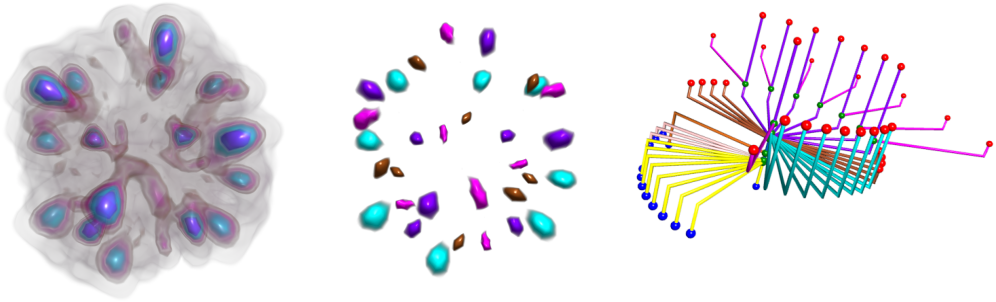
Symmetry in Scalar Field Topology

Dilip Mathew Thomas and Vijay Natarajan, *Member, IEEE*



(a) (b) (c)

Fig. 1. Symmetric patterns identified using contour trees in electron microscopy data of RuBisCO molecule in complex with RuBisCO large subunit methyltransferase (EMDB 1734). (a) Volume rendering of the molecule highlighting repeating structures in the scalar field. (b) Four different types of regions, indicative of the different subunits in the molecule, identified by the symmetry detection algorithm shown in cyan, magenta, brown, and violet. Regions with the same color are symmetric with respect to the scalar field distribution. (c) Subtrees of the contour tree are classified into different groups based on similarity. Subtrees belonging to a common group are shown with the same color and the corresponding regions are identified to be symmetric.

Abstract— Study of symmetric or repeating patterns in scalar fields is important in scientific data analysis because it gives deep insights into the properties of the underlying phenomenon. Though geometric symmetry has been well studied within areas like shape processing, identifying symmetry in scalar fields has remained largely unexplored due to the high computational cost of the associated algorithms. We propose a computationally efficient algorithm for detecting symmetric patterns in a scalar field distribution by analysing the topology of level sets of the scalar field. Our algorithm computes the contour tree of a given scalar field and identifies subtrees that are similar. We define a robust similarity measure for comparing subtrees of the contour tree and use it to group similar subtrees together. Regions of the domain corresponding to subtrees that belong to a common group are extracted and reported to be symmetric. Identifying symmetry in scalar fields finds applications in visualization, data exploration, and feature detection. We describe two applications in detail: symmetry-aware transfer function design and symmetry-aware isosurface extraction.

Index Terms—Scalar field symmetry, contour tree, similarity measure, persistence, isosurface extraction, transfer function design.

1 Introduction

Symmetric patterns are omnipresent in the world around us - be it rotational symmetry in the layout of petals of a flower or symmetric arrangement of atoms in a crystal. Symmetric patterns are used a lot in art, design, and architecture since our sense of aesthetics and beauty are greatly influenced by the presence of symmetry. Detecting and characterizing symmetry is equally important in fields like engineering, biology, chemistry, and physics. Symmetry is important in areas like engineering and manufacturing because it brings about structural and cost efficiencies. For example, study of symmetry is central to the design of mechanical and civil engineering structures. Body structure of many multicellular organisms exhibit symmetry and deviation from it helps in diagnosing abnormalities in the development of body organs. Our understanding about atomic and molecular structures, and chemical reactions and bonding have also been greatly influenced by principles of symmetry.

Symmetry in shapes is important in the area of shape processing for tasks like object recognition and reconstruction, shape matching, segmentation, and shape editing. Symmetry in the context of shapes refers to geometric properties that remain invariant under transformations and is detected and characterized by studying the geometry of shapes. Scientific data is often represented by a scalar field defined on a domain of interest. The scalar field captures scientific measurements or results of simulations, and studying its properties is important for scientific data analysis. Symmetry in scalar fields refers to invariance in the distribution of scalar field within different parts of the domain. As much as study of symmetry in shapes is important in shape processing, study of symmetry in scalar field distribution is important in scientific data analysis because it leads to better understanding of the underlying scientific experiment. For example, consider a bridge that is subjected to a load test to determine its strength. Symmetry in the shape and structure of the pillars is central to the strength of the bridge. The load on the pillars can be measured and represented using a scalar field. Asymmetry in the scalar field distribution is an indication of uneven distribution of load on the pillars. Studying symmetry in scalar field distribution can thus identify structural defects which may otherwise be difficult to detect.

The topology of level sets of a scalar field has been successfully used for studying the behaviour of the scalar field and a natural question that arises is how similar is the topology of level sets in different

- *Dilip Mathew Thomas is with Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India. E-mail: dilip@csa.iisc.ernet.in.*
- *Vijay Natarajan is with Department of Computer Science and Automation, and Supercomputer Education Research Centre, Indian Institute of Science, Bangalore, India. E-mail: vijayn@csa.iisc.ernet.in.*

Manuscript received 31 March 2011; accepted 1 August 2011; posted online 23 October 2011; mailed on 14 October 2011.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.

regions of the domain. In this paper, we study symmetry in scalar fields with respect to the topology of its level sets. We present an algorithm for robust detection of symmetry in scalar fields. Although several algorithms exist for detecting symmetry of shapes, detecting symmetry in scalar fields is not as well studied. We use a topological data structure called contour tree as an abstract representation of the underlying scalar field distribution. Using the contour tree as an abstraction eliminates the need to examine the entire data set for detecting symmetry. The size of a contour tree is typically much smaller than the size of the data from which it is computed. Thus working with contour tree significantly reduces memory and processing requirements and insulates our technique from the size of the data set. Each subtree of the contour tree represents a region of the underlying domain and captures the topology of its level sets. We define symmetry in scalar fields based on similarity of the subtrees in the contour tree, which correspond to regions in the domain with similar level set topology. This leads to an algorithm that detects symmetry by classifying subtrees into different groups. The classification is directed by a similarity measure that compares subtrees based on a topological measure called persistence.

Figure 1(a) shows a volume rendered image of the electron density distribution of RuBisCO protein molecule in complex with RuBisCO large subunit methyltransferase (EMDB 1734). The volume rendering highlights the symmetric or repeating patterns in the scalar field distribution. In Figure 1(b), four different groups of regions identified by our symmetry detection algorithm are shown in cyan, magenta, brown, and violet. Regions with the same color are symmetric. Figure 1(c) shows the simplified contour tree for this data set. Our algorithm classifies subtrees of the contour tree into different groups based on similarity. Subtrees that belong to the same group are shown with the same color in Figure 1(c). We extract regions corresponding to subtrees belonging to the same group and report them to be symmetric.

The main contributions of our paper are the following:

- Modeling the problem of detecting symmetry in scalar fields and computing the symmetric regions as a subtree classification problem in contour trees.
- An algorithm for identifying symmetric regions in a scalar field. The algorithm can detect symmetric patterns at multiple scales and is efficient in terms of memory usage and computational time.
- A comparison measure for identifying similar subtrees of a contour tree. The similarity measure is robust in the presence of noise and uses a compact representation of the contour tree hierarchy, called the hierarchy descriptor, for efficient computation.
- Applications of symmetry identification in scalar fields to visualization, data exploration, and feature detection. In particular, we describe applications to transfer function design and isosurface extraction.

The rest of the paper is organised as follows. Section 2 discusses related work. Section 3 defines the terms used in this paper. Section 4 discusses how contour trees are used in our analysis. Section 5 describes the details of our algorithm. Section 6 discusses results of our implementation. Section 7 shows applications of symmetry detection and Section 8 concludes the paper.

2 Related Work

Symmetry has been primarily studied in the context of geometry processing. Fewer techniques are available for identifying symmetry in scalar fields and these techniques are computationally expensive. In this section, we provide a brief overview of techniques for detecting symmetry in geometric models and in scalar fields. We also describe the importance of contour trees in the analysis and visualization of scientific data sets.

