Symmetry in Scalar Fields

A Thesis

Submitted For the Degree of

Doctor of Philosophy

in the Faculty of Engineering

by

Dilip Mathew Thomas

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

July 2014

To the undying spirit of human race to learn and grow.

Acknowledgements

First and foremost, I thank Almighty God for his grace and blessings without which I would not have been able to complete this thesis. There were times when I felt I was not making any progress with my research and doubted myself but eventually, everything fell in place because He was with me throughout.

I would like to thank my advisor, Vijay Natarajan, for his guidance and support. I will fondly remember the discussions we have had that immensely improved the quality of this thesis. I will always admire his disciplined approach towards research, his eye for detail, and his skills in improving the exposition and presentation of technical content.

I will always cherish the wonderful years I have spent in CSA and IISc - coursework, assignments, lab, tea board, hostel life, movies, road trips, tours, and all the crazy stuff with the gang of four. I enjoyed the time I spent with my labmates discussing myriads of topics. JC, Pherums, Kris, Sher, Math, Sumesh, and Deepak added a lot of color to IISc life. I will always be grateful to CSA office staff for their help in taking care of administrative formalities with a pleasant face.

Finally, I would like to thank my family for their patience and support. They were always there to listen to my frustrations and to encourage me.

Thanks to everyone who made this thesis a reality.

Publications based on this Thesis

1. Dilip Mathew Thomas and Vijay Natarajan.
   "Multiscale symmetry detection in scalar fields by clustering contours"
   IEEE Transactions on Visualization and Computer Graphics (IEEE SciVis 2014), 20 (12), 2014, 2427-2436.

2. Dilip Mathew Thomas and Vijay Natarajan.
   "Detecting symmetry in scalar fields using augmented extremum graphs."
   IEEE Transactions on Visualization and Computer Graphics (IEEE SciVis 2013), 19 (12), 2013, 2663-2672.

3. Talha Bin Masood, Dilip Mathew Thomas, and Vijay Natarajan.
   "Scalar field visualization via extraction of symmetric structures."
   The Visual Computer (CGI 2013), 29 (6-8), 2013, 761-771.

4. Dilip Mathew Thomas and Vijay Natarajan.
   "Symmetry in scalar field topology."
   IEEE Transactions on Visualization and Computer Graphics (IEEE Vis 2011), 17(12), 2011, 2035-2044.

Abstract

Scalar fields are used to represent physical quantities measured over a domain of interest. 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.

This thesis proposes three methods to detect symmetry in scalar fields. The first method models symmetry detection as a subtree matching problem in the contour tree, which is a topological graph abstraction of the scalar field. The contour tree induces a hierarchical segmentation of features at different scales and hence this method can detect symmetry at different scales. The second method identifies symmetry by comparing distances between extrema from each symmetric region. The distance is computed robustly using a topological abstraction called the extremum graph. Hence, this method can detect symmetry even in the presence of significant noise. The above methods compare pairs of regions to identify symmetry instead of grouping the entire set of symmetric regions as a cluster. This motivates the third method which uses a clustering analysis for symmetry detection. In this method, the contours of a scalar field are mapped to points in a high-dimensional descriptor space such that points corresponding to similar contours lie in close proximity to each other. Symmetry is identified by clustering the points in the descriptor space.

We show through experiments on real world data sets that these methods are robust in the presence of noise and can detect symmetry under different types of transformations. Extraction of symmetry information helps users in visualization and data analysis. We design novel applications that use symmetry information to enhance visualization of scalar field data and to facilitate their exploration.

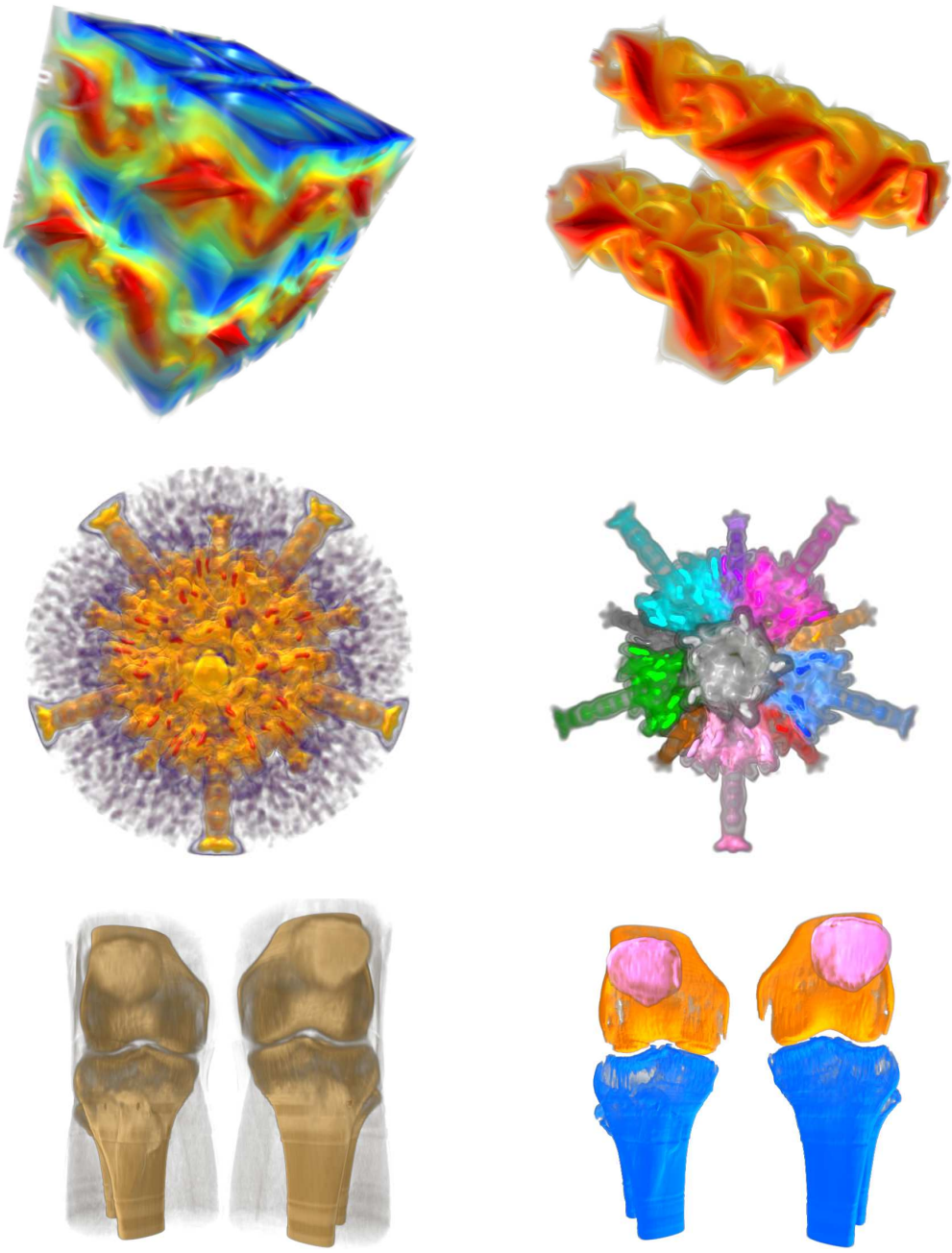


Figure 1: Symmetry is ubiquitous in scientific data sets as seen in the volume rendered images of (top left) the temperature distribution in a Taylor-Green vortex flow simulation, (middle left) a cryo-electron microscopy image of a virus, and (bottom left) a CT scan image of a pair of knees. While human beings are endowed with special cognitive ability that makes symmetry detection easy, automatic detection of symmetry is a challenging task for computers. This thesis proposes algorithms to identify symmetry and show different applications that exploit symmetry information to improve visualization and exploration of scalar field data. Symmetry-aware transfer function highlights the symmetric regions detected by (top right) identifying similar subtrees of the contour tree, (middle right) comparing distances between extrema using extremum graph, and (bottom right) clustering contours in a high dimensional shape descriptor space.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Symmetric patterns are omnipresent in the world around us – be it reflectional symmetry of the wings of a butterfly, translational symmetry of the bricks in a wall, or rotational symmetry of the spokes of a wheel. 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. Symmetry plays an important role in our understanding about the world around us. Therefore, detecting and characterizing symmetry is equally important in fields like engineering, biology, chemistry, and physics. For example, in crystallography, symmetry information is used to determine the structure of a crystal [26]. In product design, symmetry is important to ensure functional efficiency and optimal manufacturing cost [13]. Our body structure exhibit symmetry and deviation from it helps in diagnosing abnormalities in the development of organs [89].

1.1 Symmetry Detection in Scalar Fields

A scalar field is a real valued function defined on a domain of interest. Scalar field data may be generated from experimental and computational methods in different disciplines and they often contain symmetric or repeating patterns. Visualizing the symmetry present in the data provides important cues to domain experts in understanding and characterizing the properties of the underlying scientific phenomenon. The left column

of Figure 1 shows three examples of symmetry in scalar fields. In this thesis, we study the problem of detecting symmetric or repeating patterns from scalar fields and its applications to feature exploration and visualization of scalar fields. The right column of Figure 1 shows the symmetric regions extracted using the solutions proposed in this thesis.

Symmetry detection is an active area of research in geometry processing, computer graphics, image processing, and computer vision communities [63]. As much as study of symmetry is important in shape processing, study of symmetry in scalar fields 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.

While human beings have special cognitive abilities that make it easy to recognize symmetric patterns, it is a tough task for a machine to do so. Automatic detection of symmetry is a challenging problem because both the segmentation of the domain into potential symmetric segments and the correspondence between segments that are symmetric need to be determined. Moreover, real life data sets never exhibit perfect symmetry. This introduces additional challenges in determining symmetry in an approximate sense as well as handling noise and missing parts within symmetric regions in the data. Since the search space for locating symmetric segments is quite large, it is also important to design an algorithm that is computationally efficient. Symmetry in shapes is associated with a group structure on geometric objects that are invariant under transformations. For scalar fields, it is more meaningful to relax this constraint and identify all repeating occurrences since this is more useful for data exploration. Moreover, for scalar fields, domain experts are interested in studying important features that provide

insights about the underlying scientific phenomena that is being analyzed. Therefore, it is pertinent for a scalar field symmetry detection algorithm to report symmetric segments in a feature-aware manner.

1.2 Contributions

This thesis proposes three methods to address the above-mentioned challenges in detecting symmetry in scalar fields. We demonstrate the usefulness of these methods through applications that make use of the symmetry information to enhance scalar field visualization and exploration. Since it is important to report symmetric regions in a feature-aware manner, we use topological data structures, namely, the contour tree [100] and the extremum graph [25] to segment the domain into important features. Restricting the search space for symmetry detection to the segmentation induced by the contour tree and the extremum graph results in computationally efficient detection of symmetry. By making use of robust similarity measures, we design three algorithms to detect approximate symmetry and symmetry in the presence of noise. We briefly describe these methods below.

1.2.1 Symmetry Detection Using Contour Trees

The first method models symmetry detection as a subtree matching problem in the contour tree [92], which is a topological graph abstraction of the scalar field. We propose a comparison measure for identifying similar subtrees of a contour tree based on a topological measure called persistence. This leads to an algorithm that detects symmetry by classifying subtrees of the contour tree into different groups. Regions of the domain corresponding to subtrees that belong to a common group are extracted and reported to be symmetric. The main advantage of this method is that it can detect symmetry at different scales very quickly for clean data sets since the contour tree induces a hierarchical segmentation of features at different scales. However, this method is not suited for data sets with significant noise. Noise in the data may destroy the repeating structure of the

subtrees corresponding to symmetric regions in the contour tree.

1.2.2 Symmetry Detection Using Extremum Graphs

The second method we propose identifies symmetry by comparing distances between extrema from each symmetric region [93]. We propose a data structure called the augmented extremum graph and use it to design an algorithm for robust estimation of distances. The augmented extremum graph captures both topological and geometric information of the scalar field and enables robust and computationally efficient detection of symmetry. This method has the advantage that it can detect symmetry even in the presence of significant noise. However, it requires user selection of a set of seed extrema for graph traversal and does not detect symmetry at multiple scales.

1.2.3 Symmetry Detection Using Contour Clustering

The study of symmetry detection in shapes have established that clustering based analysis result in superior performance and robust identification of symmetry. Such an analysis avoids the need for comparing pairs of regions to identify symmetry and can instead group a set of symmetric regions as a cluster. This motivates the third method which uses a clustering analysis for symmetry detection [94]. In this method, we present a novel representation of scalar fields as a collection of points in a contour shape descriptor space with the aim of studying the similarity relationship between the contours. The representation uses a mapping of the contours in a scalar field to points in a high-dimensional transformation-invariant descriptor space. We leverage the power of this representation to design a clustering based algorithm for detecting symmetric regions in a scalar field. By selecting a contour from each arc of the contour tree, we are able to detect symmetry at different scales similar to the contour tree based approach. Noise in the data is seamlessly handled by the clustering framework through the choice of a robust shape descriptor. Thus, our approach combines the advantages of the earlier methods based on the contour tree and extremum graph.

The main contributions of this thesis are the following:

- A method to detect symmetry by identifying similar subtrees of the contour tree. This method can detect symmetry at multiple scales [92].
- A method to detect symmetry by comparing distances between extrema of a scalar field using the extremum graph. The distance computation is robust and hence this method can detect symmetry even in the presence of significant noise [93].
- A clustering based approach to detect symmetry by identifying geometrically similar contours. This method not only combines the advantages of the earlier two methods but also addresses their limitation in ensuring geometric symmetry of the regions reported [94].
- Applications that utilize symmetry information for improved visualization and exploration of scalar fields.

We show through experiments on real world data sets that the proposed methods are robust, computationally efficient, and can detect symmetry under different types of transformations. As part of designing symmetry detection algorithms, we also propose a comparison measure for identifying similar subtrees of the contour tree, an augmentation of the extremum graph that incorporates geometric information about the scalar field and an alternate representation of contours in a scalar field as points in a high-dimensional shape descriptor space for performing similarity analysis of scalar fields. We believe that these are significant contributions in studying scalar fields that is independent of the symmetry detection algorithms and will lead to newer techniques in analyzing scalar fields. We also believe that the methods we propose for symmetry detection will open new frontiers in analyzing structural similarity of scalar fields and more applications of symmetry detection will emerge.

Chapter 2

Related Work

Symmetry detection is an active area of research in geometry processing, computer graphics, image processing and computer vision communities. The problem of identifying symmetry in scalar fields has received attention among researchers only in recent years. Although some of the methods proposed to report symmetry in geometric shapes have been extended to scalar fields, they suffer from the inability to capture properties inherent to scalar fields. For example, these extensions are unable to report symmetric regions in a feature-aware manner. This chapter provides a brief overview of existing techniques for detecting symmetry in shapes and scalar fields, points out the drawbacks of existing symmetry detection methods in scalar fields, and highlights the solutions proposed in this thesis to address them.

