Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)
G.-P. Bonneau, S. Hahmann, C. D. Hansen (Editors)

Path Seeds and Flexible Isosurfaces Using Topology for Exploratory Visualization

Hamish Carr,[1] Jack Snoeyink[2]

[1] Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada
[2] Department of Computer Science, University of North Carolina - Chapel Hill, Chapel Hill, North Carolina, USA

Abstract

Morse theory and the Reeb graph give topological summaries of the behaviour of continuous scalar functions. The contour tree augments the Reeb graph for the isosurfaces in a volume to store seed sets*, which are starting points for extracting isosurfaces by the continuation method. We replace the minimal seed sets of van Kreveld et al. with* path seeds*, which generate paths that correspond directly to the individual components of an isosurface. From a path we get exactly one seed per component, which reduces storage and simplifies isosurface extraction. Moreover, the correspondence allows us to extend the contour spectrum of Bajaj et al. to an interface that we call* flexible isosurfaces*, in which individual contours with different isovalues can be displayed, manipulated and annotated. The largest contour segmentation, in which separate surfaces are generated for each local maximum of the field, is a special case of the flexible isosurface.*

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

In other applications, although boundaries are of particular 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[18]. In other cases, user interaction is required to find a suitable isovalue.

Not all applications focus on boundaries: computational fluid dynamics and molecular modelling, for example, 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 that give cues to suitable isovalues can significantly enhance the human's exploration of the data.

In Section 2, we review the interfaces suggested for guiding exploration of data, techniques for automatically selecting isovalues, and relevant work on isosurface extraction, feature tracking, and segmentation. In discussing isosurface extraction, we start with the *continuation* method of Wyvill, McPheeters & Wyvill[32], which divides the function up into polyhedral cells, then follows each contour from cell to cell until the contour has been completely traversed. In order for this method to work, a reliable set of starting points, or *seeds*, is required to initialize the continuation method.

The most reliable source of seed cells is the *contour tree*. This structure is describes the evolution of isosurfaces as the isovalue is varied. Not only can it be used to generate seeds for the continuation method, it can also be used as a visual representation of the data. In Section 3, we describe this structure, review algorithms for constructing it, and review the *minimal seed set* method of van Kreveld et al. for isosurface extraction using the contour tree.

In Section 4, we describe how to replace these minimal seed sets with *path seeds*. Instead of pre-computing and storing a universally sufficient set of seeds, we generate paths of valid seeds from the contour tree at runtime. Not only does this reduce the memory overhead for seed sets, it also reduces the pre-processing required, by integrating the computation directly with the contour tree algorithm of Carr, Snoeyink & Axen[4]. These path seeds also can guarantee a 1-1 correspondence between contours, isosurface seeds, and superarcs of the contour tree.

In Section 5, we exploit this correspondence to identify individual contours, to annotate them, and to track them as the isovalue varies. We extend the contour spectrum of Bajaj, Pascucci & Schikore[1] to create an interface that we call the *flexible isosurface*, in which we select, manipulate, and annotate contours visually. In particular, we free the user from the constraint that a single isovalue must be chosen for the entire data set. Instead, the user may choose several contours at different values. The local contour segmentation of Manders et al.[14] is a special case of the flexible isosurface.

We comment on our implementation in Section 6, state our conclusions in Section 7, and discuss some possible further extensions of this tool in Section 8.

2. Previous Work

We now review previous work on isosurface interfaces[1, 10], interactive transfer function design[11], automatic isovalue selection[18, 30], isosurface extraction[1, 4, 13, 31, 32], feature tracking[20, 23, 24], and segmentation[14].

As noted in the previous section, an isovalue can be specified manually or automatically. Manually specified isovalues are often used to explore otherwise ill-understood data. For this, 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 exploratory process is significantly faster if cues are available to guide the user to interesting isovalues.

In 1991, Shinagawa, Kunii & Kergosien[22] described a method of coding contour changes visually, based on a structure called the Reeb graph. Although this paper described how to code contour changes, it was apparently not used as interface for isosurface exploration. Since the Reeb graph is intimately related to the contour tree, it will be discussed in the following section, along with the contour tree itself.

In 1997, Bajaj, Pascucci & Schikore[1] described an interface for isovalue selection. This interface consisted of two parts: a main panel showing the isosurface, and a separate panel, the *contour spectrum*, which graphed summary characteristics of isosurfaces as a function of the isovalue. These summary characteristics included the surface area and enclosed volume of the isosurface. In addition, the contour spectrum suggested to display the contour tree, as a cue to topological change in the data.

Kettner, Rossignac & Snoeyink[10] modified the contour spectrum in an interface called SAFARI. This interface uses colour and intensity in the contour spectrum panel to represent the connectivity of isosurfaces over two independent variables: time and isovalue. Moreover, the contour spectrum panel is also used directly to specify the time and isovalues to display in the main panel. The connectivity of the isosurfaces at different isovalues and times was computed from the contour trees for the individual time steps, without explicitly extracting the isosurfaces.