2.1 Symmetry in Geometric Shapes

Automated detection of symmetric patterns is a challenging task since there is no prior information about what symmetric patterns one should look for in an object. Several algorithms exist in literature that detects symmetry in shapes. Early work studied detection of perfect symmetry [1, 47]. Typically, geometric properties of shapes, like location of points, are used to identify axis of symmetry [6, 20, 26]. Perfect symmetry is usually rare and there is a need for robust methods that detect partial and approximate symmetry in the presence of noise. Several techniques have been proposed for robust symmetry detection - voting procedure in a transformation space [24, 28, 34, 36, 48], shape descriptors based on spherical harmonics [23, 25, 38], and graph matching [2, 4]. Change in pose, and non-rigid shapes and transformations make symmetry identification difficult and requires detection of symmetry with respect to non-rigid transformations [8, 27, 31, 37, 48].

Exploiting symmetry information is a well researched topic within geometry processing, computer graphics, and computer vision communities. Symmetry in shapes can be represented using symmetry aware shape descriptors and is used to identify correspondence between models for shape matching and object recognition [23, 36]. Approximate symmetry present in a mesh can be enhanced through remeshing and symmetrization by modifying the mesh in a manner that retains the overall geometry of the mesh [15, 16, 29, 35]. Detecting symmetry helps in decomposition of a model into different parts and can be used for segmentation [36, 40] and to detect redundancies in the model for compression [25, 34, 40]. Symmetry in models is used for scan completion and surface reconstruction by correcting distortions present in noisy and incomplete data acquired through 3D scanners [34, 41].

2.2 Contour Tree for Data Exploration and Visualization

Contour tree is a topological data structure that tracks topology changes in the level sets of a function and has been extensively used in designing tools that assist users in data exploration and visualization [44]. Contour trees enable fast computation of isosurfaces and interactive exploration of large data sets by tracking the evolution of contours [7, 10, 11]. Contour trees partition data into different regions based on the topology of level sets. This property is used in designing spatially aware transfer functions for volume visualization by assigning distinct transfer functions to different branches of the contour tree [46] and automatic transfer function generation by assigning opacity values to branches using a fluid flow model [53]. These methods do not exploit repeating patterns in the scalar field. We demonstrate how these methods can be extended to design a transfer function that explicitly highlights similar regions. Identifying repeating patterns in scalar fields can also be used to enhance applications that use the contour tree for segmentation [7, 21, 42, 49] and shape matching [3, 5, 18, 43].

The lifetime of a level set component can be represented as an interval and has been used to design a barcode shape descriptor [12]. Two families of intervals are compared using a similarity measure that captures the degree of overlap between the intervals. We extend this measure to compare subtrees of a contour tree. Contour trees provide an abstract representation of a scalar field and have been recently used to explore high dimensional data via a two-dimensional terrain representation called the topological landscape [17, 30, 45]. Since our symmetry detection is essentially based on the contour tree, it can be used to identify repeating patterns in any dimension.

2.3 Symmetry in Scalar Fields

Although symmetry in geometric shapes is a topic of research that has been explored in detail, study of symmetry in scalar fields is a relatively new topic. Hong et al. [19] use a parallel algorithm to detect reflective symmetry in 3D scalar fields. Their work extends earlier work on reflective symmetry descriptors for shapes [22, 36] and estimates, for every plane, the distance between the given scalar function and its closest perfect symmetric function. This measure of symmetry is used to detect a plane of symmetry. Their method requires computation of symmetry distance by examining the entire data set for all

planes passing though the volume and is computationally expensive. Our algorithm, on the other hand, requires only an abstract representation of the underlying data to be examined and does not require explicit examination of all planes for symmetry. Further, our method is not restricted to reflective symmetry and can detect repeating patterns in the scalar field at multiple scales.

Bruckner et al. [9] determine similarity between different isosurfaces by computing an information theoretic measure of data independence called mutual information. This method is computationally expensive since it requires calculation of mutual information between all pairs of isosurfaces. Further, this method is restricted to identifying if two isosurfaces are similar and is not suited for extracting regions in the domain with similar scalar field distribution. Schneider et al. analyze the similarity between different scalar fields defined on a common domain using comparison measures based on spatial overlap of contours [39]. While this method studies different scalar fields defined on a common domain, we focus on the detection of repeating patterns within a single scalar field. Several methods for shape matching compare scalar fields using a multi-resolution discrete version of the contour tree and estimate a measure of similarity using geometric, topological, and functional attributes [18, 50, 51, 52]. Though these methods analyse similarity between scalar fields, their goal is to identify the overall similarity between two domains, which is different from our goal of identifying symmetry within a single scalar field. Further, we aim to identify symmetry at multiple scales. We also note that our focus is on identifying similarity in scalar field topology and hence we use a similarity measure based on topological persistence. However, our comparison framework can be used with other attributes also.

3 Definitions

Let $f : \mathbb{M} \to \mathbb{R}$ be a scalar field defined on a simply connected domain $\mathbb{M}$. The set $f^{-1}(\alpha)$, for $\alpha \in \mathbb{R}$, is called a *level set* of f. A *contour* is a connected component of a level set. Consider the equivalence relation defined by contours: $x, y \in \mathbb{M}$ are equivalent if they belong to the same contour. The *contour tree* is the quotient space induced by this equivalence relation. Contour trees track changes in the connectivity of components in level sets when α varies over the range of the function. It is obtained by continuously collapsing each contour in the level set to a single point, see Figure 2(a) - 2(b). Since the domain is simply connected, the contour tree does not contain loops. The nodes of a contour tree correspond to *critical points* of the function. Critical points are further classified as minima, maxima, and saddles. Minima and maxima, also referred to as extrema, correspond to leaf nodes of the contour tree. Saddles correspond to interior nodes where two or more contours merge into one or a single contour splits into many contours. Pairs of critical points represent the evolution of a contour. The difference in function value between such pairs of critical points, called *persistence*, is a measure of the importance of the corresponding topological feature [14].

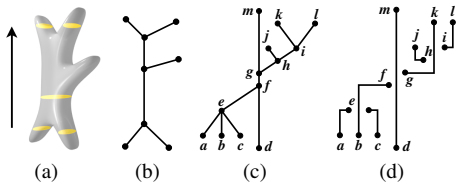


Fig. 2. Contour tree and branch decomposition. (a) Height function defined on a volumetric domain and its level sets and (b) the corresponding contour tree. (c) A contour tree and (d) its branch decomposition representation. The branch dm is the longest monotonic path and hence the root branch. Branches bf and kg are its children.

A *branch decomposition* [32] is an alternate representation of contour trees that is obtained by organizing its edges into a hierarchy based on persistence. A *branch* is a path in the graph with non-decreasing (or non-increasing) value of the function f. The branch decomposition representation is obtained by first assigning the longest branch as the *root branch*. Other branches are attached recursively to the parent branch such that each branch connects one leaf node to an interior node of its parent branch. A branch represents the evolution of a contour from a minimum where it is created to a saddle where it merges into a second contour, or from a saddle to a maximum where it is destroyed. The persistence of the end points (saddle and extremum) is equal to the difference between the function values at these points. Figure 2(c) - 2(d) shows a contour tree and its branch decomposition. The path $dfgm$ in the contour tree is the root branch in the branch decomposition. The paths bef and $ghik$ are child branches of the root branch. Removing all branches whose end points have persistence below a threshold results in a simplified contour tree. Each arc between two nodes in the contour tree represents a set of contours associated with the function values that lie between the corresponding pair of critical points. The subdomain corresponding to a branch is extracted as a union of subdomains corresponding to its constituent arcs and highlighted in a volume rendering using a spatially aware transfer function [46], see Figure 3(a) - 3(d).