2.1 Symmetry in Shapes

In the geometry processing community, geometric shapes are represented using low level primitives like polygon meshes and point clouds. Naturally, there is a lot of interest in retrieving higher level geometric information like symmetry from the lower level representation. Early work studied detection of perfect symmetry [1, 2, 103]. Typically, geometric properties of shapes, like location of points, are used to identify the axis of symmetry [14, 45, 62]. Perfect symmetry is rarely encountered in practice 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 detection of symmetry in shapes. Graph representations built from geometric features that are extracted from a shape have been studied to find similar subgraphs representing symmetric structures [6, 7, 10, 12]. Local symmetry information have been used to cast votes in a high-dimensional space where clusters of votes correspond to significant symmetries [56, 64, 71, 75, 105]. Shape descriptors based on spherical harmonics and Fourier methods [5, 47, 48, 59], and spectral analysis [5, 22, 56, 104] have also been used for symmetry identification. An interested reader may refer to the survey on symmetry in geometry by Mitra et al. for more details [63].

Symmetry information computed from shapes has been exploited to design different applications 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 [48, 75]. 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 [35, 36, 65, 74]. Detecting symmetry helps in decomposition of a model into different parts and can be used for segmentation [75, 88] and to detect redundancies in the model for compression [59, 71, 88]. Symmetry in models is used for scan completion and surface reconstruction by correcting distortions present in noisy and incomplete data acquired through 3D scanners [71, 95]. In this thesis, we propose several new applications of detecting symmetry in scalar fields that enhances visualization and exploration of scalar fields, namely, query driven exploration, symmetry-aware isosurface extraction, symmetry-aware transfer function design, asymmetry visualization, proximity-aware feature visualization, and symmetry-aware editing and rendering.

2.2 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 nascent area of research. Extensions of symmetry detection methods in shapes [12, 47] have been considered in the past for detecting symmetry in scalar fields [43, 49]. However, they struggle to address the challenges in extending geometric methods to scalar fields. Scalar field data sets are significantly larger in size compared to geometric shape data sets and hence symmetry detection is computationally costly. Geometric methods typically consider geometric information derived from a small region around each sample point of the domain for symmetry recognition. A direct extension of this approach to scalar fields suffers from the difficulty of capturing important features in the data and leads to poor performance in extracting higher level features and handling noise. Moreover, the scalar field is considered to be continuous over the domain by interpolating the values at the discrete set of sample points. Inspecting only local information at the sample points introduces additional challenges due to discretization errors since the symmetric counterpart for a given point may be an interpolated point.

Hong and Shen [43] extend an earlier work on reflective symmetry descriptors [47] and use a parallel algorithm to detect global reflective symmetry in 3D scalar fields. They estimate, for every candidate plane, the distance between the given scalar function and its closest perfect symmetric function. This distance measure is used to detect planes of symmetry that minimize the difference between the scalar value at a point and its reflection. Their method requires computation of symmetry distance by examining the entire data set for all planes passing though the volume and is computationally expensive. Moreover, it is restricted to identifying only global reflective symmetries, whereas the methods we propose can detect symmetries under other rigid body transformations as well as partial symmetries. Kerber et al. [49] build a graph network of crease line features and detect symmetry by computing geometric transformations that match subgraphs within the crease line network. This is an extension of an earlier work on symmetry detection in shapes [12]. However, unlike the case of shapes, the assumption that features

in scalar fields can be described using geometric line features is too restrictive in practice. Even for line features, this method cannot handle noise in the data robustly since the alignment of noisy networks of lines will be poor.

Several methods have been proposed to identify similarity between scalar fields without explicitly addressing the challenge of detecting symmetric regions. Bruckner et al. propose an information theoretic measure of data independence for isosurface similarity detection [17, 39]. This method uses mutual information between distance transforms to quantify the information common to two isosurfaces and builds a similarity map between all pairs of isosurfaces. Clusters with high mutual information correspond to similar isosurfaces and can be identified both within and across data sets. This method is computationally expensive since it requires calculation of mutual information between all pairs of isosurfaces. Further, their goal is to select a subset of isovalues which are important by identifying redundant isosurfaces that form an onion-peel like layered arrangement. On the other hand, the methods we propose are more suited for extracting regions in the domain which are symmetric. While comparison measures have been designed to study the relationship between different scalar fields defined on a common domain [31, 66, 84, 83], the focus of this thesis is on detecting repeating patterns belonging to different subdomains 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 [41, 109, 110, 111]. Though these methods analyze similarity between scalar fields, their goal is to identify the overall similarity between two shapes, which is different from our goal of identifying symmetry within a single scalar field.

Graph matching methods for symmetry detection significantly reduces the processing time and memory requirements for computing symmetry since the size of the graph is typically much smaller than the size of the input data set. The first two methods that we propose for symmetry detection in scalar fields make use of graph representations of the field. For scalar fields, it is important to report symmetry in a feature-aware manner. Therefore, we rely on topological graph abstractions of the scalar field, namely,

the contour tree [100] and the extremum graph [25]. The first method we propose reports symmetry by identifying similar subtrees of the contour tree through a graph matching procedure [92]. This method defines a similarity score to compare subtrees by measuring the extent of overlap between subtrees of the contour tree. The second method is based on the observation that symmetric regions exhibit similar distribution of distances. We compute distances between extrema of the scalar field by traversing an augmented version of the extremum graph [93]. Even in the presence of significant noise, the distance calculation is robust and therefore this method can detect symmetry in noisy scalar fields. Symmetry detection methods for geometric shapes based on identifying clusters that represent accumulation of local symmetry information have shown superior performance in identifying symmetry. Motivated by this observation, the third method that we propose maps contours of a scalar field to points in a high-dimensional shape descriptor space such that similar contours form clusters in the descriptor space [94]. The symmetric regions are identified through a clustering procedure that locates similar collections of contours of a scalar field.

It is interesting to note that there is a growing interest in recent years to develop distance measures to compare contour trees and its variants [3, 4, 82], similar to the comparison score we propose for detecting symmetry using the contour tree based method. Another recent work that attempts to limit the effect of noise in visualizing scalar fields uses an approach that merges extrema through a smoothing operation [87], similar to the approach followed in the extremum graph based method. We believe these are promising results that indicate the methods proposed in this thesis have wider applications in analyzing scalar field data.

Chapter 3

Definitions

Consider a *scalar field*, $f : \mathbb{M} \to \mathbb{R}$, defined on a simply connected domain $\mathbb{M}$. We use topological abstractions of the scalar field, namely, the contour tree [100] and the extremum graph [25] to detect symmetry in a feature-aware manner and define these terms formally in this chapter. These abstractions are defined using critical points of the domain which are defined for continuous functions. However, in practice, the scalar field is represented using discrete values defined on sample points of the domain. To generate a continuous field from this discrete representation, the domain is triangulated and the scalar values are interpolated linearly over the elements of the triangulation to obtain a *piecewise linear* function. For a vertex v, its *link* is defined as the mesh induced by the set of vertices adjacent to v and its *upper link* is defined as the mesh induced by the adjacent vertices having function value greater than that of v. For piecewise-linear domains, the critical points are characterized by the number of components in the upper link. A point $c \in \mathbb{M}$ is a *maximum* if its upper link is empty, a *minimum* if its upper link is same as its link and a *saddle* if the upper link consists of more than one connected component [30, 113].

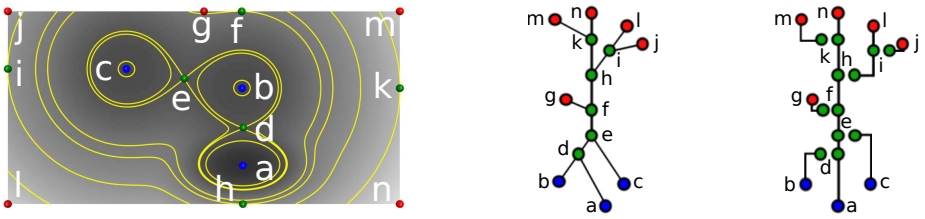


Figure 3.1: Contour tree and branch decomposition. (left) A scalar field and its level sets. (center) The corresponding contour tree captures changes in the topology of the level sets. The red nodes are local maxima, the blue nodes are local minima, and the green nodes are saddles. (right) The branch decomposition representation. The branch an is the longest monotonic path and hence the root branch. The branch hl is a child of an and the parent of ij.

3.1 Symmetry in Scalar Fields

Subdomains $\mathbb{M}_1, \mathbb{M}_2 \subseteq \mathbb{M}$ are said to be *symmetric* if there is a transformation T such that $\mathbb{M}_2 = T(\mathbb{M}_1)$ and $f(x) = f(T(x))$ for all $x \in \mathbb{M}_1$.

3.2 Contour Tree

The preimage of f for a given value $\alpha \in \mathbb{R}$, $f^{-1}(\alpha)$, is called a *level set* of f. A level set may have multiple components and each component is called a *contour*. Let R be an equivalence relation defined on points in $\mathbb{M}$: xRy for $x, y \in \mathbb{M}$ if x, y belong to the same contour. The contour tree is the quotient space induced by this 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 3.1. Consider a sweep of the domain using level sets in the order of increasing function values. A contour is created at a minimum, may merge with another contour at a *join saddle* or split into different contours at a *split saddle*, and is destroyed at a maximum. The nodes of a contour tree correspond to these critical points of the function where the number of connected components of the level set changes. Minima and maxima, also referred to as extrema, correspond to leaf nodes of the contour tree. Saddles correspond to the interior nodes. 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 [33].

3.3 Branch Decomposition

A *branch decomposition* [69] is an alternate representation of the contour tree 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 either from a minimum to a saddle or from a saddle to a maximum. In the former case, the contour is created at a minimum and it merges into a another contour at a saddle while in the latter case the contour splits from a saddle and is destroyed at a maximum. The root branch is an exception. It represents evolution of a contour from a minimum to its destruction at a maximum. The persistence of the branch is equal to the difference between the function values at these points (saddle and extremum). Figure 3.1 shows a contour tree and its branch decomposition. The path $adefhkn$ in the contour tree is the root branch in the branch decomposition. The path hil is a child branch of the root branch. *Simplification* of a contour tree refers to removing all branches whose end points have persistence below a threshold results and the resultant contour tree is called a *simplified* contour tree.

3.4 Extremum Graph

A *Morse cell* is the union of paths of steepest gradient ascent that terminate at a maximum [32], see Figure 3.2. The set of all Morse cells segment the domain $\mathbb{M}$ and this segmentation is called the *Morse decomposition* of $\mathbb{M}$. The Morse cells for the minima

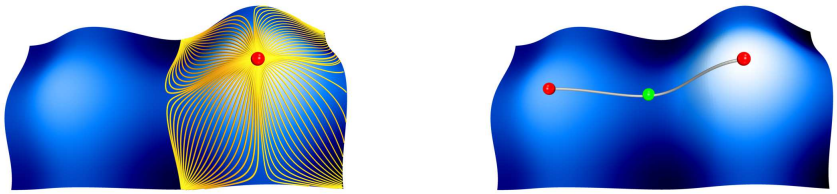


Figure 3.2: Morse decomposition and extremum graph. (left) The Morse cell for the maximum, shown in red, is the union of paths of steepest gradient ascent that terminate at the maximum. (right) Edges in the maximum graph link adjacent maxima to their shared saddle shown in green.

of f is defined analogously by considering $-f$ instead of f and the resulting segmentation is referred to as the Morse decomposition with respect to minima. Using the Morse decomposition, Correa et al. introduced extremum graphs for designing topological spines – a visual representation that captures both the geometry and the topology of the scalar field [25]. We compute an approximation of the Morse decomposition with respect to maxima by sorting all the vertices in the decreasing order of function values, assigning a label to each maximum, and propagating the labels to the adjacent vertices visited in the decreasing order of function values. Each visited vertex inherits the label of the vertices in its upper link and the Morse cell associated with a maximum is the set of vertices with the label of the maximum. The Morse decomposition with respect to minima is computed analogously. If a vertex inherits multiple labels, then it belongs to the common boundary of the Morse cells associated with the labels. Maxima whose Morse cells share a common boundary are said to be adjacent. For each pair of adjacent maxima, the vertex with the highest function value on the common boundary is marked as their *shared saddle*. To capture the adjacency relationship better, we work with the approximate Morse decomposition described above instead of computing the true Morse decomposition using gradient paths. The nodes of a *maximum graph* are the set of maxima and their shared saddles. Edges in this graph link a pair of adjacent maxima to their shared saddle. The *minimum graph* is defined analogously. The *extremum graph* of a scalar field can refer to either the maximum graph or the minimum graph.

Chapter 4

Symmetry Detection Using Contour Trees

Study of the level sets of a scalar field is a popular method for analyzing the behavior of the scalar field. A natural question that arises is if similarity in the behavior of level sets can be used to detect symmetry. We study symmetry in scalar fields with respect to the topology of its level sets using the contour tree. Contour trees have been extensively used in designing tools that assist users in data exploration and visualization like fast computation of isosurfaces and interactive exploration of large data sets by tracking the evolution of contours [16, 18, 19, 100].

We present an efficient algorithm for identifying symmetry in scalar fields that uses the contour tree as an abstract representation of the level sets of the field. This eliminates the need to exhaustively examine all the level sets of the data set for reporting 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

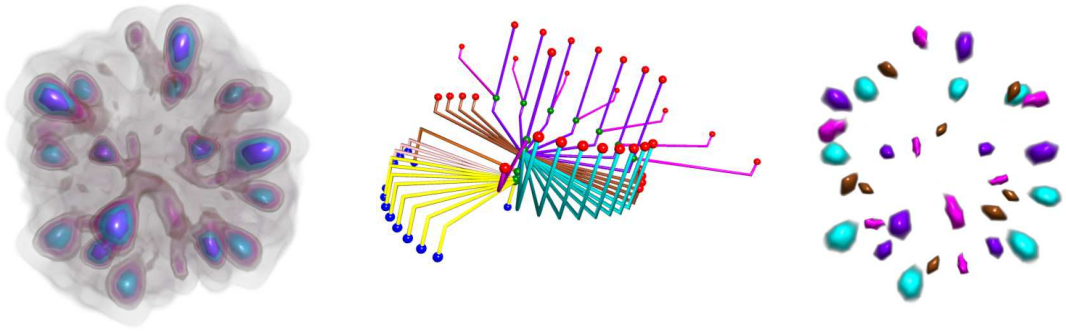


Figure 4.1: Symmetric patterns identified using the contour tree in a cryo-electron microscopy image of RuBisCO molecule in complex with RuBisCO large subunit methyltransferase (EMDB 1734). (left) Volume rendering of the molecule highlight repeating structures in the scalar field. (center) The contour tree for the data set. 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. (right) 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.

subtrees into different groups. The classification is directed by a similarity measure that compares subtrees based on a topological measure called persistence. Figure 4.1 shows a volume rendering of a cryo-electron microscopy image of RuBisCO molecule in complex with RuBisCO large subunit methyltransferase (EMDB 1734). The data set is obtained from EMDataBank (EMDB) [34]. The volume rendering highlights the symmetric or repeating patterns in the scalar field. 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. To identify these regions, 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. We extract regions of the domain corresponding to subtrees that belong to the same group and report them to be symmetric.