A related problem is the design of transfer functions for volume rendering. Transfer functions specify the opacity and optical properties of different types of material. Frequently, transfer functions are based on isovalues in the data. Kniss, Kindlmann & Hansen[11] noted that this assumes that a given isovalue has uniform meaning throughout the data set. They observed that this assumption causes problems, and designed an interface to construct multi-dimensional transfer functions interactively. This interface added gradient information as a parameter to the transfer function.

Instead of user interaction, some authors use statistical methods. Pekar, Wiemker & Hempel[18] use a gradient histogram to find isovalues for which the gradient is steepest, assuming that steep gradients mark significant boundaries. Tenginakai, Lee & Machiraju[30] use statistical signatures of the local distribution of voxel values to define a transfer function. And in a technical report published in 2001, Takahashi, Fujishiro & Takeshima[26] described how to automate transfer function design by using the contour tree to detect isovalues at which major changes in isosurface topology occurred, then emphasizing those isovalues.

Once an isovalue has been chosen, the next task is to extract the corresponding isosurface. In 1987, Lorenson & Cline[13] described *Marching Cubes*, in which the domain of the function is divided into a large number of small cubes. The intersection of the desired isosurface with each cube is then computed separately, and rendered. This approach has the drawback that all cubes are inspected, but only a small fraction of them intersect the surface. Since 1987, many researchers have sought to accelerate isosurface extraction by inspecting only the cubes that intersect the surface.

One of the most efficient method of doing so was described in 1986, by Wyvill *et al*[32]. Instead of inspecting all cubes, they assumed that a cube that intersected each surface was already known. These starting cubes have since come to be known as *isosurface seed cells*. The seed cells for a particular isosurface are placed on a queue. As cells are taken off the queue, their intersection with the isosurface is computed, and, if it is non-empty, all neighbouring cells are placed on the queue. This method has the effect of propagating along each surface until the entire isosurface is constructed. The principal weakness of this method is that it requires a reliable source of seed cells.

Various authors have reported on methods for guarantee-

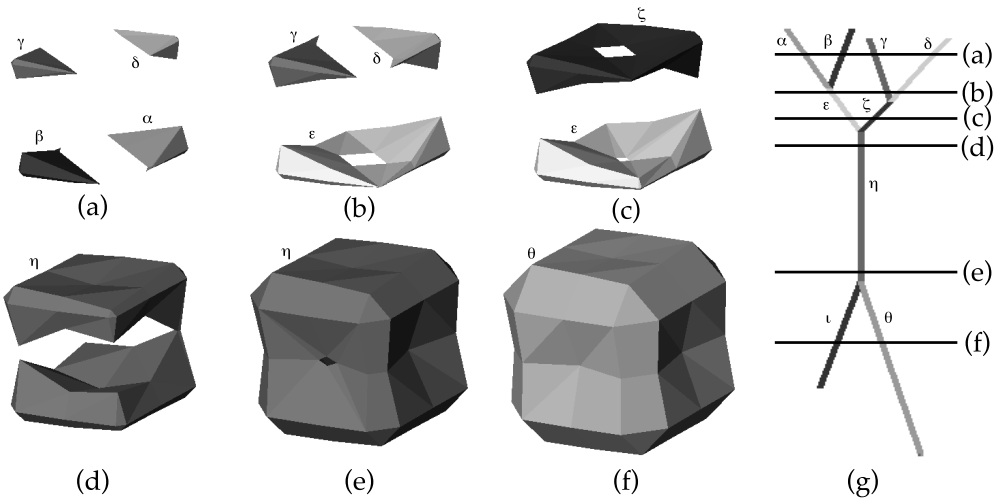

Figure 1: *A Small Example in Three dimensions*

ing such seed cells. Starting in 1994, Itoh & Koyamada[7, 8] described a structure called the *extrema graph*, in which the local extrema in the function were connected by sets of seed cells. They noted, however, that the method was heuristic, and not guaranteed to succeed. Itoh, Yamaguchi & Koyamada[9] extended this to *volume thinning*, in which a set of seeds for extracting contours was constructed by discarding redundant samples until no more could be discarded. This gave a volumetric skeleton for the data set, which also served as a sufficient set of seeds. Similar thinning methods were also used by Bajaj, Pascucci & Schikore[2], to construct the contour tree, and by Gagvani, Kenchammana-Hosekote & Silver[6] to animate volumetric data.

In 1997, Bajaj, Pascucci, & Schikore showed that the contour tree could be used to generate seed sets for contour lines or isosurfaces. We defer discussion of the contour tree until the next section. For now, it suffices to say that the statement that the contour tree captures the relationship between the contours at different isovalues, and can be used to generate seeds for isosurface extraction.