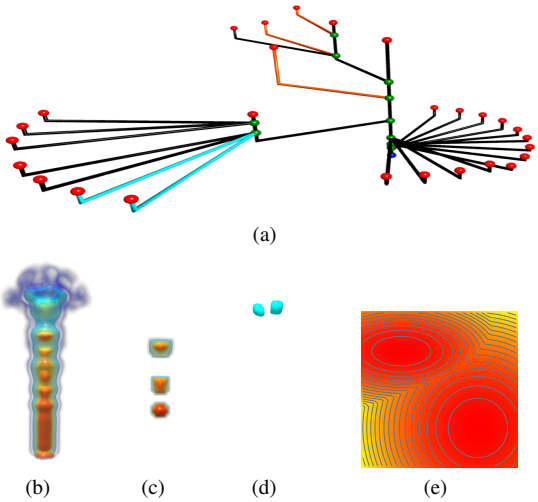


Fig. 3. Each arc in the contour tree represents a region of the underlying domain. (a) Radial layout of the branch decomposition of (b) the Fuel data set. (c) Subvolume corresponding to the orange branches (d) Two isosurface components extracted within the subvolume corresponding to cyan branches. (e) Symmetry in scalar field topology does not necessarily capture symmetry in geometry. Elliptic and circular contours are treated symmetric since they have the same level set topology.

Scalar fields can be considered to be exactly symmetric when their branch decompositions are identical. However, real world data is never perfect and scalar fields often contain noise. Hence, we need a more general definition that captures approximate symmetry. When two branch decompositions are not exact, a natural measure of approximate symmetry is the amount of overlap between their branches. Consider a level set sweep of the domain in the increasing order of function values. Level set components are created at minima, they merge or split at saddles, and are destroyed at maxima. A branch corresponds to the lifetime of a contour and can be represented as an interval whose end points are the function values at the extremum and the saddle end points of the branch. Thus the length of the interval is the persistence of the branch. The *barcode metric* compares two families of such intervals by measuring the extent of overlap between intervals [12]. Our similarity measure is similar to the barcode metric and can, in addition, handle hierarchical relationship among the intervals.

Consider two branch decompositions BD_s and BD_t and a pairing of branches $\{(b_s, b_t)\}$, where $b_s \in BD_s$ and $b_t \in BD_t$. Let I_s and I_t be the intervals corresponding to branches b_s and b_t respectively. The overlap in persistence between b_s and b_t is zero if the extremum of b_s is a minimum and that of b_t is a maximum or vice versa. If the extrema of

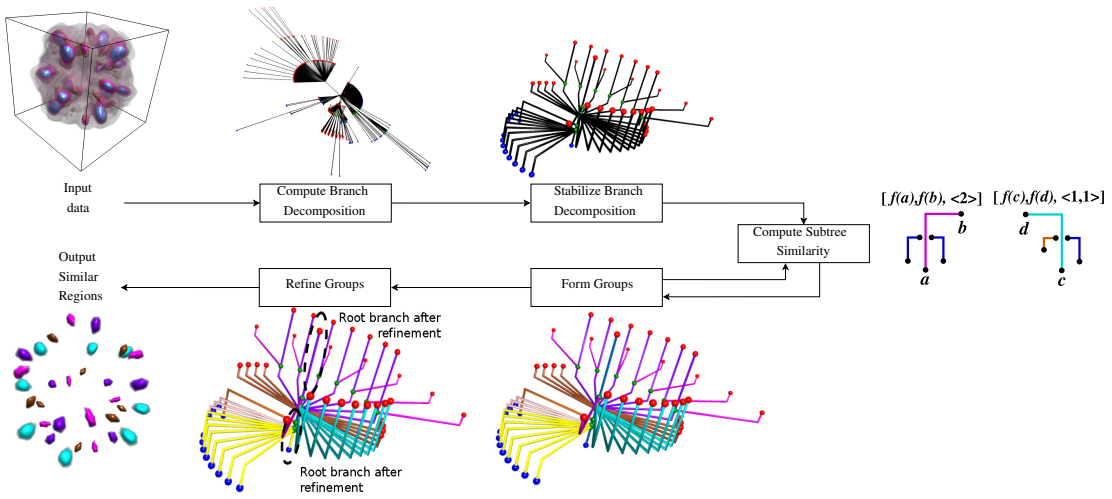


Fig. 4. Symmetry detection pipeline. Branch decomposition of input data is computed and converted to a representation that is stable in the presence of noise. A similarity measure is used to compare subtrees and classify them into different groups based on similarity during a bottom up traversal of the branch hierarchy. Group assignment is then refined and regions belonging to the same group are reported as symmetric regions.

b_s and b_t are both maxima or minima, then the overlap between them is twice the length of the interval $I_s \cap I_t$. Each such pair of branches in the pairing identifies a correspondence between the branches of BD_s and BD_t. To ensure that the pairing is consistent with the branch decomposition hierarchy, we require that each branch can be paired only once and whenever b_s and b_t are paired, the corresponding parent branches are also paired, provided both the parent branches exist. The total overlap in persistence of such a pairing is the sum of overlaps of the paired branches and the percentage of overlap is computed by normalizing the total overlap by sum of persistence of all branches of BD_s and BD_t. We define the similarity between BD_s and BD_t as the percentage of overlap that is maximum over all possible pairings of branches. We define two domains to be *exactly symmetric* with respect to scalar field topology if the percentage of overlap of their branch decompositions is 100%. This corresponds to the branch decompositions being identical. Since our definition of symmetry is purely topological in nature, it may not capture symmetry with respect to the geometry of the level sets, as shown in Figure 3(e).

4 Symmetry using Contour Trees

The topology of level sets of a scalar field in different regions of a domain often show repeating patterns due to similarity in their scalar field distribution. To detect such regions efficiently, our algorithm represents the contour tree hierarchy using a descriptor and uses it to estimate the percentage of overlap between the subtrees of the contour tree.

4.1 Overview

The pipeline used for symmetry identification is shown in Figure 4. In the first step, we compute the branch decomposition representation of the contour tree. Since branch decomposition is not stable in the presence of noise, we generate a stabilised branch decomposition representation. Next, we traverse the branches in the order of increasing persistence. During this bottom up traversal, subtrees are collected together into groups. Each subtree encountered during the traversal is compared with existing groups, using a similarity measure based on persistence. If the subtree is similar to one of the existing groups, then it is added to that group, else it is assigned to a new group. To compare subtrees efficiently, we use the group information to generate a descriptor that compactly represents the branch decomposition hierarchy. Once the group to which each subtree belongs is identified, the subtrees are revisited to refine the group assignment in a post-processing step. After this refinement, all regions of the domain that correspond to subtrees of a given group are reported as symmetric.

4.2 Representing Contour Tree Hierarchy

When two subtrees are compared to determine if they are similar, ideally the entire branch hierarchy should be examined for an accurate similarity estimation. However, since our algorithm frequently compares subtrees, this is computationally expensive. We propose a descriptor, called the *hierarchy descriptor*, as a computationally efficient representation of subtree hierarchy for matching subtrees. Each subtree of the contour is assigned to a group. Each group represents a class of similar subtrees identified by the percentage of overlap which in turn corresponds to regions with similar level set topology. The descriptor uses group assignment of lower level subtrees encountered during the bottom up traversal to generate a vector, called the *frequency vector*. For a given parent branch, we examine the group to which each of its children subtrees belong and the frequency vector is generated based on the number of times each group is encountered. Since the group to which each subtree is assigned to depends on the groups of its children subtrees, the frequency vector representation captures the tree hierarchy of children branches implicitly. Together with the frequency vector, the hierarchy descriptor also stores the extremum and saddle values of the parent branch, resulting in a complete representation of the tree hierarchy of the parent branch.

Consider a subtree rooted at r, whose extremum and saddle values are $r.ext$ and $r.sad$. Let r have n_i children subtrees (possibly zero) belonging to Group i. Then the frequency vector representation of the subtree is $< n_1, n_2, \ldots, n_i, \cdots >$. The hierarchy descriptor of branch r is $[r.ext, r.sad, < n_1, n_2, \ldots, n_i, \cdots >]$. For example, in Figure 5(a), the frequency vector of branches ae and ce is $<>$ since they have no children branches. Let ae and ce be assigned to Group 1. The frequency vector of branch bg is $< 2 >$ because bg has two children belonging to Group 1. Let bg be assigned to Group 2. The frequency vector of branch dl is $< 0, 1 >$ because dl has no child belonging to Group 1 and one child belonging to Group 2. The branches ae, ce, hk and jk in Figure 5(b), share the same extremum and saddle values, say ext_1 and sad_1, and have no children. Hence their hierarchy descriptor is $[ext_1, sad_1, <>]$. Let the extremum and saddle values of branches bg and ig be ext_2 and sad_2. The hierarchy descriptor for bg and ig will be $[ext_2, sad_2, < 2 >]$. The hierarchy descriptor is also used to represent a group, thus making it possible to compare a subtree with a group. When a subtree differs from all existing groups, a new group is created and the hierarchy descriptor of the subtree is used as the descriptor for the group.