The main contributions of this method 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.

Since the contour tree partitions volume data into different regions based on the topology of level sets, it has been used in designing spatially aware transfer functions for volume visualization by assigning distinct transfer functions to different branches of the contour tree [102]. This has been extended to automatic transfer function generation by assigning opacity values to branches using a fluid flow model [112]. These methods do not exploit repeating patterns in the scalar field. We demonstrate that 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 [16, 46, 96, 108] and shape matching [9, 11, 41, 97]. Contour trees have also been used to explore high dimensional data via a two-dimensional terrain representation called the topological landscape [40, 68, 101]. We believe that these methods will also benefit from identifying similar subtrees of the contour tree. Since our symmetry detection algorithm is essentially based on the contour tree, it can be used to identify repeating patterns in any dimension.

4.1 Contour Trees and Symmetry Identification

The topology of level sets of 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 in the topology of level sets. When two branch decompositions are not exact, a natural measure of approximate symmetry is the amount of overlap

(a) (b) (c) (d) (e)

Figure 4.2: 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.

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 [24]. Our similarity measure is similar to the barcode metric and can, in addition, handle hierarchical relationship among the intervals. 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. Thus, once similar subtrees of the contour tree are identified using our similarity measure, the region in the domain corresponding to it can be easily extracted as a union of subdomains corresponding to its constituent arcs. Such a region extracted is highlighted in Figure 4.2(a)-(d) through a volume rendering using a spatially-aware transfer function [102].

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 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. 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 maximum overlap 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 4.2(e).

4.1.1 Symmetry Detection Pipeline

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. The pipeline used for symmetry identification is shown in Figure 4.3. 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 stabilized 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.

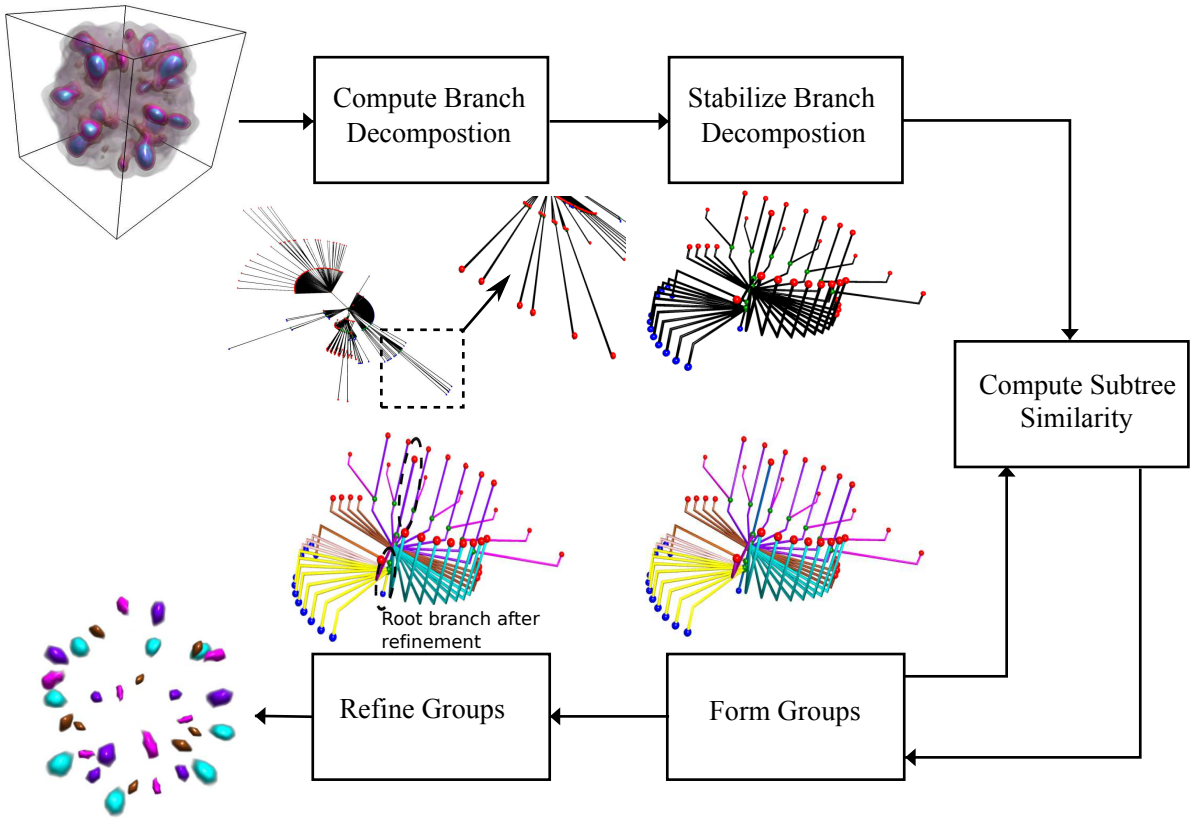
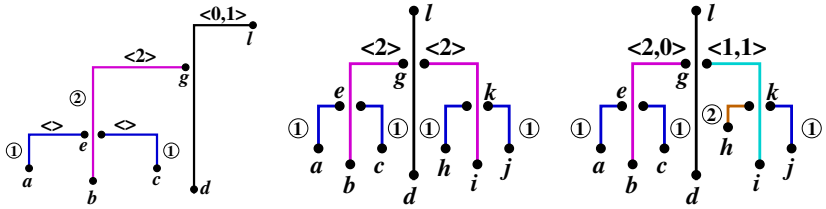


Figure 4.3: Symmetry detection pipeline. Branch decomposition of input data is computed and converted to a representation that is stable in the presence of noise. A similarity measure is used to compare subtrees and classify them into different groups during a bottom up traversal of the branch hierarchy. Group assignment is then refined and regions belonging to the same group are reported as symmetric regions.

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.

4.1.2 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



(a) (b) (c)

Figure 4.4: 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.

representation of subtree hierarchy for matching subtrees. Each subtree of the contour tree 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 4.4(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 4.4(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 4.4(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 4.4(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.

4.1.3 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 summarize the computation of the similarity measure in Algorithm 4.1.

Algorithm 4.1 Computing Similarity Measure (branch b_s, branch b_t)

1: Represent branches b_s and b_t using the hierarchy descriptor (Section 4.1.2).
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 $= \dfrac{O_1 + O_2}{p_s + p_t} \cdot 100.$

Consider the branches bg and ig in Figure 4.4(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 4.5. 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

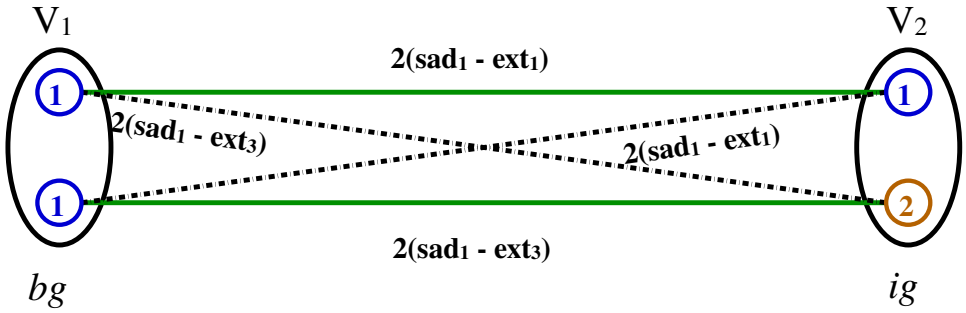


Figure 4.5: A complete bipartite graph is constructed to match groups in the frequency vectors of bg and ig in Figure 4.4(c). A vertex is created for each occurrence of a group and edges are weighted with the amount of overlap between the groups. The total overlap between the frequency vectors is defined to be the maximum weight matching of the graph, shown in green.

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 4.5. 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.

4.2 Algorithm for Grouping Similar Subtrees

We now describe the algorithm that groups subtrees of the contour tree. The algorithm operates in three stages. The first stage generates a stable branch decomposition representation of the contour tree. This ensures that the subtrees can be robustly compared in the presence of noise. In the second stage, similar subtrees are grouped together by comparing their hierarchy descriptors. The groups thus formed are further refined in the

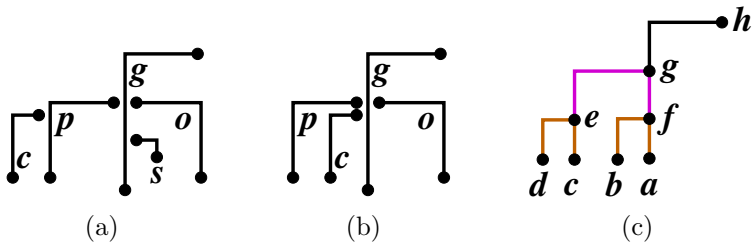


Figure 4.6: Pre-processing and post-processing of branch decomposition. (a) Parent-child relationship of branch c and branch p is not stable. (b) We stabilize the branch decomposition by changing the parent-child relationship. Branch c is moved up the hierarchy to become a sibling of p and a child of 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.

third and final stage.

4.2.1 Stabilizing 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.

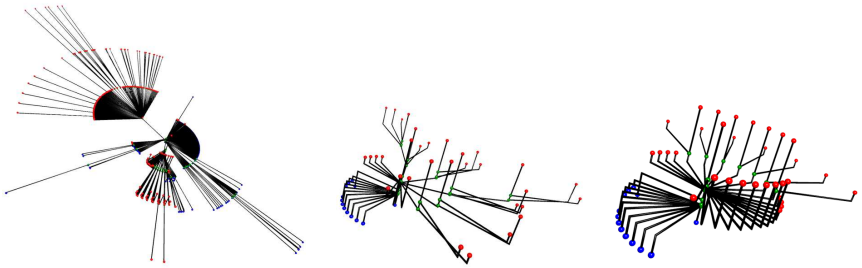
Consider a level set sweep that results in the branch decomposition shown in Figure 4.6(a). The level sets corresponding to branch c merge with that of branch p and in turn merge with that of branch g. Note that the function value corresponding to the saddle of branch c is lower than that of branch p 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 4.6(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 4.6(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 4.6(b). A higher threshold for stabilizing 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 4.7(a) shows the branch decomposition of RuBisCO molecule. Simplification removes low persistence branches that appear as flat edges in a radial layout, see Figure 4.7(b). The stable branch decomposition after changing branch hierarchy is shown in Figure 4.7(c).

4.2.2 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



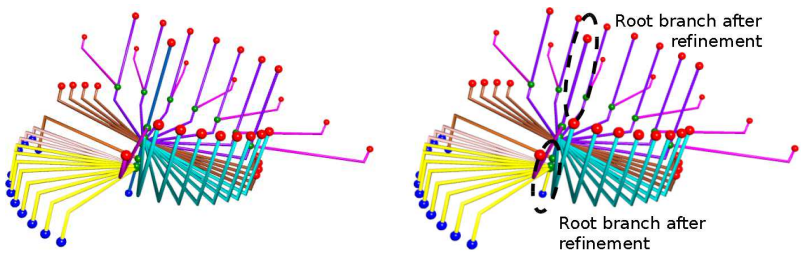
(a) (b) (c)

Figure 4.7: Simplifying and stabilizing 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.

with the hierarchy descriptor of each of the existing groups, as described in Section 4.1. 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 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 4.8(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.

4.2.3 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.



(a) (b)

Figure 4.8: 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.

During the refinement stage, each subtree is assigned to a single group. However, parts of a subtree may better match other subtrees. Consider branch ah in Figure 4.6(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 analyzed 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 4.8(a) is split and regrouped as shown in Figure 4.8(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.

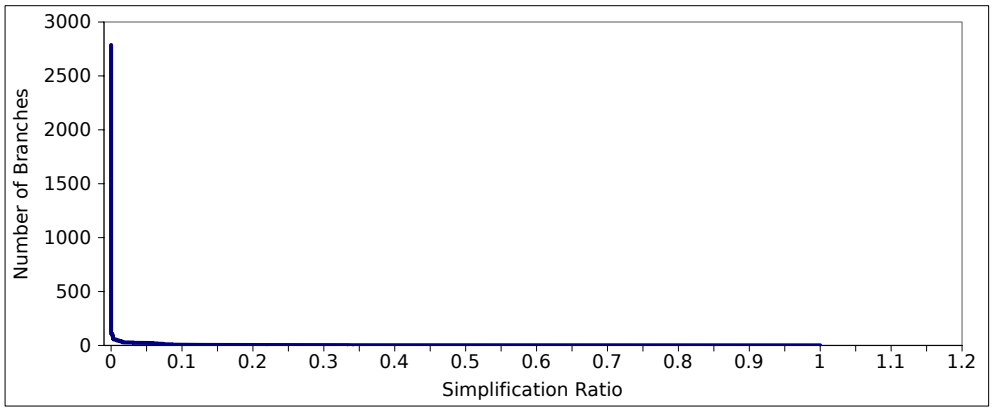


Figure 4.9: Graph showing the number of branches in the contour tree plotted against simplification ratio for the Fuel data set. Initially, there is a sudden drop due to noise in the data. The value at which the graph tapers off is used as an estimate for simplification ratio and stabilization ratio.

4.3 Results and Discussion

We now present results from our analysis of the performance of the symmetry detection algorithm. We also describe our 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 stabilization 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 Figure 4.9. 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 the stabilization 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.