Not all isosurface extraction methods are based on Marching Cubes or continuation. For example, medical data is often acquired as a set of two-dimensional slices. Contours or boundaries are constructed on each slice, and must be connected between slices. Shinagawa & Kunii[21] used the Reeb graph to track the topological changes in the boundaries between slices. A similar approach was also applied to the problem of tracking moving features in time-varying data by Samtaney et al.[20] and by Silver and Wang[23, 24].

In cell biology, Manders et al.[14] use *largest contour segmentation* to detect organelles in individual cells. Instead of looking for sharp boundaries, which were often absent, they define maximal contours 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 5.4, this is a special case of the flexible isosurfaces which we introduce in this paper.

One structure has recurred in many of these areas: the contour tree. This is not a coincidence. Once the decision has been made to study contours, the relationship between the contours is a rich source of information about the data. This information is neatly encapsulated in the contour tree, which we discuss in the following section.

3. The Contour Tree

As noted in the introduction, the *contour tree* is a structure that captures the topological evolution of a data set as the isovalue varies. It is a tree, composed of superarcs and supernodes, each of which may collapse several original edges and vertices. It is best described by showing an example.

Figure 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, while the edge colours correspond to the surface colours. Finally, the Greek letters α to ι mark correspondence between the contour tree and the contours. If we imagine a sweep from high isovalues to low isovalues, we see four small contours in Figure 1(a): these correspond to the four leaves at the top of the contour tree. Contours α and β join to form ε in Figure 1(b): note that ε rapidly becomes a contour of genus 1: this additional information can also be stored in the contour tree[16, 17]. Contours γ and δ then join to form ζ in Figure 1(c). Contours ε and

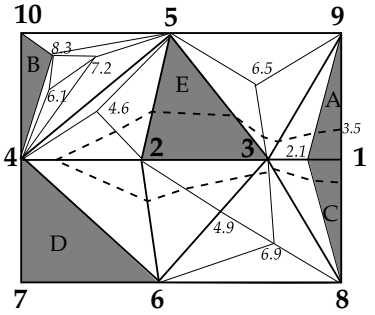

Figure 2: *A Small Sample Triangulation*

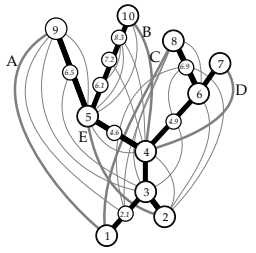

Figure 3: *Graph Matching to extract Seed Sets*

ζ then join to form η, in Figure 1(d). This surface gradually wraps around a hollow core (Figure 1(e)), then pinches off to form two surfaces θ, ι in Figure 1(f). Note that only θ is visible, as ι is contained inside it: we observe a change of colour between Figure 1 (e) and (f), which alerts us to the change in topology. As we sweep to the global minimum value, ι contracts to a local minimum and disappears, followed by θ at the global minimum.

More formally, the contour tree is a special case of the Reeb graph[19]. This graph, first reported by Reeb in 1946, describes the evolution of cross-sections of an arbitrary manifold. We note that every function defines a manifold, and that the cross-sections of that manifold are the contours. Thus, the Reeb graph of a function with respect to the isovalue is the contour tree. Both the Reeb graph and the contour tree are defined in terms of equivalence classes of contours[19, 4]. Note that the contour tree was independently described by Boyell & Ruston[3] in terms of the nesting relationship of a set of explicit polygonal contours.

The best short description of the contour tree is that it is the result of contracting each possible contour to a single point[16]. A contour that passes through a local extremum, or a saddle point at which the connectivity of the contours changes, always contracts to a vertex of the contour tree: the extremum or saddle is called a *critical point*. Otherwise, the contour contracts to a point on one of the edges of the tree, and is called a *regular point*. Although general Reeb graphs may have cycles, the contour tree may not: as is easily seen on a topographic map, contours are always properly nested. Any contour (or point on the contour tree) therefore divides the function domain (and the contour tree) into disjoint pieces. Cycles are therefore excluded, guaranteeing that the contour tree is in fact a tree.

The contour tree was initially used for two-dimensional data sets, as an index structure for extracting contours[3, 31], and as an abstract description of a landscape[5, 12, 25, 27]. Prior to 1995, it was typically constructed from previously extracted polygonal contour lines.

In 1995, Takeshima et al.[27] gave the first algorithm for computing the contour tree for a triangulation in two dimensions. This algorithm traces ascending and descending paths from saddles in the mesh to form a "surface network" connecting all saddles and local extrema in the mesh. Local extrema in the mesh are identified, and transferred to the contour tree, working in to the centre of the tree. This algorithm was extended to three dimensions in 2001 in a technical report by Takahashi, Fujishiro & Takeshima[26], although the contour tree is referred to in this case as the *volume skeleton tree*. No formal analysis is given in either case, but a bound of $O(n^2)$ is not difficult to prove, and a tighter bound may be possible. Special treatment was required for boundary cases, and multiple saddles had to be decomposed into single saddles.

