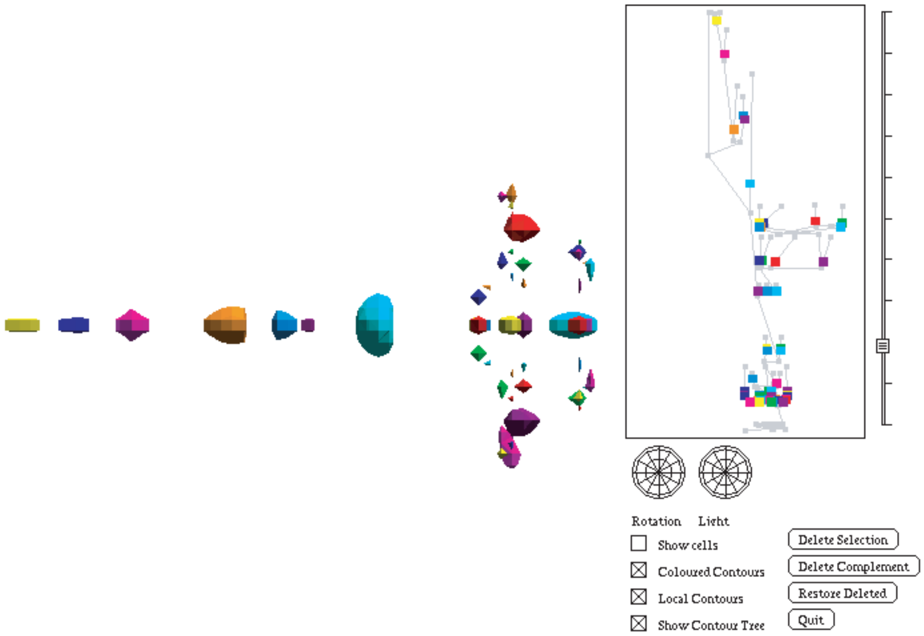



Figure 13: Local Isosurfaces: Simultaneously Isolating All the Local Maxima in Figure 5

## 6 Local Isosurfaces

In some applications, the peaks in the data are more important than specific isovalues. Where this is the case, it is convenient to define an initial flexible isosurface by enclosing each peak (local maximum) in a small surface. Each local maximum is a vertex in the contour tree: no contour can develop past the local maximum in the upward direction. Thus, each local maximum is a vertex at the upper edge of the contour tree - an *upper leaf*. We can define a set of surfaces by cutting each edge leading to an upper leaf. We call this particular flexible isosurface a *local isosurface*, to emphasize the connection to local maxima. An example of such a local isosurface is shown in Figure 13 for the same data set used in Figure 5. Note that, for this data set, it would not be possible to choose a single isosurface that simultaneously identifies all local maxima, as is apparent from the contour tree display to the right.

This local isosurface approach is closely related to the largest contour segmentation of Manders et al. [13], where each peak is surrounded by the largest contour which does not contain another peak. The largest contour segmentation is identical to local isosurfaces, except for one case. If the surface developing downwards from a local maximum splits into two surfaces, as in Figure 1, before it joins up with another surface, then the local isosurface will generate a smaller surface than the largest contour segmentation. Otherwise, the two sets of surfaces will be identical.

## 7 Implementation

As noted in Section 2, algorithms for constructing contour trees have been described by a variety of authors. For the purposes of this paper, we assume that the contour tree has been constructed for the data set using one of these algorithms. We implemented the flexible isosurface interface shown using the algorithm of Carr, Snoeyink & Axen [4]. Since not all contour trees have suitable planar layouts, we laid out the vertices of the contour tree based on the sum of the vertex coordinates, and allowed the user to adjust the horizontal position of the vertices for clarity.

We implemented the user interface shown with the commonly-used GLUT toolkit, and a set of custom interface widgets. Surface rendering was accelerated by storing each extracted contour surface in a separate display list, as described in Section 5.

## 8 Conclusions

We have described how to generate isosurface seeds directly from the contour tree, and how to take advantage of the topology of a scalar field to display multiple contour surfaces simultaneously. We use the contour tree, not only as a cue for interesting isovalues, but as a visual control for the isosurface display, and as a proxy for manipulating sets of partial isosurfaces. We believe that this can provide additional cues and interactivity for exploring volumetric data.

## 9 Future Work

Several extensions are possible, involving improved layouts for the contour tree, simplification of the contour tree, or extension to boundaries extracted by techniques other than isosurfacing.

In Section 5, we noted that not all contour trees have suitable planar layouts. It would be useful to see if a good heuristic exists for reasonable layouts, and also to investigate layouts in three dimensions. In addition, the contour tree interface would be easier to work with if some of the edges in the contour tree were merged or removed. We are currently investigating methods for simplifying the contour tree to ease the task.

The contour tree was originally defined by Boyell & Ruston [3] to express the nesting relationship of a set of contour lines extracted from a map. This definition could be applied to any arbitrary set of nesting boundaries. If a set of surfaces were extracted using some other technique, such as the gradient methods used by Kniss, Kindlmann & Hansen [11], we could construct a contour tree based solely on those surfaces, for manipulation with the interface we have described above.

## 10 Acknowledgements

Acknowledgements are due to the National Science and Engineering Research Council (NSERC) for support in the form of post-graduate fellowships and research grants, and to the Institute for Robotics and Intelligent Systems (IRIS) for research grants. Acknowledgements are also due to Thomas Theußl and Alan Ableson for providing data sets, and to Tamara Munzner for suggesting colour-coded tags.