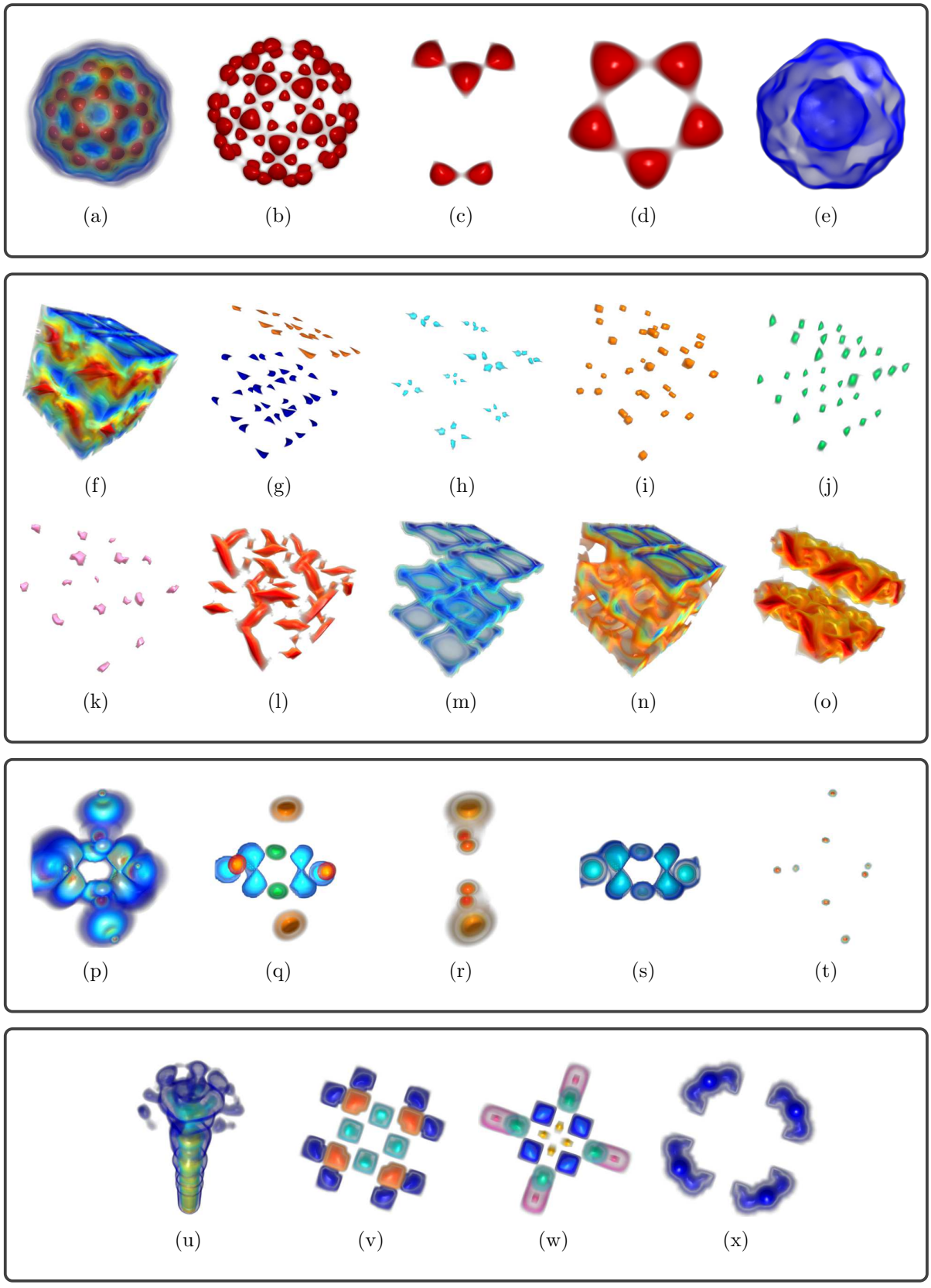


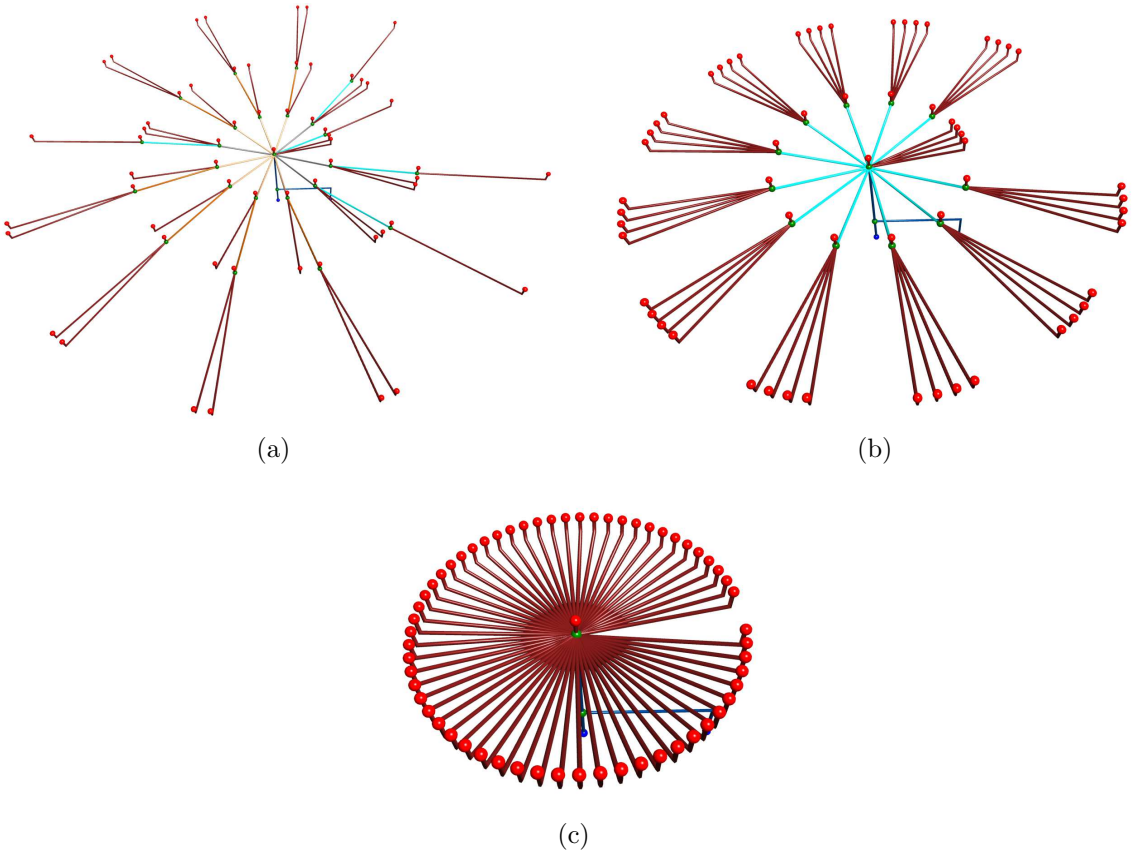
Figure 4.10: Symmetric regions identified within four data sets - Buckyball (first box), Vortex simulation (second box), Neghip (third box), and Fuel (fourth box).

4.3.1 Discussion

Figure 4.10(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 4.10(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 4.10(c). Increasing the threshold to 2% results in five carbon atoms merging together to form a ring as shown in Figure 4.10(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 Figure 4.11.

Figure 4.10(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. Figure 4.10(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. Figure 4.10(g)-(k) show repetitions at small scales, where as Figure 4.10(l)-(m) show repetitions at a larger scale, and Figure 4.10(n)-(o) show the symmetric regions at the largest scale. Though the regions shown in Figure 4.10(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 4.10(m). Figure 4.10(p) shows the Neghip data set and reflective symmetry present in it. Figure 4.10(q) shows four different reflective symmetric regions identified by our algorithm. Figure 4.10(r)-(s) show symmetric regions at larger scales. We also identify eight small symmetric regions as shown in Figure 4.10(t). Figure 4.10(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 Figure 4.10(v)-(w). All regions repeat four times except the blue region in Figure 4.10(v) which repeats eight times. The blue regions merge to form a larger symmetric region shown in Figure 4.10(x). The branch decomposition for these data sets are shown in Figure 4.12. Figure 4.1 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.



(a) (b)

(c)

Figure 4.11: Contour tree for the Buckyball data set after pre-processing using different stabilization ratios. (a) At 1%, one set of carbon atoms merge in groups of two while another set merge in groups of three. (b) At 2%, carbon atoms merge in groups of five. (c) At 8%, all children branches merge with the root branch and we do not see any relationship in the way the carbon atoms merge.

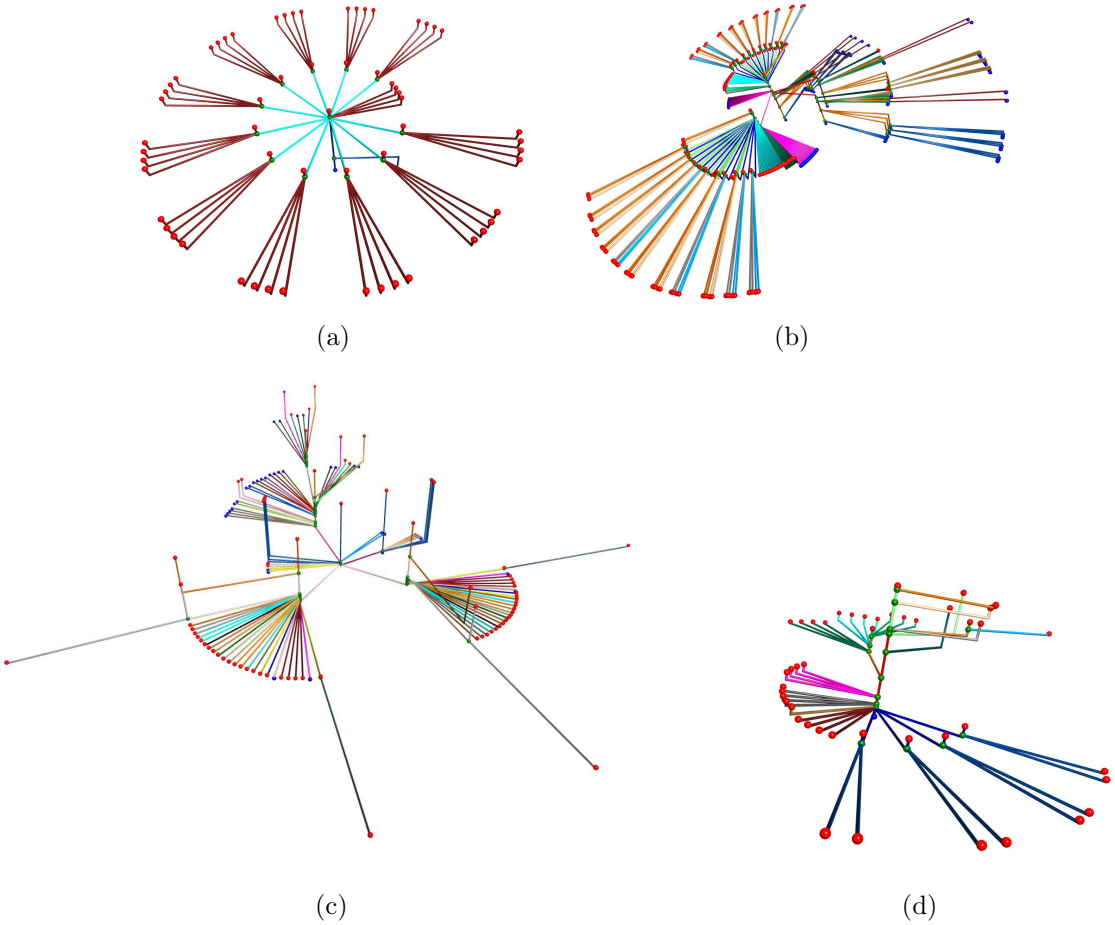


Figure 4.12: Stable branch decomposition showing refined groups for (a) Buckyball, (b) Vortex simulation, (c) Neghip, and (d) Fuel.

4.3.2 Performance

The computational cost of our algorithm is dominated by the time spent in the bottom up traversal to identify similar subtrees. Table 4.1 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 [28, 29, 70]. 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.

Table 4.1: 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

4.4 Conclusions

We have described a method for detecting symmetry in scalar field topology. Our method analyzes 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.

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. This is illustrated with an example in Figure 5.1. 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 and the next two methods we propose address these challenges.

Chapter 5

Symmetry Detection Using Extremum Graphs

Symmetry detection methods that use graph-based representations of the scalar field like the contour tree assume that symmetry in the data manifests as similar subgraphs. These methods employ subgraph matching algorithms to identify symmetry. As shown in Figure 5.1, noise in the data results in noisy subgraphs that are not similar. Hence, these methods perform poorly when there is significant noise in the data. In this chapter, we propose a data structure called the augmented extremum graph and use it to design a novel symmetry detection method based on robust estimation of distances. The design of the augmented extremum graph involves incorporation of both topological and geometric information of the scalar field and is derived from a simplified representation of the Morse-Smale complex called the extremum graph [25]. The Morse-Smale complex is a partition of the domain into cells based on the gradient of the scalar field and has been used for segmentation [20, 106], meshing [27, 54], and studying various scientific phenomena such as turbulent structures, channel structures in porous material, and for terrain representation [15, 37, 53].

Augmenting topological constructs with geometric information results in powerful abstract representations as demonstrated recently in the study of pore networks [42, 98]. Augmenting the extremum graph with geometric information facilitates efficient

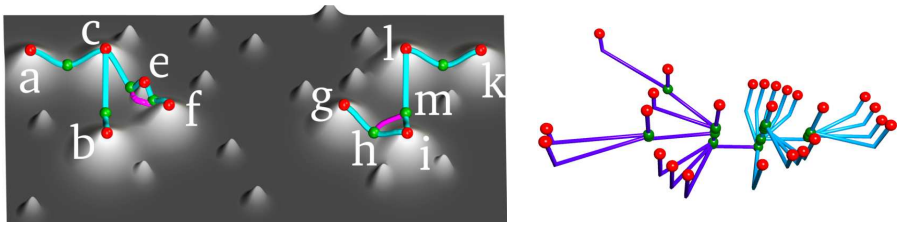


Figure 5.1: The contour tree-based method fails to detect symmetry when the repeating structure of the branches corresponding to the symmetric regions is destroyed either due to noise or changes in the level set topology. (left) A synthetic terrain data set where the region on the left and right are symmetric. Paths in the augmented extremum graph may be used to estimate distances robustly in the presence of noise whereas (right) noise in the data destroys the repeating structure of the subtrees in the branch decomposition representation. Thus the violet and the blue subtrees corresponding to the symmetric regions on the left and the right are not identified to be repeating and the symmetry is not detected.

detection of symmetry in a geometry-aware manner. Figure 5.2 shows an illustration of our method. The left figure shows 4-fold rotational symmetry in the cryo-electron microscopy (cryo-EM) data of Rubisco RbcL8-RbcX2-8 complex (EMDB 1654). Our algorithm performs Morse decomposition of the data and selects a set of Morse cells as seed cells. The augmented extremum graph is traversed to compute distances from the seed cells to the remaining Morse cells and merge the seed cells into super-seeds. Eight seeds are selected for this data set and they merge to form four super-seeds, as shown at the center in Figure 5.2. The four super-seeds provide an initial estimate of the 4-fold symmetry in the data which is then expanded in a region growing stage. The 4-fold symmetry in the data thus identified is shown on the right.