Van Kreveld et al.[31] gave an algorithm for computing the contour tree on simplicial meshes by sweeping a polygonal contour through the mesh from high to low, then from low to high. They also noted that the contour tree could be used to extract isosurfaces from volume data. For two dimensions, the algorithm took $O(n\log(n))$ steps, for three or more dimensions, $O(n^2)$ steps. Again, multiple saddles and boundary cases required special handling. Tarasov & Vyalyi[28] then extended the $O(n\log(n))$ analysis to three dimensions, at the expense of subdividing simplices into as many as 576 cells each. Pascucci[16] further extended this algorithm to track topological genus, as well as connectivity.

Carr, Snoeyink & Axen[4] then extended and improved this algorithm to take $O(n\log(n)+t\alpha(t))$ steps in three and higher dimensions, where t is an output parameter measuring the size of the contour tree. This algorithm combines the sweeps from high-to-low and low-to-high isovalues from van Kreveld et al.with the outside-in construction from Takeshima et al., while dispensing with the explicit contour construction of the former, and the surface network construction of the latter. Moreover, no special cases are required for multiple saddles or boundary cases. Subsequently, Pascucci & Cole-McLaughlin[17] extended the algorithm of Carr, Snoeyink & Axen[4] to track topological genus, and also adapted the algorithm to handle cubic cells with a trilinear interpolation function.

Once the contour tree was constructed, van Kreveld et al. showed how to use it to generate a *minimal seed set*: a set of seed cells that was guaranteed to intersect each

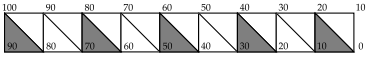

Figure 4: *A Minimal Seed Set of Size $O(n)$*

possible contour at least once. Figure 2 shows a small 2-D triangulation that has the same contour tree as the 3-d example shown in Figure 1. Figure 3 shows how a minimal seed set is constructed for this triangulation. In Figure 3, the contour tree is shown with thick black lines. For each triangle in Figure 2, an edge is added between the highest and lowest-isovalued vertices of the triangle. These edges are shown in Figure 3 as thinner grey edges. A minimal set of edges is then chosen to cover all the original contour tree edges. In this case, one such minimal seed set is the set of five medium-weight grey edges marked A-E. The corresponding triangles A-E in the triangulation are then sufficient to generate all possible contours, and are shown in grey in Figure 2.

Several drawbacks to this approach can be noted. First, although the minimal seed set can be computed in polynomial time, van Kreveld et al.[31] do not specify the polynomial. Instead, they present 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. This approximate algorithm guarantees that no more than twice the minimum number of seed cells are chosen.

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 grey 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 the triangulation in Figure 4. The contour tree in this case consists of a single superarc (i.e. $t = 1$), but the minimal seed set always uses $\Theta(n)$ cells. This construction can be extended to any dimension.

Third, although the contour tree is used to construct the seed set, the correspondence noted between the superarcs of the contour tree, and the individual contours in the image is lost. For each given contour, we may in fact have more than one seed cell. In Figure 2, the contour at a height of 3.5 is shown as a dotted line. Note that it intersects three of the grey triangles: A, C, and E. This many-to-one correspondence makes it difficult to relate contours in the image to the superarcs to which they correspond. This redundant representation makes it necessary to mark which cells have been visited, in order to guarantee that the same contour is not extracted multiple times. This is unnecessary for two-dimensional data sets, which do not need such flags for contour extraction. If we start at the seed cell, we need merely

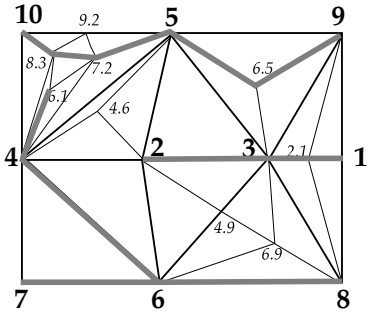

Figure 5: *Monotone Paths as Seeds*

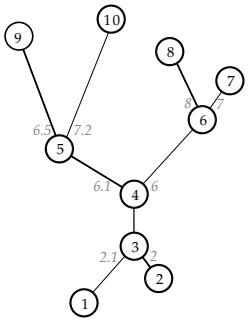

Figure 6: *Storing Monotone Paths as Path Seeds*

follow the contour in either direction until we return to the seed cell, or reach the boundary of the data set. If we reach the boundary, we return to the seed cell and extract the other half-contour travelling in the opposite direction. For three- and higher-dimensional data, however, the number of possible directions to travel requires us to store flags for each cell in the mesh during our extraction step.

In the next section, we describe an improvement to the seed set approach: path seeds, which we generate directly from the contour tree.

4. Path Seeds

The distance is nothing: it is only the first step that is difficult: (the Marquise du Deffand (1697-1780), remarking 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 choose our seed edges so that they

form a set of monotone paths starting at critical points. To do this, we use paths similar to those that Takeshima et al.[27] use to compute the contour tree.