Hierarchy descriptors can be used to find exact matches between subtrees by assigning two subtrees to the same group if they have identical descriptors. For example, consider a bottom up traversal of the

branch decomposition in Figure 5(b). Let ae be the first branch encountered and let it be assigned to Group 1. Hierarchy descriptor of Group 1 is equal to that of ae. Branches ce, hk, and jk are identical to ae and hence they have the same hierarchy descriptor as ae. Since the hierarchy descriptors of these branches match with that of Group 1, they are assigned to Group 1. Hierarchy descriptor of bg differs from that of Group 1, hence it is assigned to a new group, Group 2. The descriptor of ig matches with that of Group 2 and hence ig is assigned to Group 2. However, such a matching is not robust - once two branches are assigned to different groups, their parent branches are always assigned to different groups, no matter how similar the parent subtrees are. For example, in Figure 5(c), jk is assigned to Group 1 and hk to Group 2, and hence frequency vector of ig is $< 1,1 >$, which is different from that of bg. So, ig and bg are necessarily assigned to different groups even though they are very similar. We next propose our similarity measure that is robust to small changes in branch hierarchy and assigns subtrees to the same group if the overall hierarchy is similar, but not necessarily identical.

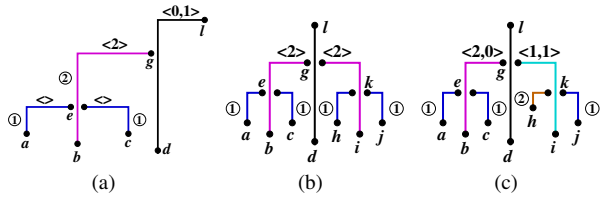


Fig. 5. Groups and frequency vectors. (a) Frequency vector represents the frequency of groups to which children branches are assigned (b) Branches bg and ig match exactly and have the same frequency vector. (c) Extrema of children branches hk and jk are different, hence they belong to different groups, Group 1 and Group 2. Frequency vector of ig differs from that of bg.

4.3 Similarity Measure for Grouping Subtrees

For the exact computation of overlap between subtrees, the maximum overlap over all possible pairing of branches needs to be computed. Since this is computationally expensive, we use a heuristic to estimate the overlap between two branch decompositions. Instead of examining the entire branch hierarchy, we estimate the amount of overlap between the corresponding hierarchy descriptors. The overlap is calculated in two steps. The first step determines the overlap between parent branches. The second step determines the overlap between groups in the two frequency vectors by estimating, for each group in one descriptor, the group that overlaps the most from the other descriptor.

Let the hierarchy descriptor for branches b_s and b_t be

$$H_s = [b_s.ext, b_s.sad, < n_1^{b_s}, \ldots, n_i^{b_s}, \cdots >]$$
$$H_t = [b_t.ext, b_t.sad, < n_1^{b_t}, \ldots, n_i^{b_t}, \cdots >].$$

Let I_s and I_t be the intervals corresponding to b_s and b_t respectively, as described earlier. Overlap between H_s and H_t is zero if the extremum of b_s is a minimum and that of b_t is a maximum or vice versa. Overlap between parent branches can be computed using their interval representation. The overlap between the children groups in the frequency vector of b_s and b_t is estimated as the maximum weight matching of a complete bipartite graph that we construct. The vertices of this graph correspond to groups in the frequency vector of b_s and b_t and its edges are weighted with the overlap between pairs of groups.

Consider a complete bipartite graph, G, where V_1 and V_2 are the set of vertices in the first and second partition, respectively, of the vertex set of G. The vertex sets V_1 and V_2 represent the frequency vectors of H_s and H_t respectively. We construct G with $\sum n_i^{b_s}$ vertices in V_1 and $\sum n_i^{b_t}$ vertices in V_2. Each vertex in the graph represents an occurrence of a group and the overlap between groups in H_s and H_t is computed via a maximum weight matching of G. Consider an edge between a vertex corresponding to Group l in H_s and Group m in H_t. Let H_l and H_m be the hierarchy descriptors for Group l and Group m respectively. The edge is weighted by the overlap between H_l and H_m. Once matching is computed and the overlap between H_s and H_t is estimated, we calculate the percentage of overlap with respect to sum of persistence of all branches in the subtrees rooted at b_s and b_t and use this percentage to determine if H_s and H_t are similar. Intuitively, for each occurrence of a group in H_s, we try to determine which group in H_t is most similar to it. The edges in the matching give the correspondence between groups and the maximum weight matching gives the overall similarity between groups in H_s and H_t. Although we have defined overlap recursively, for efficiency reasons, our implementation uses a table that stores overlap between groups and computes the overlap in an iterative procedure. We summarise the computation of the similarity measure in Algorithm 1.

Algorithm 1 Computing Similarity Measure (branch b_s, branch b_t)

1: Represent branches b_s and b_t using the hierarchy descriptor.
2: Let the overlap between parent branches b_s and b_t be O_1.
3: Construct a complete bipartite graph G with vertices representing groups of children branches of b_s and b_t and edge weights representing their overlap.
4: Compute maximum weight matching of G. Let the weight of this matching be O_2.
5: Let p_s and p_t be the sum of persistence of all branches in the subtrees rooted at b_s and b_t.
6: Similarity measure = $\frac{O_1 + O_2}{p_s + p_t} \cdot 100$.

Consider the branches bg and ig in Figure 5(c). Let $f(x)$ denote the value of the scalar field at vertex x. Let

$$ext_1 = f(a) = f(c) = f(j), \qquad sad_1 = f(e) = f(k),$$
$$ext_2 = f(b) = f(i), \qquad sad_2 = f(g).$$
$$ext_3 = f(h),$$

Overlap between bg and ig is $f(g) - f(b) + f(g) - f(i) = 2(sad_2 - ext_2)$. The bipartite graph construction of bg and ig for matching groups in the frequency vector is shown in Figure 6. The vertices on the left correspond to the frequency vector of bg and has two instances of Group 1, whereas the vertices on the right correspond to ig and has one instance each of Group 1 and Group 2. An edge from Group 1 to Group 2 has weight $2(sad_1 - ext_3)$ and an edge from Group 1 to itself has weight $2(sad_1 - ext_1)$. The maximum weight matching pairs the first instance of Group 1 from bg to Group 1 from ig and the second instance of Group 1 from bg with Group 2 from ig, indicated by the green edges in Figure 6. Thus, the overlap between frequency vectors is $2(sad_1 - ext_1) + 2(sad_1 - ext_3)$ and the total overlap is $2(sad_2 - ext_2) + 2(sad_1 - ext_1) + 2(sad_1 - ext_3)$. The similarity measure correctly identifies that the two hierarchies overlap except for $ext_3 - ext_1$. The percentage of overlap is $\frac{2(sad_2 - ext_2 + sad_1 - ext_1 + sad_1 - ext_3)}{2(sad_2 - ext_2 + sad_1 - ext_1) + (sad_1 - ext_3 + sad_1 - ext_1)} \cdot 100$. Since there is considerable overlap, bg and ig are assigned to the same group. Our similarity measure is robust in the presence of noise because it identifies overlap between the children groups even when children branches are assigned to different groups. Parent branches are assigned to the same group when their hierarchies are similar but not necessarily equal.

5 Algorithm

We now describe the algorithm that groups subtrees of the contour tree. The contour tree is preprocessed to ensure that the group computation is robust in the presence of noise. A post-processing step allows further refinement of groups.

5.1 Stabilising Branch Decomposition Representation

The branch decomposition of a contour tree cannot be directly processed for detecting symmetry because it is highly sensitive to noise in the input scalar field. We pre-process the branch decomposition

through simplification and rearrangement of branch hierarchy to get a more stable representation. Branches whose persistence is less than a small threshold are first removed. A lower threshold helps detect small repeating patterns at the cost of more computation. If the saddle value of a child branch is very close to that of the parent branch, then a small change in saddle values can alter the parent-child relationship. To make the branch hierarchy stable with respect to small perturbations in the scalar field, we move the child branch up the hierarchy and make it a sibling of its erstwhile parent branch. We continue moving the child branch up the hierarchy until the saddle values of the child branch and parent branch are significantly different.