The main contributions of this work are the following:

- A symmetry detection method that can detect symmetries in scalar fields with respect to rigid body transformations. We propose a data structure called the augmented extremum graph, which integrates topological and geometric information of the scalar field and use it for computationally efficient and geometry-aware symmetry detection.

- Our algorithm detects global symmetry even in the presence of significant noise.

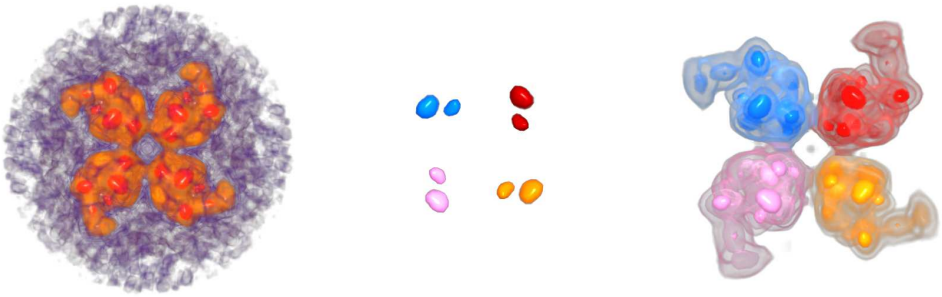


Figure 5.2: Symmetry identification algorithm based on augmented extremum graph detects symmetry even in the presence of significant noise in the electron microscopy data of the Rubisco RbcL8-RbcX2-8 complex (EMDB 1654). (left) Volume rendering shows symmetry and noise in the data. (center) A set of seed cells is chosen as source vertices for traversing the augmented extremum graph of the data. During the traversal, the seed cells merge together to form four symmetric super-seeds. Seed cells that belong to a common super-seed are shown with the same color. (right) The initial estimate of symmetry is expanded in a region growing stage to identify the symmetric regions. A symmetry-aware transfer function highlights the 4-fold rotational symmetry detected in the Rubisco complex.

If the amount of noise is not high, our algorithm can detect partial symmetry as well.

- Existing techniques that detect scalar field symmetry using subgraph matching have limitations in handling noisy data [12, 92]. Our method addresses this shortcoming by using the augmented extremum graph only for distance computation and not relying on subgraph matching for symmetry detection.

- We demonstrate the effectiveness of our method through extensive experiments on several cryo-EM data sets and show that our method yields better results in the presence of noise. Thus, our method is able to bridge a gap in the applicability of previous methods to noisy data sets.

5.1 Augmented Extremum Graphs

We use the extremum graph as a data structure for computing distances because it captures proximity relationship between the extrema of a scalar field. Since symmetry

identification with respect to the maximum graph and the minimum graph are analogous, henceforth we restrict our attention to the maximum graph. Figure 5.3 shows a scalar field where the light gray patches correspond to regions with high function values. The maximum graph, on the left, shows adjacent maxima such as a and b connected to their shared saddle c. Correa et al. introduced extremum graphs for designing topological spines – a visual representation that captures both the geometry and the topology of the scalar field [25]. They note that the extremum graph captures the geometric proximity of the extrema much better than the contour tree. This is illustrated in Figure 5.3, where the contour tree (shown partially) requires a path of five edges, *e-g-h-k-l-n*, to connect the geometrically close extrema e and n. On the other hand, the extremum graph connects them with a shorter and a more natural path *e-l-n* of length two. Also, the contour tree connects extrema a and m that are far apart with the path *a-c-h-k-m* of length four, which is shorter than the corresponding natural path *a-p-d-g-e-k-m* of length six in the extremum graph. Such unintuitive connections in contour trees fail to capture the proximity relationship between the extrema.

While the abstract definition of the extremum graph captures minimal proximity information, it does not capture significant geometric information about the scalar field. The extremum graph can be viewed as a gradient flow graph, whose edges connect a shared saddle to its adjacent extrema along the path of steepest gradient ascent / descent. For a 2D scalar field, such a path in the domain can be embedded in the surface plot of the function and this allows incorporation of more geometric information about the

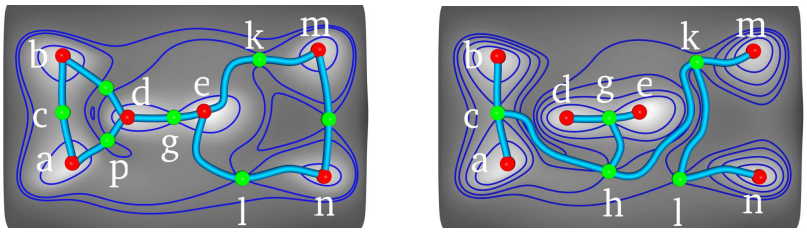


Figure 5.3: The extremum graph represents the geometric structure of a scalar field better than the contour tree. (left) The extremum graph connects the adjacent extrema through their shared saddle and represents proximity information better than (right) the contour tree.

scalar field into the extremum graph, see Figure 5.4.

5.1.1 Geodesic Distance Between Extrema

Geodesic distance between two points on a shape is an intrinsic property of the shape that remains invariant under isometric transformations and has been used for identifying symmetries of a shape [50, 77]. In the rest of this chapter, we use the phrase "path between two points" to refer to the path in the hypersurface plot of the scalar function.

Our method requires the computation of the geodesic path between two extrema. Since computing the exact geodesic path is expensive, we propose to approximate it using the shortest path in the embedded extremum graph. However, the extremum graph consists only of edges between extrema and shared saddles and thus the shortest path between two non-adjacent extrema in the extremum graph deviates considerably from the true geodesic path. For example, consider the shortest path between the extrema c and f in the extremum graph in Figure 5.4. For clarity, only a subset of the edges of the extremum graph is shown in blue. The shortest path consists of four edges - two edges that connect the extrema c and e and two edges that connect the extrema e and f through their respective shared saddles. Depending on the height of the intermediate extremum e, the length of this path can be arbitrarily different from the true geodesic path because the shortest path in the extremum graph is forced to pass through e whereas the geodesic path may bypass the extremum.

To overcome this problem, we augment the extremum graph by inserting additional

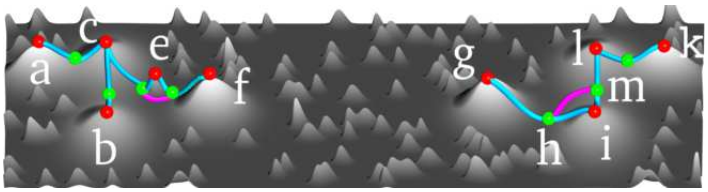


Figure 5.4: The geodesic distance between an extremum and its shared saddle is approximated as the length of the corresponding edge in the embedded extremum graph. To approximate the geodesic path from c to f, the intermediate extremum e is bypassed by directly connecting its shared saddles with the magenta edge. Distance between the pair of extrema c and f is similar to the distance between its symmetric pair l and g.

edges that directly connect the shared saddles of an extremum as shown by the magenta edge that bypasses the extremum e. Thus, inserting edges between the shared saddles allows extrema corresponding to noise to be bypassed. This also makes the geodesic path estimate more robust. Assume that due to noise, a shared saddle does not exist between the extrema g and l and the shortest path between g and l in the extremum graph is g-h-i-m-l. The path has to traverse through all the intermediate extrema, in this case the extremum i, which distorts the shortest path estimate. By directly connecting the intermediate shared saddles, shown by the magenta edge hm, we obtain a better estimate for the shortest path. This path is not significantly different from the true geodesic path between g and l. On the other hand, if a shared saddle did exist between g and l and the path between g and l through this shared saddle was the shortest, then our method will detect and use this path.

We also note that explicit computation of the gradient lines is required to embed the extremum graph, whereas an abstract representation of the extremum graph can be easily obtained once the shared saddles and their adjacent extrema are determined. Hence, instead of computing the length of the edges in the embedded extremum graph, we assign to each edge a weight equal to the approximate length of the geodesic path between the endpoints of the edge. Though we augment edges between the shared saddles for distance computation, we do not explicitly insert these edges in the extremum graph. Instead, we perform all computations on the extremum graph and infer the augmented information on-the-fly during the path computation. While all illustrations of the extremum graph are for 2D scalar fields, the discussion holds for higher dimensions also.

5.1.2 Symmetry from Distances

We observe that for two regions that are symmetric, the distance between a pair of extrema in one region is equal to the distance between their symmetric counterparts in the second region. Consider the paths from symmetric mountains c and l in Figure 5.4 to the neighboring symmetric mountains f and g, respectively. Although the two paths differ significantly, the geodesic distance estimate is roughly the same. The noisy hill e

is bypassed and the absence of the shared saddle between g and l is compensated for by directly connecting the saddles h and m. The distance between extrema is estimated robustly and similar distances is a good indicator of symmetric regions. We assume that different symmetric regions are well separated in terms of distances in the hypersurface plot of the scalar field while features within a symmetric region lie in close proximity to each other. This can be seen in Figure 5.4 where the path between a mountain on the left and one on the right is longer than the path between a pair of adjacent mountains both of which lie on the same side. We exploit this separation between symmetric regions and remove long paths from the augmented extremum graph to obtain a set of disconnected subgraphs. These graph cuts partition the domain into different symmetric regions. For example, the domain shown in Figure 5.4 will be partitioned into the region on the left and the region on the right after removing the long paths in the augmented extremum graph.

5.2 Extremum Graph and Symmetry Detection

Our symmetry detection pipeline is shown in Figure 5.5. Given a scalar field as input, we first compute its Morse decomposition and the associated extremum graph. The resulting Morse decomposition may be over-segmented. We coarsen it and generate a simplified extremum graph. Next, we classify the Morse cells into different groups based on similarity of their histograms. The extrema of the cells that belong to one such group are selected as a seed set. These pre-processing steps are described in Section 5.2.1 and Section 5.2.2. Starting from each seed, the augmented extremum graph is traversed iteratively by visiting, at each iteration, the extremum that is closest to the seed. When the traversal from a seed reaches another seed, the two seeds are declared to be merged. Once all the seeds merge, we use the distance between pairs of seeds to cluster them into super-seeds. Augmented extremum graph traversal and super-seed formation is explained in Section 5.2.3 and Section 5.2.4. The Morse cells corresponding to the seeds that belong to a common super-seed are the initial estimates of the symmetric regions.

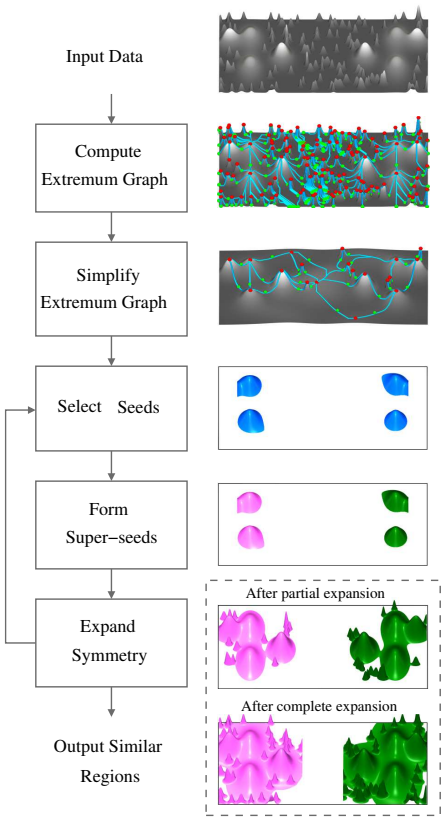


Figure 5.5: Symmetry detection pipeline. The extremum graph of the input data is computed and simplified. A set of extrema are then selected as seeds and the augmented graph is traversed starting from the seeds. During the traversal, seeds merge to form symmetric super-seeds. Finally, the symmetry of the super-seeds is expanded in a region growing stage and the symmetric regions are reported.

The symmetry is expanded in a region growing stage by merging cells in the neighborhood of the seeds. For detecting different partial symmetries, the above procedure is repeated with a different seed set. These post-processing steps are described in Section 5.2.5.

5.2.1 Simplification of the Extremum Graph

We compute the extremum graph and simplify it for computationally efficient traversal. Our simplification procedure coarsens the initial Morse decomposition by merging a Morse cell into its adjacent cell based on two parameters – persistence and size. For an edge between the extremum of a cell i and the saddle shared with an adjacent cell j, if its persistence [33], which is the difference in function values between the extremum

and the saddle, falls below a threshold, then i is merged into j. Similarly, if the size of a Morse cell, measured in terms of the number of vertices that lie within the cell, falls below a threshold then it is merged with an adjacent cell.

When a cell i merges into a cell j, the extremum of i, and all its shared saddles and edges are removed from the graph. The shared boundary of j also changes and the shared saddles are updated / created and edges are inserted between the shared saddles and the adjacent extrema. A saddle that exists after simplification is called a *surviving saddle* and a saddle removed during simplification is called a *canceled saddle*. Since computing the length of geodesic paths encountered during the graph traversal is expensive, we approximate it as the sum of Euclidean distances of the edges in the path. For 3D scalar fields, we normalize the spatial coordinates and the scalar values of the input to lie within the unit hypercube in $\mathbb{R}^4$ and embed the abstract extremum graph in $\mathbb{R}^4$. The first three coordinates of a node are the spatial coordinates and the fourth coordinate is the scalar value of the corresponding critical point. For simplicity, we use uniform weights for normalization. Any normalization procedure may be used as long as it preserves the relative distances between the seeds.