Consider Figure 5, in which the heavy grey edges mark monotone paths starting at critical points. Each path corresponds to a superarc in the contour tree shown in Figure 6. 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.

Unlike the paths used by Takeshima et al.[27], these paths need not extend to a local extremum. For example, one path at 4 ascends to vertex 6.1, then stops. This path is used only for the range of isovalues represented by the corresponding superarc: in this case, $4-5$, and does not need to extend beyond the isovalue 5. Moreover, the paths need not take the steepest ascent (although that will often be the most efficient), nor need they terminate at a critical point. Finally, note that these paths are computed only as needed, and are not used to compute the contour tree itself.

These path seeds are provably minimal, in the sense that no stored seed set can be smaller. At any fork in the contour tree, there must be distinct seeds for each branch. Thus, we must always store at least as many pieces of seed information as there are branches in the tree. Since these path seeds store exactly one piece of information per branch, they are optimal, taking $\Theta(t)$ space to store in the contour tree. Using it to compute contour seeds on the fly does add an additional cost: this can be as much as $\Omega(n)$ for the triangulation in Figure 4. In most cases, 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? Carr, Snoeyink & Axen[4] 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[29] 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: $O(n \log n + t\alpha(t))$. It follows that path seeds are cheaper to store than the minimal seed sets of van Kreveld et al.[31], 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 direct control for manipulating or annotating individual contours.

5. Flexible Isosurfaces

Once we have defined path seeds with a 1-1 correspondence to the contours, we can begin to manipulate contours individually, generating several contours at different isovalues. In order to do so, we extend the contour spectrum of Bajaj, Pascucci & Schikore[1], both to represent these contours abstractly, and to manipulate them.

We define a *flexible isosurface* to be a set of disjoint contours $\{C_1, \ldots, C_m\}$, with isovalues $\{h_1, \ldots, h_m\}$. A conventional isosurface is a special case of the flexible isosurface, consisting of the complete set of contours at isovalue h. In order to display and work with flexible isosurfaces, we must define how to represent the flexible isosurface, how to display it, and how to manipulate it.

5.1. Representing a Flexible Isosurface

In order to represent a flexible isosurface, we take advantage of the fact that each contour in $\{C_1, \ldots, C_m\}$ corresponds to a unique point on the contour tree, and vice versa. Thus, a flexible isosurface can be stored as annotations to the contour tree, marking which superarcs in the contour tree are currently active, and the isovalue of the corresponding contours. Alternately, we can simply store a list of active superarcs and their isovalues.

Even for conventional isosurfaces, this ability to annotate the contour tree is powerful. Not only can we associate an isovalue with a superarc, we can also associate textures, colours, display list IDs, or arbitrary text strings with the superarc. We start by arbitrarily assigning each superarc a colour. The colours used are drawn from a small palette of

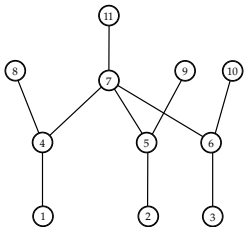

Figure 7: *A Contour Tree that is Hard to Display*

easily distinguished colours. Since the exact colour does not carry semantic information, the choice of palette is immaterial, and colour-blind users can still benefit, by changing the palette of colours. At present, the colour assigned to any given surface is based on the ID number of the superarc modulo the number of colours. In the future, we hope to assign colours based on the structure of the tree, or on domain-specific semantic information.

These colours serve three main purposes. First, they make it much easier to distinguish contours in an image, as in Figure 11 or Figure 12(c), even when they are tightly intertwined. Second, as we will see next, these colours allow us to associate each contour visually with the superarc in the contour tree to which it corresponds. Third, changing the isovalue in such a way that the topology of the surface changes will usually cause the contour to change colour, cuing us visually to changes in the topology.

In addition to colours, we use the superarcs to store OpenGL display list IDs for each active contour. This enables us to avoid regenerating the entire flexible isosurface when we make a change to a single contour in it.

5.2. Displaying a Flexible Isosurface

Once we have identified the active contours, we display the flexible isosurface in two ways. Figure 8 shows the interface, with the contours rendered in the *contour display* on the left, and the contour tree rendered in the *contour spectrum* on the right.

For each contour in the flexible isosurface, we take the path seed stored on the corresponding superarc of the contour tree, and use this to generate a single seed cell for that contour. The contour is then extracted using the continuation method of Wyvill, McPheeters & Wyvill[32], and stored in an OpenGL display list, for efficient rendering. The display list ID is then stored on the superarc for future reference. The contour is then rendered in the colour associated with the superarc.