Consider a level set sweep that results in the branch decomposition shown in Figure 7(a). The level sets corresponding to branch c merge with that of branch p and in turn merge with that of branch g. Note that the function value corresponding to the saddle of branch c is lower than that of branch p and this could be an artifact of the noise in the data. If the saddle value of branch c was higher than that of branch p, then branch c would have been a child of branch g instead of branch p. We consider such parent-child relationships to be unstable since small perturbations in the scalar field can change the branch decomposition hierarchy. In Figure 7(a), the topology of level sets of branches c, o and p are similar. However, if this unstable branch decomposition is used, our algorithm will treat them to be different since the subtree rooted at branch p is different from the subtree corresponding to branch o. For robust detection of similar subtrees, we require a stable branch decomposition.

We consider a parent-child relationship to be unstable if the ratio of the difference in the saddle values to the persistence of the parent branch is less than a tunable threshold. Since the branches c and p are unstable, we move the branch c up the hierarchy and make it a child of g, see Figure 7(b). We do this rearrangement for each branch encountered during the bottom up traversal of the branch decomposition. Note that the parent of c may further change while processing branch g if the parent-child relationship of branch c and branch g is unstable. Similarity between the subtrees corresponding to branches c, o, and p can now be identified by our algorithm from the stable branch decomposition in Figure 7(b). A higher threshold for stabilising branches makes the branch hierarchy more stable in the presence of noise and leads to more repeating patterns being detected at the cost of a less faithful representation of the branch decomposition hierarchy. Figure 8(a) shows the branch decomposition of RuBisCO molecule. Simplification removes low persistence branches that appear as flat edges in a radial layout, see Figure 8(b). The stable branch decomposition after changing branch hierarchy is shown in Figure 8(c).

5.2 Comparing Subtrees To Form Groups

We traverse the stable branch decomposition bottom up and assign a group to each subtree encountered during the traversal. Groups are numbered in increasing order beginning with 1. Initially, no groups exist and Group 1 is created from the first leaf branch. A subtree is processed by generating its hierarchy descriptor and comparing it with the hierarchy descriptor of each of the existing groups, as described in Section 4. We determine the group for which the percentage of overlap is maximum and if this overlap is greater than a threshold, we assign the subtree to the group. If the percentage of overlap is

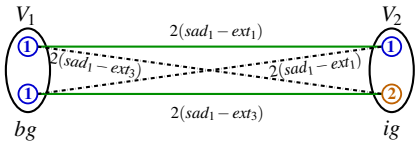


Fig. 6. A complete bipartite graph is constructed to match groups in the frequency vectors of bg and ig from Figure 5(c). A vertex is created for each occurrence of a group and edges are weighted with the amount of overlap between the groups. The total overlap between the frequency vectors is defined to be the maximum weight matching of the graph, shown in green.

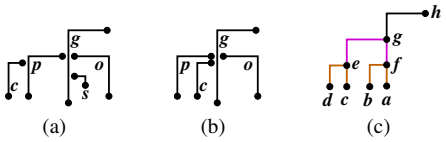


Fig. 7. Pre-processing and post-processing of branch decomposition. (a) Parent-child relationship of branch c and branch p is not stable. (b) We stabilize the branch decomposition by changing the parent-child relationship. Branch c is moved up the hierarchy to become a sibling of p and a child of g. The low persistence branch s is also removed. (c) Subtree rooted at branch ah is split further to identify similar subtrees. Subtree rooted at partial branch ag is similar to cg and the partial branches af and ce are similar to bf and de.

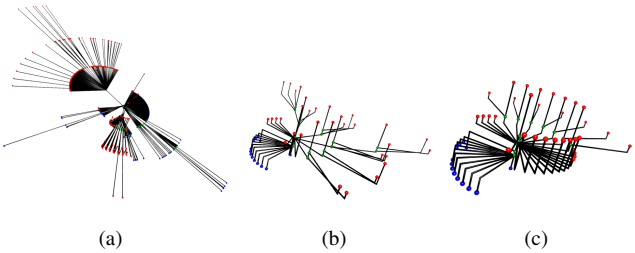


Fig. 8. Simplifying and stabilising the branch decomposition. (a) Branch decomposition of RuBisCO molecule. (b) Simplification removes low persistence branches. (c) Unstable branches are made stable by changing parent-child relationship.

lower than the threshold, we create a new group, assign the subtree to this group, and store the hierarchy descriptor of the subtree as the hierarchy descriptor for the group. Figure 9(a) shows the assignment of subtrees to different groups. The hierarchy descriptor of a subtree is compared with the hierarchy descriptor of all existing groups. When a new group is created, the overlap of the group with each existing group is stored in a table and used for determining the weight of edges during the bipartite graph construction.

5.3 Refining Groups

Groups to which subtrees are assigned to during the bottom up traversal of branches require further refinement before symmetric regions can be identified. Subtrees are assigned to groups on-the-fly during the bottom up traversal. For a given subtree, it is possible that a group created later during the traversal is a better match than what it was initially assigned to. Since all groups are created after the bottom up traversal, we fix this issue by traversing the branch hierarchy again, and assigning each subtree to the best group by comparing the hierarchy descriptor of each subtree with that of all groups.

During the refinement stage, each subtree is assigned to a single group. However, parts of a subtree may better match other subtrees.

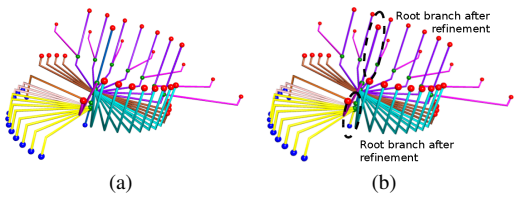


Fig. 9. Computing and refining groups of subtrees of the branch decomposition. (a) Subtrees in the stable branch decomposition are classified into different groups based on similarity between the hierarchy descriptors. Branches with the same color belong to the same group. (b) The root branch is split and is assigned to groups corresponding to magenta and yellow branches.

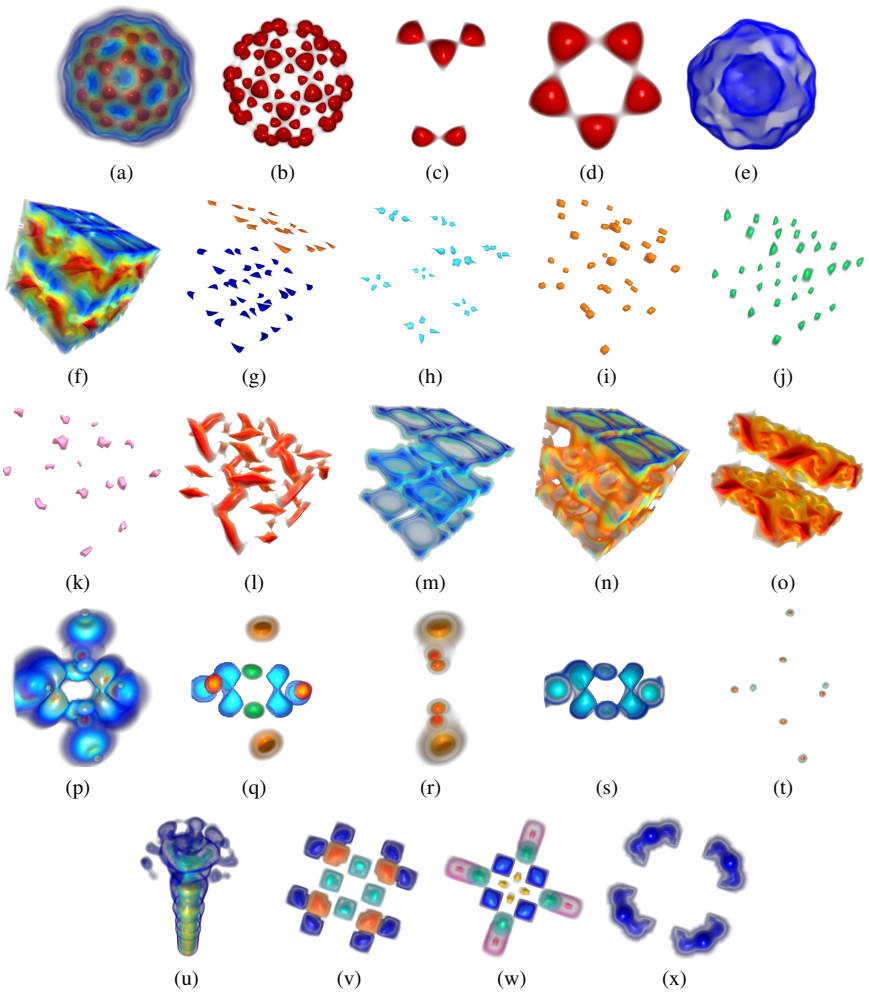


Fig. 10. Symmetric regions identified within four data sets - Buckyball (first row), Vortex simulation (second and third row), Neghip (fourth row), and Fuel (fifth row).