Simplification removes many of the shared saddles and a surviving saddle may lie arbitrarily far away from the extrema that it is adjacent to after simplification. Directly using the Euclidean approximation for computing the length of a path in the simplified graph will lead to significant errors. Therefore, the canceled saddles and the edges removed during simplification are taken into account to estimate the length of paths in the simplified graph. An edge between a surviving shared saddle and an extremum is

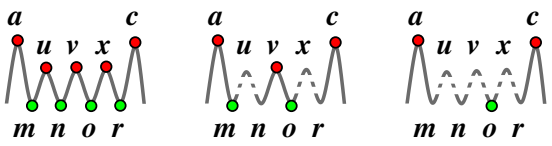
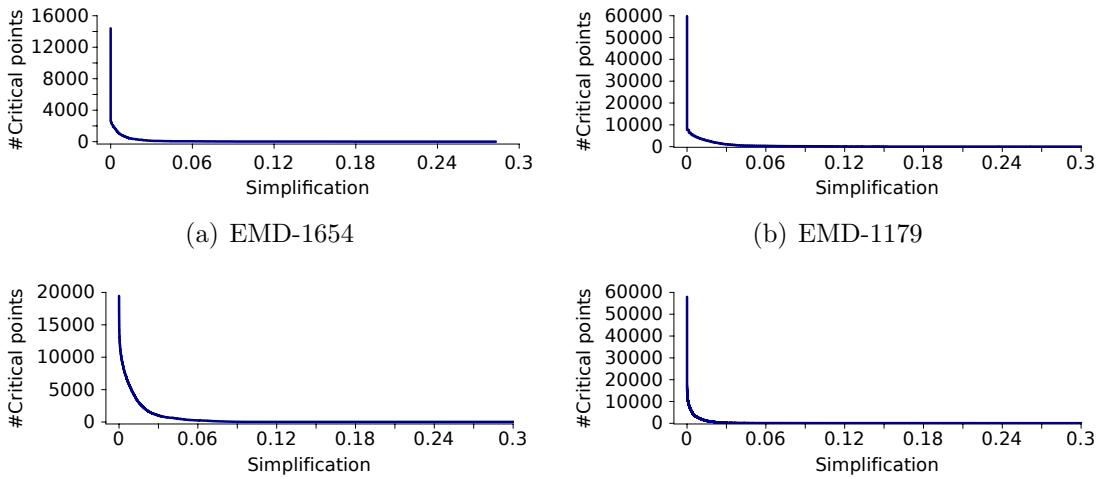


Figure 5.6: Bookkeeping of distance during simplification. (left) An extremum graph where the extrema are shown in red and the shared saddles in green. (center) During simplification, u merges into v and x merges into c. The surviving saddle m stores the length of the path between m and n. Similarly, o stores the length of the path between o and r. (right) When m is canceled, o stores the length of the path between m and r.

the result of zero or more simplifications. The shared saddle of the extremum in the unsimplified graph that lies on the path to the surviving saddle is called the *originating saddle*. To keep track of edges removed during simplification, each surviving saddle stores the originating saddles of the extrema that it is adjacent to and the length of the path between the originating saddles. Consider the extremum graph shown on the left in Figure 5.6. During simplification, let the cell u merge with v, canceling the saddle n and the cell x merge with c, canceling the saddle r as shown in the middle figure. The surviving saddle m is adjacent to a and v and stores their originating saddles, m and n. Similarly, o stores the originating saddles of v and c, namely o and r. To ensure that distances are calculated correctly, each surviving saddle also stores the distance between its two originating saddles. Before simplification, this distance stored at each saddle is zero. When a saddle is canceled, the distance stored at the surviving saddle is updated to equal the sum of the distance stored at the canceled saddle, the distance stored at the surviving saddle, and the length of the edge connecting the two originating saddles of the cell that gets merged. Before n is canceled, the distance stored by the surviving saddle m and the canceled saddle n is zero. After canceling n, the distance stored by m is the length of the edge m-n that bypasses the extremum u, calculated as the Euclidean distance between m and n. Similarly, after r is canceled, the surviving saddle o stores the distance between o and r. Eventually, after m is canceled, the surviving saddle o stores m and r as its originating saddles and the distance stored at o is equal to the sum of the distance stored at o (length of the path o-r), the distance stored at the canceled saddle m (the length of the path m-n), and the distance of the edge no, which connects the originating saddles of v. Thus the evaluation of distances remains the same in the unsimplified graph and the simplified graph.

To determine the simplification threshold, we plot the drop in the number of critical points against the simplification parameter which is the persistence normalized to lie between 0 and 1. This plot for simplification is shown in Figure 5.7 for four data sets EMDB-1654, EMDB-1179, EMDB-1603, and EMDB-5331. A volume rendered image of EMDB-1654 is shown in Figure 5.2 and volume rendered images of the remaining data

sets are shown in Figure 5.14. A similar plot is obtained for simplification with respect to the size of the Morse cells. In both cases, we see that initially there is a sudden drop in the number of critical points due to noise in the data. We set the thresholds to the value at which the drop in the number of critical points stabilizes, which can be identified as the beginning of the horizontal section in the plot. Later stages of the pipeline and the results are not sensitive to the exact value of the threshold chosen because the bookkeeping procedure described above ensures that the path length computed is not affected by the simplification. The main purpose of simplification is to improve efficiency of graph traversal by removing spurious critical points introduced by noise in the data. A simplification threshold of 1% of the maximum persistence or size suffices in most cases.



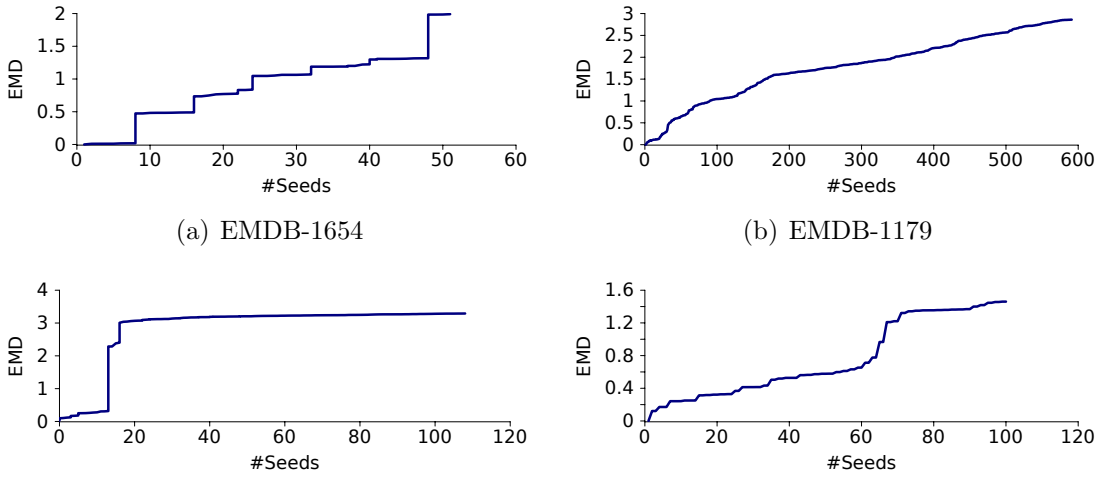
(a) EMD-1654

(b) EMD-1179

(c) EMD-1603

(d) EMD-5331

Figure 5.7: Plot of the number of critical points with increasing values of the simplification parameter for four data sets EMDB-1654, EMDB-1179, EMDB-1603, and EMDB-5331. The value at which the drop in the number of critical points stabilizes is used as the threshold for simplification.



(a) EMDB-1654

(b) EMDB-1179

(c) EMDB-1603

(d) EMDB-5331

Figure 5.8: Plot of the increase in the number of seeds with increase in EMD threshold for four data sets EMDB-1654, EMDB-1179, EMDB-1603, and EMDB-5331. For data sets other than EMDB-1179, initially, the EMD of the seeds is nearly zero after which there is a significant increase in the EMD. The value just before the sudden increase in EMD can be used to identify seed sets. For the data set EMDB-1179, the plot does not show such a sudden increase and hence the seeds are identified using a semi-automatic procedure.

5.2.2 Seed Set Selection

Selection of good seed sets is crucial for meaningful symmetries to be detected by our algorithm. Ideally, seeds should be chosen such that they are symmetric and representative of the symmetry in the input. Automatic selection of meaningful features from a data set is a challenging problem and though solutions may be designed for specific cases, a generic approach that works across data sets is not known. Hence, we do not attempt to find a generic solution to the related problem of automatic seed selection. For typical cryo-EM data sets, the histogram of the scalar values of symmetric seeds cells are very similar. We use a heuristic procedure based on histogram similarity for selecting seed sets.

We initially choose a Morse cell based on its attributes like size, persistence, and the function value of the extrema. In all our experiments, we select the Morse cell that contains the extrema with the highest function value. We then compute the histogram of

each Morse cell by uniformly dividing the range of function values into thirty bins. The histogram is then normalized and the Earth Mover's Distance (EMD) [72, 81] between the histograms of the chosen Morse cell and all other cells is computed. Those cells whose EMD falls below a threshold together with the initially chosen cell form a seed set. To determine the threshold, we plot the number of seeds added to the seed set as the EMD threshold is increased. This plot is shown for four data sets EMDB-1654, EMDB-1179, EMDB-1603, and EMDB-5331 in Figure 5.8. For clarity of illustration, we show only the initial part of the plot. For data sets other than EMDB-1179, initially, the number of seeds increase even for a very small increase in the EMD threshold as shown by the leftmost horizontal section in the plot. Further increase in the number of seeds occurs only after a significant increase in the EMD threshold as indicated by the vertical section in the plot. We can identify a meaningful seed set by setting the threshold to the value just before this jump in the EMD threshold. For the data set shown in Figure 5.2, eight seeds are identified using this procedure. For cryo-EM data sets that we use in our experiments, we have empirically determined that the threshold of 0.9 results in a good selection of seed sets in many cases.

The histogram matching procedure is a heuristic for selecting seeds and may not always give the desired result. For example, in the case of the data set EMDB-1179, the Morse cells corresponding to the seeds show considerable variation in their geometry and as a result histogram comparison is not meaningful. We employ the following semi-automatic procedure for selecting seeds which performs well in practice and does not require significant effort from the user. The selection procedure uses the toporrery layout, which is in turn based on the branch decomposition representation of the contour tree, and provides a powerful user interface for exploring and selecting features from scalar field data sets [69]. A user identifies a seed (extremum) of interest either based on its attributes (such as persistence) or visual examination of the features in the data. The branches in the toporrery layout corresponding to the extrema with similar function values are then automatically highlighted. The user examines the geometry of the level sets of the highlighted branches. The user needs to examine only a few level sets in

the neighborhood of the scalar value at the extrema. The user then selects a subset of the branches that exhibit symmetry in the evolution of the level sets. The extrema of the branches selected by the user are chosen as the seed set. The above procedure automatically prunes away a majority of branches in the branch decomposition layout and limits user interaction to selecting a subset of branches from the remaining small set of branches. An illustration of this procedure is shown in Figure 5.9.

Figure 5.9: Seed selection using the contour tree. (left) User identifies an extremum of interest and selects the branch corresponding to it from the toporrery layout. The selected branch is shown in orange and other branches whose extrema have function values similar to extrema of the selected branch are automatically selected and shown in blue. (middle) User inspects the level sets of the selected branches. Among the sixteen branches selected, the evolution of the level sets corresponding to eight of the branches are similar and (right) the user selects this subset of branches shown in orange. The extrema corresponding to the selected branches are chosen as the seed set.

5.2.3 Augmented Extremum Graph Traversal

Given an initial set of seeds, an efficient procedure for the augmented extremum graph traversal is employed both to form super-seeds and to expand the initial estimate of symmetry through region growing. Each seed is marked as a source vertex and traversals are initiated simultaneously and independently from each source vertex, as shown in Algorithm 5.1. Each traversal proceeds iteratively, where at each step, the saddle that is closest to the seed is processed. Only the source and destination extrema are contained in a path and the rest of the extrema are bypassed. The function len(i,j) calculates

Algorithm 5.1 Augmented extremum graph traversal algorithm. For a Morse cell i, its extremum is denoted by e_i. The shared saddle between i and j is denoted by $sad_{i,j}$. A path is a 3-tuple (*src*, *des*, *sad*), where *src* is the source seed and *des* is the cell adjacent to *sad*, the saddle at the end of the path.

Input: Extremum graph with selected seeds
Output: Seed-merge tree
foreach *seed* **do** `// initialize pq`
  j = cell_id(seed)
  foreach *cell_id k adjacent to j* **do**
    path = (j,k, $sad_{j,k}$)
    path.len = len(e_j, $sad_{j,k}$))
    pq.push(path)
  end
end
while *not all seeds merged* **do** `// traverse`
  p = pq.pop()
  if *not visited*[*p.src*, *p.sad*] **then**
    visited[p.src, p.sad] = true
    foreach *cell_id i adjacent to p.des* **do** `// extend`
      ext_path = (p.src, i, $sad_{p.des,i}$)
      ext_path.len = p.len+len(p.sad, $sad_{p.des,i}$))
      pq.push(ext_path)
    end
    des_len = p.len + len(p.sad, $e_{p.des}$)
    if *dist*[*p.src*, *p.des*] > *des_len* **then**
      dist[p.src, p.des] = des_len `// update`
      if *isseed(p.des)* **then** `// merge seeds`
        merge(p.src, p.des, des_len);
      end
    end
  end
end

the length of the edge from the node i to node j in the graph. To store distances from multiple seeds, each Morse cell maintains an array of distances. This is denoted by dist[e_i, e_j] which stores the length of the shortest path from the extremum e_i to the extremum e_j. The output of the algorithm is a binary tree, called *seed-merge tree*, that represents the merging of seeds. The function merge(*src*, *des*, *lenght*) generates a new internal node with weight *length* in the seed-merge tree that represents the merger of the connected component associated with the seed *src* with that of the seed *des*. This is explained in Section 5.2.4. Similarly, each saddle maintains an array that indicates if the saddle has

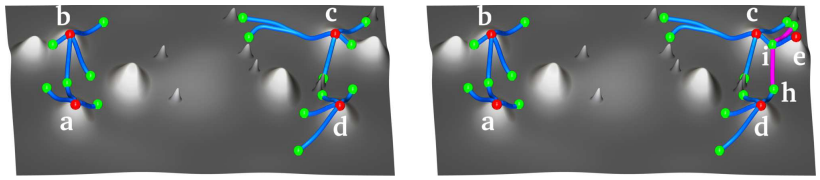


Figure 5.10: Augmented extremum graph traversal. (left) Blue edges from the seeds a, b, c, and d to their shared saddles are inserted into a priority queue. (right) The shortest path c-i is popped out and extended to the adjacent saddles by adding the magenta edges. When the path c-i-h is popped out, the traversal from the seed c reaches the seed d and the two seeds merge.

been visited from a seed.

Consider the surface plot of a scalar function as shown in Figure 5.10. Let the mountain peaks a, b, c, and d be chosen as seeds. Initially, the edges from each seed to its adjacent shared saddles, shown on the left, are inserted into a common priority queue. The cost of each edge is the distance from the seed to the shared saddle. After this initialization step, the edge with the shortest distance, say ci, is popped from the queue. Next, the distance from the source seed c to the extremum e adjacent to the endpoint saddle i in the path is computed as the sum of the length of the edge ei and the length of the edge ci which was popped out, see the figure on the right. The popped path c-i is extended to the unvisited shared saddles of the adjacent extremum e. When the path is extended to h, the extremum e is bypassed and the shared saddles are directly connected. The length of the extended path is calculated as the sum of the length of the saddle-saddle direct edge and the length of the path popped from the queue. The extended path is inserted into the queue and the traversal proceeds to visit other saddles in the order of increasing distance from the seeds. When a saddle is visited from a seed, the distance from the seed to the adjacent extremum is updated and is later used during the region growing stage. When the path c-i-h is eventually popped out of the queue, the path from the source seed c reaches the seed d and the two seeds merge. The traversal continues until all the seeds merge.