In the contour spectrum, we render the contour tree. We follow Bajaj, Pascucci & Schikore[1] in fixing the y-coordinate of vertices according to their isovalues. Choosing the x-coordinates is more difficult. In fact, some contour trees cannot be laid out in this fashion without their edges crossing. Figure 7 shows an example of a contour tree that does not have a suitable layout. We then render a coloured tag on each superarc corresponding to an active contour of the flexible isosurface. In Figure 8, for example, there are six active contours labelled α to ζ in the contour display. Visually, since the colours of the tags correspond to the colours of the contours, we can associate the contours on the left with their position in the contour tree on the right, although in this case, both γ and ε have been assigned the colour green.

In this case, the leftmost contour, α has been marked in grey to contrast with the colours assigned to other contours. This indicates that the contour has been *selected*, following the common user interface metaphor of selecting an object, then performing an operation on it. Since we have chosen to use colours to distinguish between contours, we use a distinctive monochrome colour to indicate the selection. Other colour schemes could of course be devised.

5.3. Manipulating the Flexible Isosurface

Now that we know how to define a flexible isosurface, and how to display it, we turn our attention to manipulating it. As we indicated above, we follow the common user interface metaphor of selecting an object, then performing an operation on it. We support the following operations:

1. *Select*. To select a contour, we click the mouse on the desired contour in the contour display, or on the corresponding tag in the contour spectrum. Once we have selected a contour, we display it in a distinctive colour in the contour display and the contour spectrum, as shown in Figure 8.
2. *Unselect*. To unselect contours, we click in any white space.
3. *Add*. To add another contour to the flexible isosurface, we select a previously inactive contour in the contour spectrum: i.e. one with no coloured tag. We then compute the isovalue corresponding to the y-coordinate of the cursor location, add the corresponding contour to the flexible isosurface, and redisplay.
4. *Delete*. One of the buttons below the contour spectrum can be used to delete the current selection from the flexible isosurface: the flag for the corresponding superarc is set to false, and the deleted superarc's ID is temporarily stored in case we wish to reverse the deletion.
5. *Isolate*. The second button below the contour spectrum can be used to isolate a single contour by deleting all contours in the flexible isosurface *except* the selection. Again, the deleted superarcs' IDs are stored in case we wish to reverse the deletion.
6. *Restore*. The third button reverses the last deletion, by transferring superarcs from the deleted list to the flexible isosurface. This is convenient if we wish to temporarily isolate an object to examine it, then return the other contours to the view to give context.

7. *Adjust Isovalue*. We adjust the isovalue of the selection with the vertical slider at the right-hand side of the contour spectrum. This adjusts the isovalue of the selected contour(s). We also incorporate the slider into the contour spectrum itself, by treating the tags in the tree as "thumbs" for the slider. In this mode, instead of dragging the thumb of the slider, we simply select a tag, and drag it vertically to the desired isovalue.
   In Figure 9, we show the result of adjusting the isovalue of the selected contour in Figure 8. As the isovalue increases, the tag slides up the superarc until it reaches a fork, at which point it breaks into two tags, and we render the two corresponding contours in the contour display. As we drag further, more contours appear, and some may disappear, until we reach the final isovalue for the selected contour. Thus, dragging the slider (or tag) with a selection tracks the evolution of the selection as the isovalue is changed.
8. *Select Isosurface*. If no contour is currently selected, dragging the slider at the right-hand side resets the flexible isosurface to a conventional isosurface at the corresponding isovalue.
9. *Select Local Contours*. In some data sets, such as the fuel data set shown in Figures 8, 9, & 10, it is useful to isolate all local maxima simultaneously. Rather than selecting these manually, we provide a check-box marked *local contours*. When this box is checked, and no contour is currently selected, dragging the slider isolates all local maxima, instead of specifying a new isosurface. Since each local maximum is an upper leaf of the contour tree, we add a tag to each upper leaf edge: the slider is used to control the position of the tag on the edge. If the slider is at the midway position, the tag is placed at the midpoint of the edge, and so on. By dragging the slider from top to bottom in this mode, we can see all of the local maxima growing until they reach the maximum size that contains only one critical point. 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.

To give a more complex example, we take the standard UNC "3dhead" MRI data set in Figure 12, and show how to use these operations to isolate the brain and spinal cord, starting from almost any isovalue. In this example, we start with the isosurface in Figure 12(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 12(b), then delete it to see what is inside, in Figure 12(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 12(d), then delete all the other contours in Figure 12(e). After we have reduced our field of interest to this single contour, we can adjust its isovalue. In this case, as we adjust the isovalue downwards, the contour grows, until we reach Figure 12(f), in which the brain and spinal cord are visible unimpeded by any other structure.

In this particular instance, as we delete contours, the display accelerates, as we are no longer generating the missing surfaces, or rendering them to the screen.

We note that, in the example just cited, we do not show the contour tree. This is for three reasons. First, the operations can be performed without having the contour spectrum as a visual reference. Even without showing the contour tree, these operations have immediate meanings that are apparently useful. Second, as the size and complexity of the data set increase, the complexity of the contour tree grows to a point where it saturates the contour spectrum, as in Figure 11. Third, even for small data sets, it is not always possible to draw the contour tree in such a way that no two edges cross, as is shown in Figure 7, above.