Consider branch ah in Figure 7(c). The branch hierarchy of branch ah has several repeating parts - the subtree rooted at branch cg is similar to the subtree rooted at the partial branch afg and its child bf. Similarly, the partial branch af and ce match with bf and de. Hence, if bf and de are assigned to one group then the partial branches af and ce also should be assigned to the same group. Similarly, the subtree rooted at the partial branch ag should be assigned to the same group as the subtree rooted at cg. In order to identify all repeating regions corresponding to a group, each subtree is analysed to detect if a part of it is similar to the group. We sweep each subtree from extrema to saddle and generate the hierarchy descriptor for the partial subtree encountered. If the partial subtree is similar to a group, then the partial subtree is assigned to that group and the sweep continues. Since the root branch has two extrema it requires two sweeps, in the direction of increasing and decreasing function values. The upper and lower end of the root branch in Figure 9(a) is split and regrouped as shown in Figure 9(b). Now, for each group, all similar subtrees belonging to the group are identified and the region of the domain corresponding to each subtree is extracted and reported to be symmetric.

6 Results and Discussion

We now present results from our analysis of the performance of the symmetry detection algorithm. We also describe our extensive experiments on several data sets that demonstrate the wide applicability of the symmetry detection algorithm. Branches whose persistence is less than 1% of the root branch are simplified. A branch is considered to be unstable if the difference in saddle values of parent and child branches is less than 1% of the persistence of the parent branch. Subtrees are assigned to a group if the similarity between the corresponding hierarchy descriptors is more than 90%. The above mentioned parameter values are used uniformly for all data sets. The threshold for simplification and stabilisation depends on the noise in the data and we estimate this by plotting the number of branches in the contour tree against increasing simplification ratio as shown in the supplemental material. Initially, there is a sudden drop in the number of branches as the low persistence branches corresponding to noise in the data are removed. The simplification ratio and stabilisation ratio that we choose corresponds to the value at which the initial drop in the number of branches tapers off. We identify symmetry at different scales - regions that correspond to leaf branches are at the smallest scale and larger scale symmetric regions correspond to subtrees that are higher in the branch decomposition hierarchy.

Table 1. Time taken to group subtrees of the contour tree for various data sets. All experiments were performed on a workstation with a 2 GHz Intel Xeon processor.

Data set	vertices	branches	simplified branches	groups	time (sec)	memory (MB)
Buckyball	509^3	309693	61	8	0.18	8.51
Vortex	64^3	1014	321	21	0.41	0.20
Neghip	505^3	24437	112	51	0.20	0.70
Fuel	505^3	2788	45	19	0.03	0.09
RuBisCO	80^3	41150	1703	126	27.7	6.67

6.1 Performance Analysis

The computational cost of our algorithm is dominated by the time spent in the bottom up traversal to identify similar subtrees. During this traversal, each subtree is compared with all existing groups. Since each subtree can be assigned to a different group, the maximum number of groups can be as high as the number of branches in the branch decomposition. Maximum weight matching for a bipartite graph can be computed in $O(n^2 + mn \log n)$ time, where m and n denote number of edges and vertices respectively. The algorithm works on a complete bipartite graph where the vertices correspond to groups. Let t be the number of branches, then m is quadratic in t and hence matching can be computed in $O(t^2 + t^3 \log t)$ time. The number of comparisons done in the bottom up traversal is quadratic in t, hence the worst case running time of our algorithm is $O(t^5 \log t)$. Note that t is much smaller than the size of the input data set and hence our algorithm performs well in practice. Table 1 shows the time taken and memory used for grouping subtrees of the contour tree. We assume that the contour tree is available as input. The contour tree of a scalar field can be computed efficiently in terms of computational time and memory usage [13, 33]. Since our method only examines the contour tree, it is insulated from the size of the data. The time taken and memory used by our implementation depends only on the size of the contour tree and the number of groups identified.

6.2 Visualization

Figure 10(a) shows the Buckyball data set that contains sixty carbon atoms shown in red. Each maximum corresponds to a carbon atom. We identify the scalar field distribution of the spatial region corresponding to each of these atoms to be symmetric. The extracted regions are shown in Figure 10(b). With our default threshold of 1% for identifying unstable branches, we see that some of the atoms merge together in groups of two while others merge in groups of three as shown in Figure 10(c). Increasing the threshold to 2% results in five carbon atoms merging together to form a ring as shown in Figure 10(d) and we detect twelve symmetric rings. If the threshold is increased to 8% then all children branches merge with the root branch and we do not identify any relationship among the carbon atoms with respect to the way they merge. The contour trees corresponding to these thresholds are shown in supplemental material. Figure 10(e) shows two regions grouped together, one of which is occluded by a spherical envelope. Though the geometry of these regions are different, we report them to be symmetric since the scalar field distribution of these regions are similar. The corresponding branch decomposition is included as a figure in the supplemental material.

Figure 10(f) shows a volume rendering of the Vortex data set, which shows the temperature distribution in a vortex flow. There are several repeating patterns at different scales. Figures 10(g) - 10(k) show repetitions at small scales, where as Figures 10(l) - 10(m) show repetitions at a larger scale, and Figures 10(n)- 10(o) show the symmetric regions at the largest scale. Though the regions shown in Figure 10(g) are symmetric geometrically, they are classified into two groups since the difference in scalar field distribution is significant at the smallest scale. However, the similarity measure is robust and correctly identifies that at larger scales the difference is not significant and these regions are grouped together as shown in Figure 10(m).

Figure 10(p) shows the Neghip data set and reflective symmetry present in it. Figure 10(q) shows four different reflective symmetric regions identified by our algorithm. Figures 10(r) - 10(s) show symmetric regions at larger scales. We also identify eight small symmetric regions as shown in Figure 10(t). Figure 10(u) shows the Fuel data set which is devoid of symmetric regions in the subvolume corresponding to the shaft. We identify different symmetric regions in the subvolume corresponding to the crown as shown in Figures 10(v) - 10(w). All regions repeat four times except the blue region in Figure 10(v) which repeats eight times. The blue regions merge to form a larger symmetric region shown in Figure 10(x). Figure 1(b) shows four symmetric regions in RuBisCO data set identified using simplification and stabilization threshold of 7%. A higher simplification ratio was used to reduce the number of groups for ease of illustration.

6.3 Limitations

The main limitation of our method is that symmetry detection is solely based on the structure of contour trees and our method fails when symmetric regions do not manifest as repeating subtrees of the contour tree. Hence, our method does not perform well if the scalar field is noisy and branches of the contour tree corresponding to noise have high persistence or when the scalar field has large flat regions. The hierarchy descriptor and the similarity measure we use for subtree matching is a good estimate but not as accurate as examining the complete branch hierarchy. Further our method ignores the geometry of repeating regions. Hence it is possible that regions with different geometry are grouped together and regions with similar geometry are grouped differently. These limitations present interesting challenges that we plan to address in future work.

7 Applications

We utilize the symmetric patterns to enhance the classical visualization techniques, volume rendering and isosurface extraction, and to facilitate data exploration. The data used to illustrate all applications is a 3D Taylor-Green vortex flow simulation on a Cartesian grid.