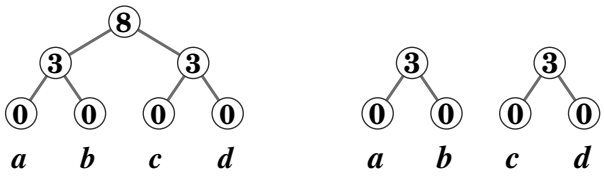
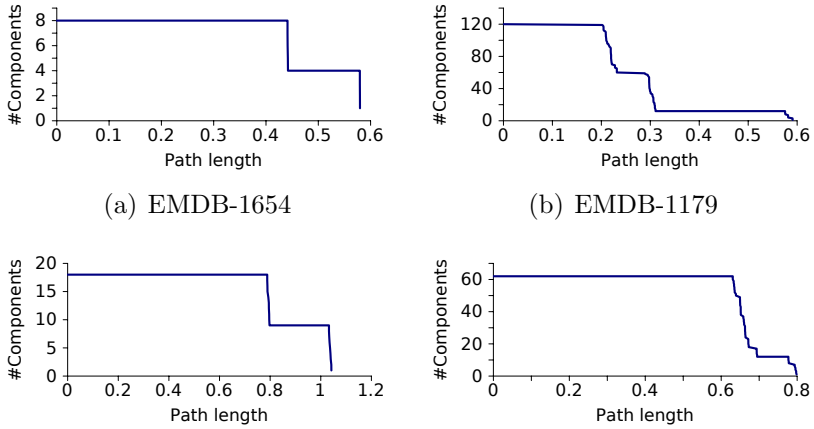


Figure 5.11: (left) Labels on the interior nodes of the seed-merge tree is equal to the distance at which its children nodes merge. (right) Nodes with values above a threshold are removed from the tree and the seeds that remain connected belong to a common super-seed.

5.2.4 Super-seed Formation

Studies on the way humans perceive patterns, like Gestalt theory [107], have shown that items that are located spatially near each other are perceived to be part of a group. Repeating structures in close proximity are grouped together and perceived to be part of bigger patterns. Thus, for symmetry detection it is important to identify the formation and repetition of such bigger patterns formed from smaller patterns. In our context, seeds that are closer merge first and form a hierarchy of bigger patterns. A super-seed is the set of seeds that merge to form the biggest such repeating pattern. The above description of the way humans perceive patterns also explains our assumption on distances between symmetric regions as stated in Section 5.1.2. When the path from a source seed reaches another seed, the two seeds merge and the distance between the two seeds is the length of the shortest path between the two seeds. At the leaf level of the hierarchy, all the seeds, a, b, c, and d, in Figure 5.10, are separate repeating units. During the traversal, a merges with b and similarly c merges with d. These merged components form the next level in the hierarchy and they further merge into a single component at the root level. A graph can be constructed by inserting a node for each seed and a weighted edge between two seeds that have merged. The edge weight is equal to the distance between the two seeds. After all the seeds merge, a graph cut that removes edges with high weights is used to identify the super-seeds.

Our graph cut algorithm uses the seed-merge tree that represents the merging of seeds. Each of the seeds a, b, c, and d is a leaf node of the seed-merge tree and represents a component. When the components corresponding to two nodes merge, an interior parent



(a) EMDB-1654 (b) EMDB-1179

(c) EMDB-1603 (d) EMDB-5331

Figure 5.12: Plot of the number of components against the path length. The threshold for disconnecting the seed-merge tree is chosen from the rightmost horizontal interval, just before the number of components drop to one.

node is created to represent merging of the components. Each interior node is assigned a weight equal to the distance at which the corresponding components merge in the augmented extremum graph traversal, see Figure 5.11. All leaf nodes are assigned zero weight. Once all the seeds merge and the complete tree is constructed, we disconnect it into different subtrees by removing nodes with weight above a threshold. Thus we disconnect the seeds that merge during the augmented extremum graph traversal by means of paths whose length is above the threshold and the seeds that remain connected are identified as super-seeds. Figure 5.11 shows two disconnected subtrees on the right formed when the threshold is set to eight. One of the subtrees corresponds to the super-seed comprising of a and b and the other subtree corresponds to the super-seed comprising of c and d.

To determine a suitable value for this threshold, we plot the decrease in the number of components with respect to increase in the path length. Figure 5.12 shows this plot for four data sets EMDB-1654, EMDB-1179, EMDB-1603, and EMDB-5331. Each of the vertical sections in the step-like plot show the sudden drop in the number of components due to the merging of the components. The merging happens at different scales and

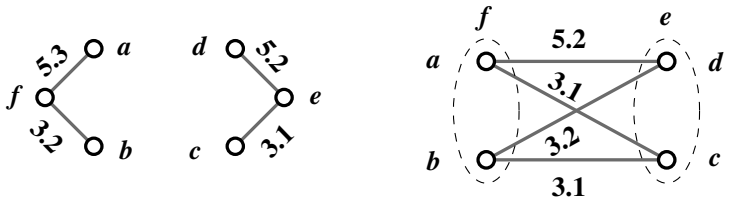


Figure 5.13: (left) The distance from the candidate cells f and e to the seeds a, b and c, d are shown by the labels on the edges. (right) An edge connecting two seeds in the bipartite graph is weighted with the minimum of the distances to the seeds. The sum of weights of matching edges ad and bc, 8.3, is close to the average of the distances to the seeds, (5.3+3.2+5.2+3.1)/2 = 8.4 and hence f and e are considered to be symmetric.

reflects the hierarchical relationship in the merging of the seeds. The number of components at the start of the horizontal sections in the plot remains unchanged till the end of the interval and shows the stability of the components in this section. The super-seeds are the largest stable components formed just before all the seeds merge into a single component. Hence, the threshold for identifying the super-seeds is chosen as any value that lies in the last horizontal section in the plot just before the number of components drop to one. For example, in the case of EMDB-1654, eight seeds were chosen and based on the plot we identify four super-seeds and these are shown using four distinct colors in the middle figure in Figure 5.2. As described in Section 5.1.2, we assume that the distance at which the seeds within a super-seed merge is significantly lower compared to the distance at which seeds merge across super-seeds. This separation of distances will manifest as a distinct horizontal section in the plot used to identify the super-seeds and can be easily determined.

5.2.5 Symmetry Expansion

Once the super-seeds are formed, its symmetry is expanded by inserting neighboring Morse cells through a region growing procedure. For detecting partial symmetry, when a Morse cell is considered as a candidate to be inserted into its closest super-seed, we ensure that a symmetric candidate exists in the remaining super-seeds. Consider candidate cells f and e and the distances to the seeds that constitute their closest super-seeds, $\{a, b\}$ and $\{c, d\}$ as shown on the left in Figure 5.13. If f and e are symmetric, then the

distance from f to a seed in its closest super-seed, say a, will be nearly equal to the distance from e to the corresponding symmetric seed, in this case d, and the minimum of the two distances will be nearly equal to the average of the distances. When the distances from f to each seed in its super-seed satisfy this property, we consider the distribution of distances from f and e to be similar and treat them to be symmetric. To determine this, we construct a complete bipartite graph as shown on the right in Figure 5.13, where the nodes in each partition are the seeds that constitute the closest super-seed and an edge between two seeds is weighted with the minimum of the distances from f and e to the respective seeds. Next, we compute a maximum weight matching and if the weight of the matching is close to the average of the distances, we consider f and e to be symmetric. Based on empirical results, we set this threshold to 80% of the average distance to the seeds. The ideal threshold depends on the noise in the data and selecting the threshold involves a trade-off between tolerance to noise and quality of the detected symmetry. If each super-seed has a symmetric candidate, then these candidates are inserted into the symmetric region of their super-seeds and the regions are expanded. The procedure is repeated till all candidate cells visited during the graph traversal are considered. For noisy data, the farther we traverse away from the seed cells, the distance estimate accumulates errors as noise causes variations in the shortest path estimate. This makes it difficult to compare distance distribution for detecting partial symmetry. However, for identifying global symmetry, the symmetry verification at each step can be avoided since, by definition, each candidate cell has a symmetric counterpart in all the super-seeds. For global symmetry, we continue the graph traversal, even after the seeds merge, till all the cells are visited by at least one of the seeds. The symmetric region corresponding to a super-seed is then reported as the set of all cells that are closest to the super-seed.

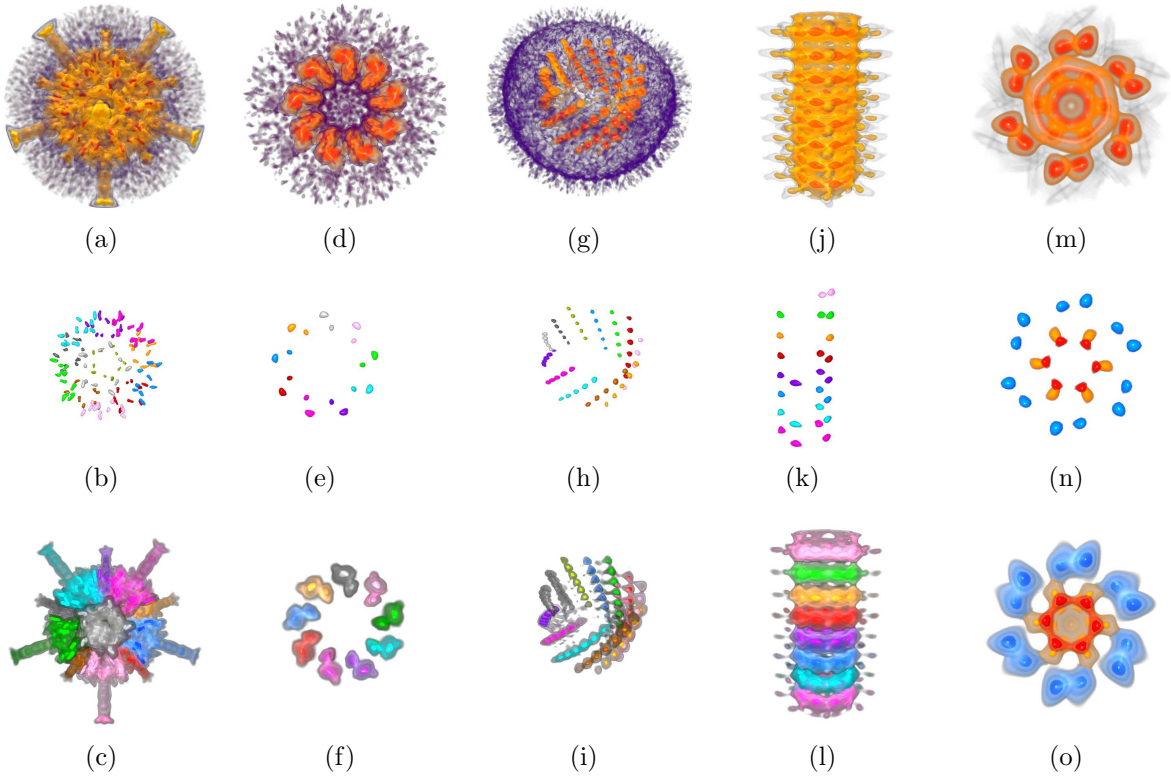


Figure 5.14: Results of symmetry analysis on cryo-EM data sets. Five data sets - EMDB 1179, 1603, 5331, 2094, and 1706 are shown in the first row. EMDB 1179, 1603, and 5331 contain significant noise as shown by the violet regions in the volume rendered images. The seeds selected are shown in the second row. The extracted symmetries are shown in the third row. Symmetries are extracted even in the presence of noise. Columns 1-4 show dodecahedral, 9-fold rotational, screw, and translational symmetries and the rightmost column shows partial symmetry extraction.

5.3 Results and Discussion

We now present experimental results of our symmetry detection algorithm run on different cryo-EM data sets. Cryo-EM is a form of electron microscopy performed at extremely low temperatures and is a popular method for imaging biomolecules. The electron density map is represented as a 3D structured grid. All cryo-EM data sets used in this thesis are obtained from EMDataBank [34], which is an online public repository containing cryo-EM data sets.

5.3.1 Global and Partial Symmetry

The first row in Figure 5.14 shows volume rendered images of five cryo-EM data sets. The volume rendering shows the lower function values in dark violet color and they correspond to noise in the data. The first four columns show global symmetry extracted by our algorithm and the fifth column shows an example of different partial symmetries extracted. For all data sets, except the first, seed selection was done based on histogram matching.