This layout problem occurs because we have followed Bajaj, Pascucci & Schikore[1] in insisting that the y-axis encode the isovalue. While we personally prefer this, abandoning this constraint allows alternate representations such as the Morse surface coding of Shinagawa, Kunii & Kergosien[22], planar layout of the tree, or even three-dimensional layout of the tree. We have considered using such alternate layouts, but note that, even when this is possible, larger contour trees become impractical to display due to the sheer number of edges. As the number of edges increases, the contour tree becomes visually saturated, as in Figure 11. For example, in Figure 12, the data set is 256^3 in size, and the contour tree consists of over $1,000,000$ edges, most of which represent noise. In generating these images, we disabled the contour tree, and worked directly with the contour display. A better long-term solution is to simplify the contour tree as Takahashi, Fujishiro & Takeshima[26] have proposed; ways to do this are currently under investigation.

5.4. Largest Contour Segmentation

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. This forms the basis for operation 9. on the flexible isosurface.

This approach is closely related to the largest contour segmentation of Manders et al.[14], where each peak is surrounded by the largest contour which does not contain another peak. In largest contour segmentation, however, 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, and it is clearly possible to modify this local contour extraction to reproduce the largest contour segmentation.

6. 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], modified to use marching cubes instead of tetrahedra. This also entailed modifying the continuation algorithm of Wyvill, McPheeters & Wyvill to propagate strictly along one surface per cell, rather than propagating to all adjacent cubes.

To layout the contour tree in the contour spectrum, we assign x-coordinates by taking the sum of the vertex coordinates in three dimensions, then allow 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. As noted in Section 5, surface rendering was accelerated by storing each extracted contour in a separate display list. Thus, adjusting the isovalue of one contour did not require any other contours to be re-extracted.

Table Table 1 shows the results of using flexible isosurfaces to display and manipulate contours on various data sets, on a dual 1 GHz Macintosh with 1 GB of RAM and a 64 MB GeForce video card, using Mac OS X 10.1.5. Note that the frame rates and path lengths are cited as ranges, as they depend on exactly which contours have been chosen. Of these data sets, f368 is an electron distribution field calculated by Alan Ableson from a molecule in the Protein Data Bank. The second, marlobb, is the test function introduced by Marschner & Lobb [15]. The fuel data set was contributed to volvis.org by the German Research Council, and is computational in nature. The neghip and lobster data sets are also available at volvis.org, and come from the SUNY VolVis distribution. The neghip data is analytic in nature, while the lobster is a CT scan. The hipiph data set is available from the University of North Carolina at Chapel Hill, and is analytic in nature. The teddybear is a data set made available by the University of Erlangen-Nuremberg, while both 3dhead and 3dknee come from the University of North Carolina at Chapel Hill: all three are either MRI or CT scan. The figures for the 3dhead data set come from a run that produced the same images used in Figure 12. Initially, the frame rate was about 0.5 Hz, but this increased once we started deleting surfaces, until it reached a peak of about 1.5 Hz. We acknowledge that these results could, and should be faster, but have not yet optimized the code for maximum frame rate. We also note that the analytic data images tend to be noise-free, and have small contour trees. But, when the contour tree is small, the path lengths tend to be longer than for noisy sampled images, which have large contour trees.

7. Conclusions

We have described how to generate isosurface seeds directly from the contour tree using *path seeds*, and how to take advantage of this to display several contours 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.

8. Future Work

Several extensions to this work 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: this would require a simplified contour tree. Takahashi, Fujishiro & Takeshima[26] have shown how to simplify the contour tree: we are presently working on maintaining seed information efficiently for simplified trees, and methods for using volumetric information to guide the tree simplification.

Interestingly, 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], it should be possible to construct a contour tree based solely on those surfaces, for manipulation with the interface we have described above.

Finally, we would like to use the flexible isosurface interface to construct spatially local transfer functions for volume rendering, in an extension of the work of Takahashi, Fujishiro & Takeshima[26].

9. 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 Tamara Munzner for suggesting colour-coded tags in the contour tree.

References

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

file	*n* samples	time	tree size	longest path	triangle count	frame rate
f368	30,345	0.41s	915	4-6	10K - 17K	15 - 60
marlobb	68,921	1.04s	912	1-23	10K - 35K	24 - 65
fuel	262,144	3.17s	227	2-3	3K - 11K	25 -180
hipiph	262,144	3.80s	1,360	5-29	2K - 22K	15 -150
neghip	262,144	4.50s	2,063	7-10	10K - 30K	20 - 60
lobster	489,600	6.03s	17,867	4	19K - 96K	7 - 32
teddybear	1,015,808	14.80s	245,588	4-5	19K -250K	3 - 21
3dhead	7,143,424	178.05s	2,231,900	2-5	85K -500K	0.5 - 3.5
3dknee	8,323,072	253.93s	2,751,506	1-6	325K -1634K	0.5 -1.64

Table 1: *Results for some selected data sets*

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 Contour Spectra. *Computational Geometry: Theory and Applications*, 25(1-2):97–116, 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. W. E. Lorenson and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, 1987.