7.1 Symmetry-aware Transfer Function Design

A good transfer function assigns color and opacity values to each point of a volume in a manner that highlights important features of the volume. Designing good transfer functions is a challenging problem in visualization. Transfer functions that are typically used assign color and opacity values based on scalar field value at each point of the volume. Such transfer functions will not be able to highlight different features of the volume if they belong to the same range of function values. To overcome this problem, contour trees have been used to design spatially aware transfer functions by assigning different transfer functions to different branches of the contour tree [46, 53]. However, these methods do not detect repeating patterns in the domain while assigning different transfer functions. Our method can be used to classify regions into different groups based on symmetry. Distinct regions belong to different groups and can be assigned different transfer functions while repeating regions within the same group can be assigned the same transfer function, thus highlighting repeating patterns in the domain.

Figure 11(a) shows a volume rendered image of the 3D Taylor-Green vortex flow data. Rendering the entire volume to show all features often results in visual clutter. We identify symmetric regions in the domain and assign different transfer functions to different patterns in the scalar field distribution, as shown in Figure 11(b). These repeating regions can also be examined in isolation, see Figures 11(c) - 11(f). Classification of regions of the domain into different groups allows the user to examine similar regions of interest in the context of the rest of the volume while remaining unobstructed by other features in the volume. We achieve this by assigning a high opacity transfer function to regions that belong to the group and a low opacity transfer function to the rest of the volume, see Figures 11(g) - 11(h). Similarly, it is possible to hide repeating occurrences of a feature, thereby allowing the user to focus on the unique features in the volume.

7.2 Symmetry-aware Isosurface Extraction

Volumetric data is often analysed by extracting different isosurfaces and studying their properties. The number of components in the isosurface for a given isovalue can be quite large. Data exploration via isosurfaces often becomes difficult because the larger components may occlude the smaller components. For instance, if an inner component is nested within a bigger outer component, then a user may miss the inner component as it is completely hidden. Users could use cut-section views to clearly see the components of interest, see Figures 12(a) - 12(b). Such steps often hinder the data exploration process since users have to manually hide parts of the isosurface that are not of interest.

Our classification of regions into different groups can be used for effective isosurface extraction. Each component of an isosurface belongs to a unique branch of the contour tree. We use the group information of the branches of the contour tree to classify the different

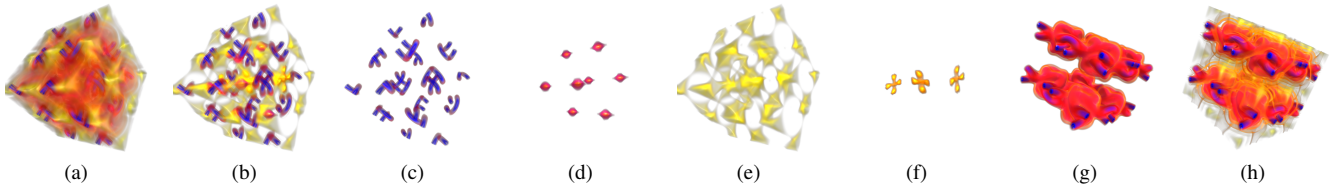


Fig. 11. Symmetry-aware transfer function design. (a) Volume rendering of a 3D Taylor-Green vortex flow simulation data set. Traditional volume rendering makes it difficult to distinctly highlight different features belonging to the same function range. (b) Symmetric or repeating patterns are detected and highlighted by using a distinct transfer function for each symmetric group. (c) - (f) All occurrences of a single pattern can be examined in isolation. (g) - (h) A pattern can be examined in the context of the rest of the volume without being obstructed by other features of the domain.

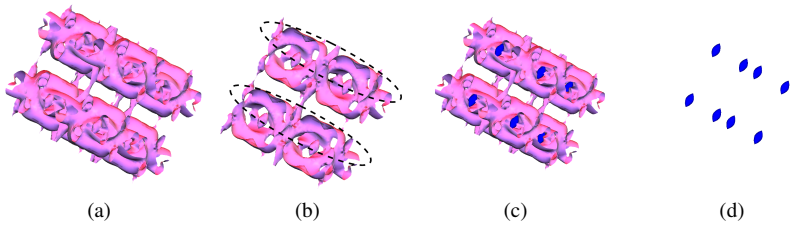


Fig. 12. Symmetry-aware isosurface extraction. (a) Oval-shaped isosurface components are occluded by the surrounding isosurface components. (b) Cut-section view of the isosurface that shows some of the oval components. (c) Group information computed by our method can be used to highlight the oval-shaped components in blue, or (d) Hide rest of the components that occlude the oval-shaped components.

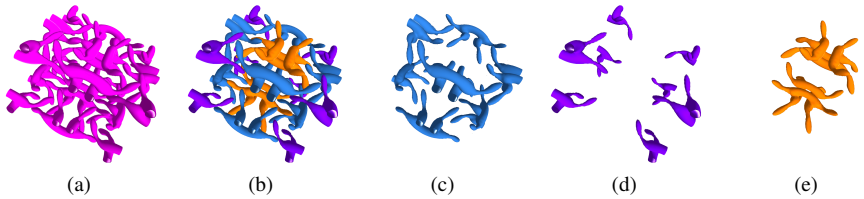


Fig. 13. Symmetry-aware isosurface extraction. (a) Traditional isosurface visualization extracts all isosurface components for a given isovalue without differentiating between the various isosurface components. (b) Isosurface components are classified into different groups. Components that are similar are shown with the same color. (c) - (e) Tools can be designed to allow the user to select a component and automatically show or hide similar components.

components of a given isosurface. Highlighting the isosurface components based on the group to which they belong can avoid problems of occlusion and visual clutter while visualizing them, see Figure 13.

Our method can also be used for designing user interface tools that can aid users in interacting with isosurfaces. For example, a user can select an isosurface component and query for all components that are similar to the selected component. All the selected components could then be optionally shown or hidden. Such tools are popular in image manipulation software and extending them to isosurface visualization will help users effectively explore their data. Attention can be easily directed towards components of interest without having to navigate through components that are not relevant.

8 Conclusions and Future Work

We have described a method for detecting symmetry in scalar field topology. Our method analyses the contour tree of a data set and uses a similarity measure based on persistence to group together similar subtrees of the contour tree. Symmetric regions are then extracted from subtrees belonging to the same group. Our method is efficient in terms of computational time and memory usage. We show applications of scalar field symmetry detection to transfer function design and isosurface extraction. We believe that our method will find several more applications in areas related to data exploration and visualization. A possible application is to automate the process of identifying the presence of a predefined feature in a dataset. The contour tree for the predefined feature can be generated and our similarity measure can be used to identify if the feature is present by comparing each subtree of the contour tree of the input dataset with the contour tree corresponding to the predefined feature. Such an automated matching can be quite beneficial when manual analysis is tedious as in the case of identifying all occurrences of a given pattern or searching through a repository of data sets for the occurrence of a predefined pattern.

Since our method for symmetry detection is based on contour trees, it fails to detect symmetric patterns in the domain that do not manifest as similar subtrees in the branch decomposition hierarchy. Another limitation of our method is that it is restricted to identifying regions in the domain with similar level set topology and is insensitive to its geometry. In such cases, an alternate representation of the scalar field distribution may be able to detect symmetric patterns better. Identifying symmetry in the geometry of the level sets is a direction for future work and appears to be challenging. We are currently investigating alternate methods for identifying symmetry based on a combination of geometric and topological properties of the scalar field.

Acknowledgments

This work was supported by the Department of Science and Technology, India, under Grant SR/S3/EECE/048/2007. The authors wish to thank Ratnesh Shukla for the Vortex flow datasets. Volume rendered images used in the paper were generated using Voreen (www.voreen.org).

References

[1] M. J. Atallah. On symmetry detection. *IEEE Trans. Computers*, 34(7):663–666, 1985.

[2] A. Berner, M. Bokeloh, M. Wand, A. Schilling, and H.-P. Seidel. A graph-based approach to symmetry detection. In *Proc. Symposium on Volume and Point-Based Graphics*, pages 1–8, 2008.

[3] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theor. Comput. Sci.*, 392(1-3):5–22, 2008.