## A Addendum to Analysis of Contour Tree Algorithm

In Section 4, we claimed that the algorithm of Carr, Snoeyink & Axen [4] took $O(n \log n + N + t\alpha(t))$ time to run. A literal reading of that paper leads to $O(n \log n + N\alpha(n))$ time. This is based on using Tarjan's union-find algorithm [24] to all $n$ vertices and $N$ edges of the mesh.

During the join and split sweeps, most vertices are added to existing components. We modified the union-find algorithm so that it stores the component to which the vertex was added. We then perform union-find representation, merging and path compression with respect to these components.

Adding an edge to the union-find requires two representative lookups: one for each vertex: we charge the cost of accessing the first component to the edge. Since $N$ edges are added to the structure, this is $O(N)$. Thereafter, the path-compression is applied to $O(t)$ components, where $t$ is either the number of local extrema, or the size of the contour tree. This reduces the cost of looking up the union-find representatives to a total of $O(N) + t\alpha t$, giving an overall bound of $O(n \log n + N + \alpha(t))$. For two-dimensional meshes, or regular meshes in higher dimensions, $n = \Theta(N)$, which reduces this to $O(n \log n + t\alpha(t)$. Finally, since $t < n$, we can reduce this to $O(n \log n)$.

Alternately, if we have an irregular mesh in three or more dimensions, with $N \sim n^2$, the cost will be dominated by $N$, the size of the mesh (or the number of edges).

## References

[1] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The Contour Spectrum. In *Proceedings of Visualization 1997*, pages 167–173, 1997.

[2] C. L. Bajaj, V. Pascucci, and D. R. Schikore. Seed Sets and Search Structures for Optimal Isocontour Extraction. Technical Report 99-35, Texas Institute for Computational and Applied Mathematics, Austin, Texas, 1999.

[3] R. L. Boyell and H. Ruston. Hybrid Techniques for Real-time Radar Simulation. In *Proceedings of the 1963 Fall Joint Computer Conference*, pages 445–458. IEEE, 1963.

[4] H. Carr, J. Snoeyink, and U. Axen. Computing Contour Trees in All Dimensions. *Computational Geometry: Theory and Applications*, 24(2):75–94, 2003.

[5] H. Freeman and S. Morse. On Searching A Contour Map for a Given Terrain Elevation Profile. *Journal of the Franklin Institute*, 284(1):1–25, 1967.

[6] N. Gagvani, D. Kenchammana-Hosekote, and D. Silver. Volume Animation using the Skeleton Tree. In *Proceedings of Visualization 1998*, pages 47–53, 1998.

[7] T. Itoh and K. Koyamada. Isosurface Extraction By Using Extrema Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 1:77–83, 1994.

[8] T. Itoh and K. Koyamada. Automatic Isosurface Propagation Using an Extrema Graph and Sorted Boundary Cell Lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1995.

[9] T. Itoh, Y. Yamaguchi, and K. Koyamada. Fast Isosurface Generation Using the Volume Thinning Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):32–46, 2001.

[10] L. Kettner, J. Rossignac, and J. Snoeyink. The Safari Interface for Visualizing Time-Dependent Volume Data Using Iso-surfaces and a Control Plane. *Computational Geometry: Theory and Applications*, page to appear, 2001.

[11] J. Kniss, G. Kindlmann, and C. D. Hansen. Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets. In *Proceedings of Visualization 2001*, pages 255–262,562, 2001.

[12] I. S. Kweon and T. Kanade. Extracting Topographic Terrain Features from Elevation Maps. *CVGIP: Image Understanding*, 59:171–182, 1994.

[13] E. Manders, R. Hoebe, J. Strackee, A. Vossepoel, and J. Aten. Largest Contour Segmentation: A Tool for the Localization of Spots in Confocal Images. *Cytometry*, 23:15–21, 1996.

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

[15] V. Pascucci and K. Cole-McLaughlin. Efficient Computation of the Topology of Level Sets. In *Proceedings of Visualization 2002*, pages 187–194, 2002.

[16] V. Pekar, R. Wiemker, and D. Hempel. Fast Detection of Meaningful Isosurfaces for Volume Data Visualization. In *Proceedings of Visualization 2001*, pages 223–230, 2001.

[17] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing Features and Tracking Their Evolution. *Computer*, 27(7):20–27, 1994.

[18] Y. Shinagawa and T. L. Kunii. Constructing a Reeb Graph Automatically from Cross Sections. *IEEE Computer Graphics and Applications*, 11(6):45–51, 1991.

[19] D. Silver and X. Wang. Volume Tracking. In *Proceedings of Visualization 1996*, pages 157–164, 1996.

[20] D. Silver and X. Wang. Tracking Scalar Features in Unstructured Datasets. In *Proceedings of Visualization 1998*, pages 79–86, 1998.

[21] J. K. Sircar and 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*, pages 171–184, 1986.

[22] S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, and M. Ueda. Algorithms for Extracting Correct Critical Points and Constructing Topological Graphs from Discrete Geographical Elevation Data. *Computer Graphics Forum*, 14(3):C–181–C–192, 1995.

[23] S. P. Tarasov and M. N. Vyalyi. Construction of Contour Trees in 3D in $O(n \log n)$ steps. In *Proceedings of the 14th ACM Symposium on Computational Geometry*, pages 68–75, 1998.

[24] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[25] S. Tenginakai, J. Lee, and R. Machiraju. Salient Iso-Surface Detection with Model-Independent Statistical Signatures. In *Proceedings of Visualization 2001*, pages 231–238, 2001.

[26] M. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. R. Schikore. Contour Trees and Small Seed Sets for Isosurface Traversal. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 212–220, 1997.