14. 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.

15. S. R. Marschner and R. J. Lobb. An Evaluation of Reconstruction Filters for Volume Rendering. In *Proceedings of Visualization 1994*, pages 100–107, 1994.

16. 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.

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

18. 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.

19. 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*, 222:847–849, 1946.

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

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

22. Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien. Surface Coding Based on Morse Theory. *IEEE Computer Graphics and Applications*, 11:66–78, September 1991.

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

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

25. 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.

26. S. Takahashi, I. Fujishiro, and Y. Takeshima. Topological Volume Skeletonization and its Application to Transfer Function Design. Technical Report OCHA-IS 2000-3, Department of Information Sciences, Faculty of Science, Ochanomizu University, Ochanomizu, Japan, February 2001.

27. 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.

28. 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.

29. R. E. Tarjan. Efficiency of a Good But Not Linear Set Union Algorithm. *Journal of the ACM*, 22:215–225, 1975.

30. 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.

31. 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.

32. G. Wyvill, C. McPheeters, and B. Wyvill. Data Structure for Soft Objects. *Visual Computer*, 2:227–234, 1986.

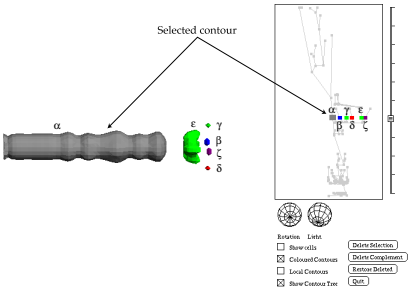

Figure 8: *A Level Set, with Colour-Coded Tags*

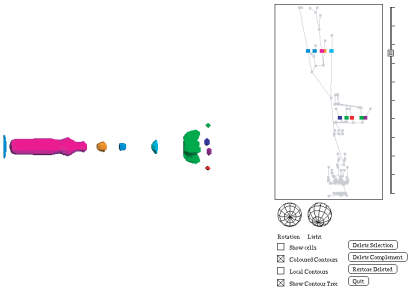

Figure 9: *After Adjusting The Isovalue, and Unselecting*

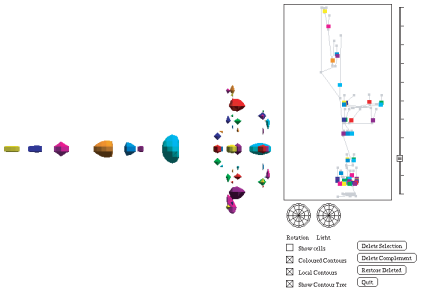

Figure 10: *Isolating Local Maxima*

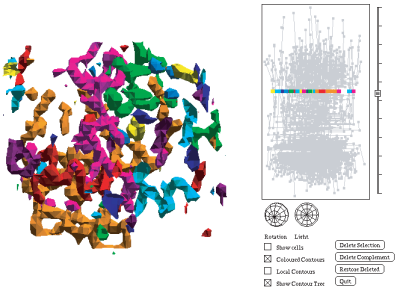

Figure 11: *A Molecule with a Complex Contour Tree*

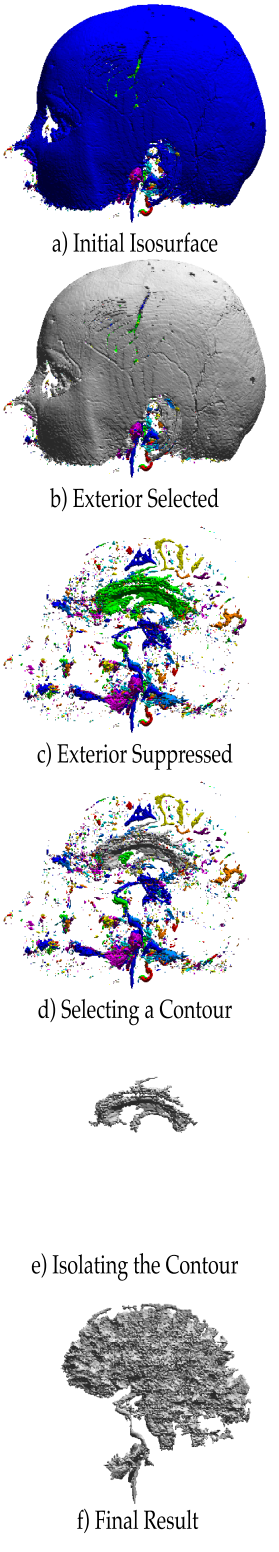

Figure 12: *Flexible Isosurface Operations*