[4] S. Biasotti and S. Marini. Sub-part correspondence using structure and geometry. In *Proc. Eurographics Italian Chapter Conference*, pages 23–28, 2006.

[5] S. Biasotti, S. Marini, M. Spagnuolo, and B. Falcidieno. Sub-part correspondence by structural descriptors of 3d shapes. *Computer-Aided Design*, 38(9):1002–1019, 2006.

[6] P. Braß and C. Knauer. Testing congruence and symmetry for general 3-dimensional objects. *Comput. Geom.*, 27(1):3–11, 2004.

[7] P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. S. Day, and J. B. Bell. A topological framework for the interactive exploration of large scale turbulent combustion. In *Proc. IEEE International Conference on e-Science*, pages 247–254, 2009.

[8] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Topology-invariant similarity of nonrigid shapes. *International Journal of Computer Vision*, 81(3):281–301, 2009.

[9] S. Bruckner and T. Möller. Isosurface similarity maps. *Comput. Graph. Forum*, 29(3):773–782, 2010.

[10] H. Carr and J. Snoeyink. Path seeds and flexible isosurfaces using topology for exploratory visualization. In *Proc. Symposium on Data visualisation*, pages 49–58, 2003.

[11] H. Carr, J. Snoeyink, and M. van de Panne. Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. *Comput. Geom.*, 43(1):42–58, 2010.

[12] A. D. Collins, A. Zomorodian, G. Carlsson, and L. J. Guibas. A barcode shape descriptor for curve point cloud data. *Computers & Graphics*, 28(6):881–894, 2004.

[13] H. Doraiswamy and V. Natarajan. Output-sensitive construction of reeb graphs. *IEEE Trans. Vis. Comput. Graph.*, 99(PrePrints), 2011.

[14] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[15] D. Ghosh, N. Amenta, and M. M. Kazhdan. Closed-form blending of local symmetries. *Comput. Graph. Forum*, 29(5):1681–1688, 2010.

[16] A. Golovinskiy, J. Podolak, and T. A. Funkhouser. Symmetry-aware mesh processing. In *Proc. IMA Conference on the Mathematics of Surfaces*, pages 170–188, 2009.

[17] W. Harvey and Y. Wang. Topological landscape ensembles for visualization of scalar-valued functions. *Comput. Graph. Forum*, 29(3):993–1002, 2010.

[18] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH*, pages 203–212, 2001.

[19] Y. Hong and H.-W. Shen. Parallel reflective symmetry transformation for volume data. *Computers & Graphics*, 32(1):41–54, 2008.

[20] X. Jiang and H. Bunke. Determination of the symmetries of polyhedra and an application to object recognition. In *Workshop on Computational Geometry*, pages 113–121, 1991.

[21] G. Johansson, K. Museth, and H. Carr. Flexible and topologically localized segmentation. In *EuroVis*, pages 179–186, 2007.

[22] M. M. Kazhdan, B. Chazelle, D. P. Dobkin, T. A. Funkhouser, and S. Rusinkiewicz. A reflective symmetry descriptor for 3d models. *Algorithmica*, 38(1):201–225, 2003.

[23] M. M. Kazhdan, T. A. Funkhouser, and S. Rusinkiewicz. Symmetry descriptors and 3d shape matching. In *Symposium on Geometry Processing*, pages 117–126, 2004.

[24] G. Loy and J.-O. Eklundh. Detecting symmetry and symmetric constellations of features. In *Proc. European Conference on Computer Vision*, pages 508–521, 2006.

[25] A. Martinet, C. Soler, N. Holzschuch, and F. X. Sillion. Accurate detection of symmetries in 3d shapes. *ACM Trans. Graph.*, 25(2):439–464, 2006.

[26] P. Minovic, S. Ishikawa, and K. Kato. Symmetry identification of a 3-d object represented by octree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:507–514, 1993.

[27] N. J. Mitra, A. M. Bronstein, and M. M. Bronstein. Intrinsic regularity detection in 3d geometry. In *Proc. European Conference on Computer Vision*, pages 398–410, 2010.

[28] N. J. Mitra, L. J. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph.*, 25:560–568, 2006.

[29] N. J. Mitra, L. J. Guibas, and M. Pauly. Symmetrization. *ACM Trans. Graph.*, 26(3):63, 2007.

[30] P. Oesterling, C. Heine, H. Jänicke, G. Scheuermann, and G. Heyer. Visualization of high dimensional point clouds using their density distribution's topology. *IEEE Trans. Vis. Comput. Graph.*, 99(PrePrints), 2011.

[31] M. Ovsjanikov, J. Sun, and L. J. Guibas. Global intrinsic symmetries of shapes. *Comput. Graph. Forum*, 27(5):1341–1348, 2008.

[32] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli. The toporrery: computation and presentation of multi-resolution topology. *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pages 19–40, 2009.

[33] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust on-line computation of reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007.

[34] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. J. Guibas. Discovering structural regularity in 3d geometry. *ACM Trans. Graph.*, 27(3), 2008.

[35] J. Podolak, A. Golovinskiy, and S. Rusinkiewicz. Symmetry-enhanced remeshing of surfaces. In *Proc. Symposium on Geometry Processing*, pages 235–242, 2007.

[36] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. A. Funkhouser. A planar-reflective symmetry transform for 3d shapes. *ACM Trans. Graph.*, 25(3):549–559, 2006.

[37] D. Raviv, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Symmetries of non-rigid shapes. In *Proc. IEEE International Conference on Computer Vision*, pages 1–7, 2007.

[38] R. M. Rustamov. Augmented planar reflective symmetry transform. *Vis. Comput.*, 24:423–433, May 2008.

[39] D. Schneider, A. Wiebel, H. Carr, M. Hlawitschka, and G. Scheuermann. Interactive comparison of scalar fields based on largest contours with applications to flow visualization. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1475–1482, 2008.

[40] P. D. Simari, E. Kalogerakis, and K. Singh. Folding meshes: hierarchical mesh segmentation based on planar symmetry. In *Proc. Symposium on Geometry Processing*, pages 111–119, 2006.

[41] S. Thrun and B. Wegbreit. Shape from symmetry. In *Proc. IEEE International Conference on Computer Vision*, pages 1824–1831, 2005.

[42] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Topology driven 3d mesh hierarchical segmentation. In *Proc. Shape Modeling International*, pages 215–220, 2007.

[43] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Partial 3d shape retrieval by reeb pattern unfolding. *Comput. Graph. Forum*, 28(1):41–55, 2009.

[44] M. J. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Symposium on Computational Geometry*, pages 212–220, 1997.

[45] G. H. Weber, P.-T. Bremer, and V. Pascucci. Topological landscapes: A terrain metaphor for scientific data. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1416–1423, 2007.

[46] G. H. Weber, S. E. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE Trans. Vis. Comput. Graph.*, 13(2):330–341, 2007.

[47] J. D. Wolter, T. C. Woo, and R. A. Volz. Optimal algorithms for symmetry detection in two and three dimensions. *The Visual Computer*, 1(1):37–48, 1985.

[48] K. Xu, H. Zhang, A. Tagliasacchi, L. Liu, G. Li, M. Meng, and Y. Xiong. Partial intrinsic reflectional symmetry of 3d shapes. *ACM Trans. Graph.*, 28(5), 2009.

[49] E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. *ACM Trans. Graph.*, 24(1):1–27, 2005.

[50] X. Zhang. Complementary shape comparison with additional properties. In *IEEE Proceedings of Volume Graphics*, pages 79–86, 2006.

[51] X. Zhang, C. L. Bajaj, and N. A. Baker. Affine invariant comparison of molecular shapes with properties. In *ICES Tech Report*, 2005.

[52] X. Zhang, C. L. Bajaj, B. Kwon, T. J. Dolinsky, J. E. Nielsen, and N. A. Baker. Application of new multi-resolution methods for the comparison of biomolecular electrostatic properties in the absence of global structural similarity. *SIAM J. Multiscale Modeling and Simulation*, 5:1196–1213, 2006.

[53] J. Zhou and M. Takatsuka. Automatic transfer function generation using contour tree controlled residue flow model and color harmonics. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1481–1488, 2009.