Figure 5.14(a) shows a data set with dodecahedral symmetry detected using 120 seeds. The Morse cells corresponding to the seeds show considerable variation in their geometry and as a result histogram comparison is not meaningful in this case. Hence, seeds were selected semi-automatically based on the procedure described in Section 5.2.2. We choose 120 maxima with the highest function values and the selected seeds merge to form twelve super-seeds. The seeds belonging to a common super-seed are shown with the same color in Figure 5.14(b). The extracted symmetric regions after region growing are shown in Figure 5.14(c). Though only ten small Morse cells are selected per symmetric region, the region growing stage correctly identifies symmetry of the remaining Morse cells even though some of these cells in the long fiber-like structures are far away from the seeds at the base. The second column shows 9-fold rotational symmetry detected by our algorithm after selecting eighteen seeds that merge to form nine super-seeds. The third column shows a data set with twelve repeating strand-like structures arranged helically and our algorithm detects screw symmetry of the repeating strands. Figure 5.2 shows another example where our algorithm is able to detect 4-fold rotational symmetry in the data using eight seeds which merge to form four super-seeds. The volume rendering of all these data sets indicate that there is significant amount of noise in the data. However, due to the robustness of the technique we are able to identify the symmetry present in the data. The fourth column shows a data set with translational symmetry detected by the algorithm. Three different partial symmetries extracted from the data set in Figure 5.14(m) are shown in Figure 5.14(o) as orange, blue, and red segments in a symmetry-aware segmentation. The seed sets used in each

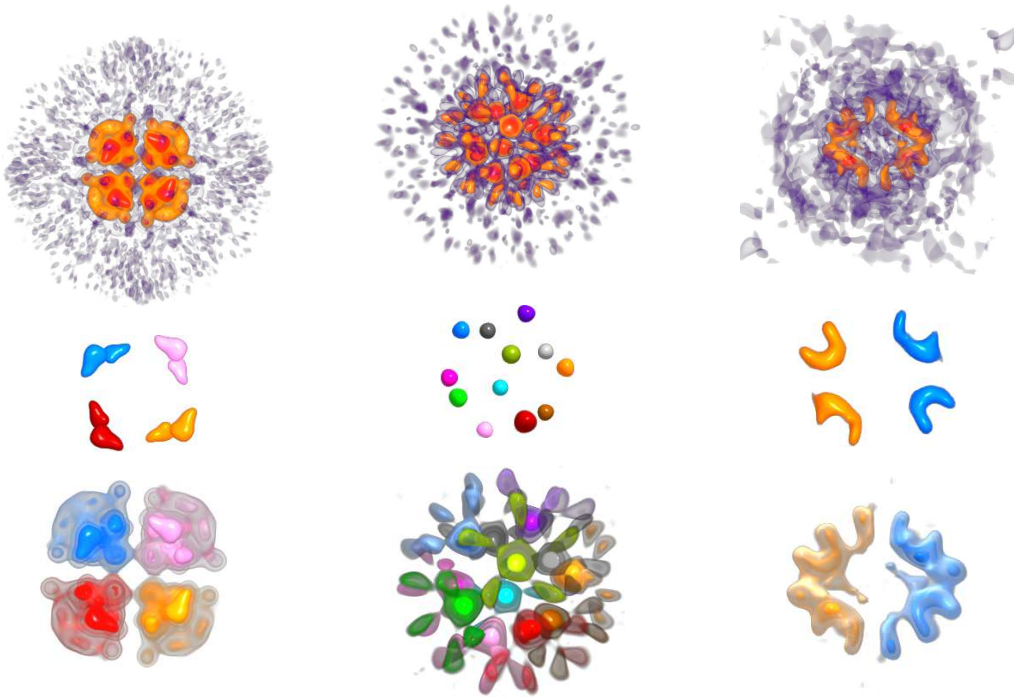


Figure 5.15: Symmetry detection results on additional cryo-electron microscopy data sets - EMDB 1319, 1659, and 5215. Each column shows a volume rendering of a data set, the seed set, and the symmetric regions detected by our method. Each symmetric region is shown in a unique color. The violet regions are representative of the noise in the data. The symmetry detection method works even in the presence of noise.

case is shown in Figure 5.14(n) with the same color as the color of the symmetry-aware segment. Figure 5.15 shows results on additional cryo-EM data sets. Figure 5.16 shows results on a synthetic data set with increasing noise levels.

5.3.2 Sensitivity to Seed Selection

Ideally, seeds should be chosen such that they are symmetric and representative of the symmetry in the input. However, our method can tolerate asymmetric distribution of seeds among the symmetric regions. Consider an ideal set of seeds and the assignment of each Morse cell to the seed that is closest to it. Erroneous omission of a seed from the ideal set affects the assignment of only those cells that were originally assigned to the omitted seed. Cells in the neighborhood of the omitted seed are reassigned to the remaining seeds while other cells are unaffected. Similarly, insertion of a new seed into the ideal set

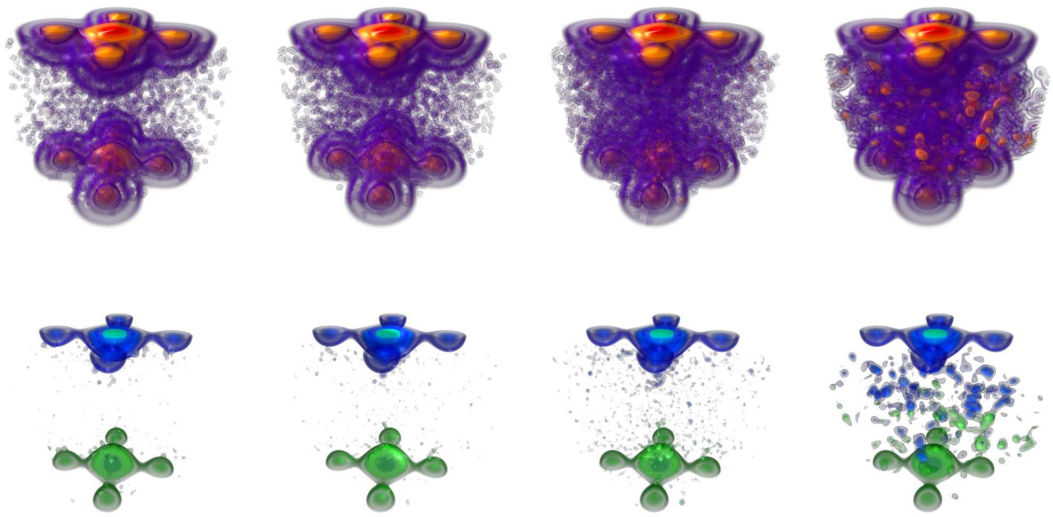


Figure 5.16: Our symmetry detection method is robust in the presence of noise. (top) A synthetic data set with increasing levels of noise is shown from left to right. The transfer function used is identical in each case and shows the increasing levels of noise. (bottom) The symmetric regions detected are shown in blue and green in the bottom row.

affects the assignment of only those Morse cells that belong to the neighborhood of the new seed. Also note that the distance between a pair of seeds is calculated independent of the remaining seeds. Thus, deletion or insertion of seeds causes changes only in the local neighborhood of the seeds. We illustrate this for the data set shown in Figure 5.2 by modifying the ideal set of two seeds per super-seed. When one seed is dropped, the blue super-seed is formed with only one seed. However, the symmetric regions detected are unaffected as shown in the left column of Figure 5.17 since many of the Morse cells that were closest to the dropped seed are now closest to the remaining blue seed. In the extreme case, when two seeds are dropped resulting in only three super-seeds, the Morse cells that were assigned to the blue seeds are reassigned to the remaining closest seeds - pink and red as shown in the middle column. Note that the symmetric region corresponding to the orange super-seed is unaffected since dropping of seeds only causes local changes to the detected symmetry. When two new seeds are inserted, the blue super-seed consists of four seeds as shown in the right column. The Morse cells assigned to the other super-seeds are unaffected and thus the symmetric regions detected are

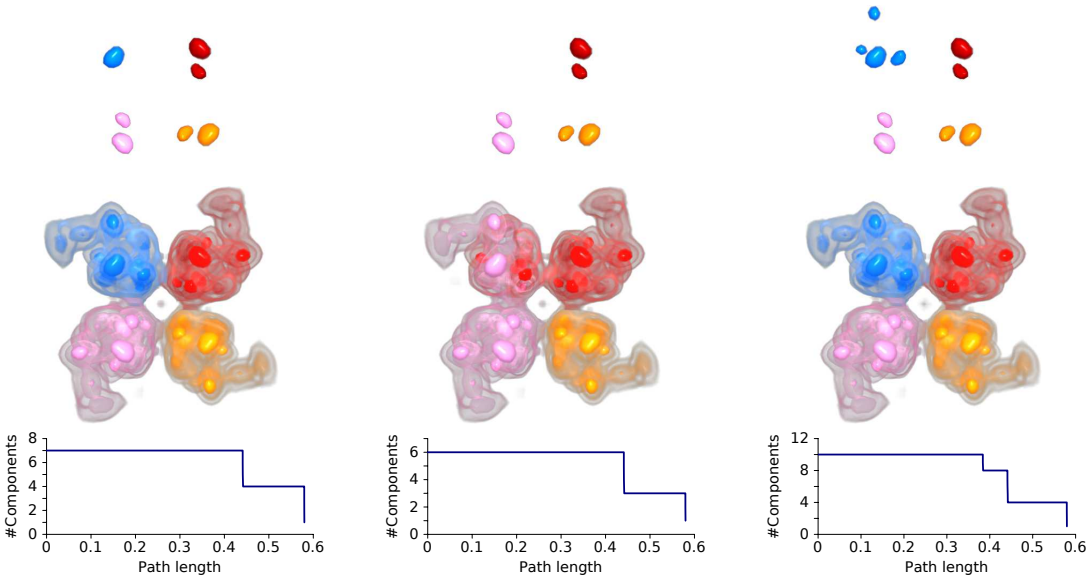


Figure 5.17: Sensitivity to seed set selection. (left) When one seed is dropped, the symmetry detected is unaffected. (center) When the blue super-seed is dropped by removing both of its seeds, the Morse cells which were closest to the blue super-seed are reassigned to the remaining super-seeds. (right) When two new seeds are inserted, the symmetry detected is again unaffected. The plot for identifying super-seed is shown for each case in the bottom row.

again not affected. For each of these cases, the plot used to identify the super-seeds is similar to the plot shown in Figure 5.12 and is shown in the bottom row of Figure 5.17. The fourth column in Figure 5.14 shows another example where each symmetric region is a ring with 9-fold rotational symmetry. Ideally, nine seeds from each ring should have been selected as the seed set. However, due to variations in the histogram, the algorithm selects two, three, or four seeds from the rings as shown in Figure 5.14(k). Though the seed distribution is inconsistent, symmetry can be detected as long as the seeds chosen are such that the distance between seeds belonging to different symmetric regions is higher compared to the distance between seeds within the same symmetric region. This ensures that seeds within a symmetric region merge first and form super-seeds that represent the symmetry in the data. Eight super-seeds are formed in this case corresponding to the translational symmetry and the symmetric regions are correctly identified as shown in Figure 5.14(l). Results on more data sets are shown in Figure 5.18, Figure 5.19, and Figure 5.20.

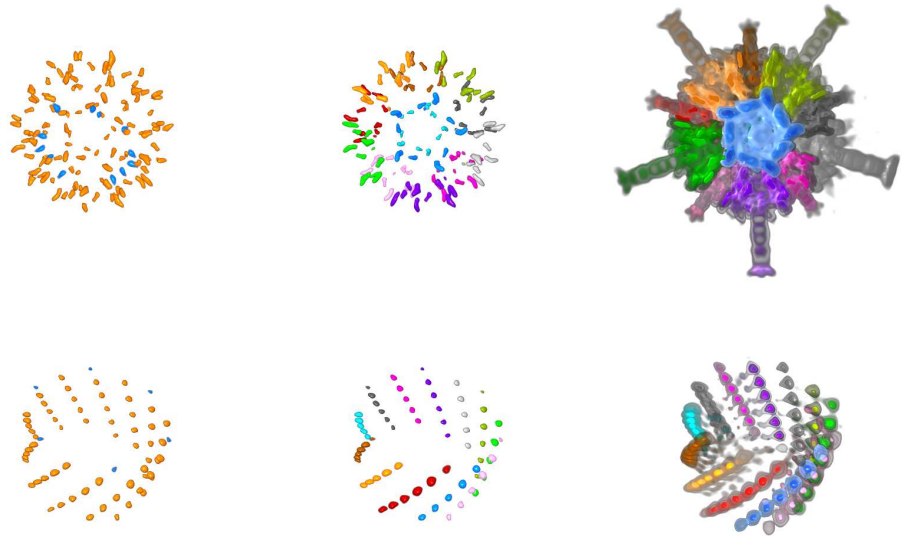


Figure 5.18: Effect of insertion of seeds to the ideal seed set for two data sets shown in the top and bottom rows. (left) New seeds shown in blue are inserted into the ideal seed set shown in orange. (middle) Despite the asymmetric distribution of seeds, they form super-seeds that represent the symmetry in the data and (right) the symmetric regions are detected correctly.

5.3.3 Comparison with the Contour Tree-based Method

The symmetry detection method based on extremum graph uses Morse decomposition, which is based on gradient flow topology as compared to the earlier approach that uses the contour tree, which is based on level set topology. Therefore, the symmetric segments identified by the two methods may be different and it is not meaningful to compare the results of the two methods. So, we qualitatively evaluate the two methods and list the pros and cons of both approaches.

The contour tree-based method detects symmetry by identifying similar subtrees from the branch decomposition representation of the contour tree. Since branches corresponding to noise in the data can destroy the similarity of the subtrees, this method requires the removal of such branches. For this purpose, the branch decomposition is simplified by removing low persistence branches under the assumption that only low amplitude noise

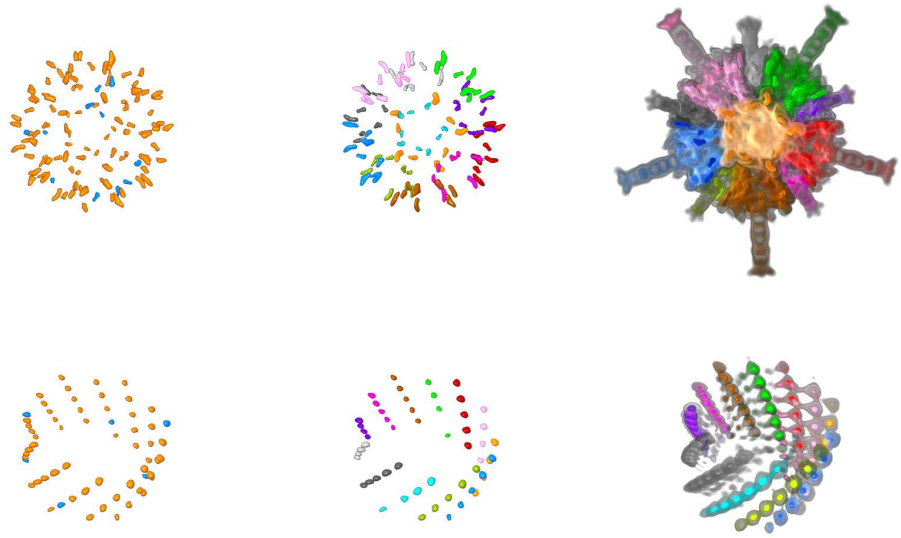


Figure 5.19: Effect of deletion of seeds from the ideal seed set for two data sets shown in the top and bottom rows. (left) Seeds shown in blue are removed from the ideal seed set and the remaining seeds are shown in orange. (middle) Despite removal of seeds, the super-seeds are formed correctly and (right) the symmetric regions are detected.

exists in the data. Though our method also simplifies the extremum graph for computational efficiency, in contrast to the assumption made by the contour tree-based method, we do not necessarily require that the noise is removed through the simplification step. This is because the presence of noise does not affect the distance calculation and hence our method can handle noise of larger amplitude in the data. Moreover, our method incorporates geometric information for more effective symmetry detection and also uses a region growing procedure to identify the largest symmetric region. In comparison, the contour tree-based method ignores geometric information and does not necessarily identify the largest symmetric region. We illustrate these advantages with a real-world data set in Figure 5.21.

One of the major limitations of our method is that the symmetry detected depends on the choice of seed sets used and selection of seed sets may require user interaction unlike the contour tree-based method. The contour tree-based method is well suited for identifying partial symmetry as opposed to our method which requires different seed

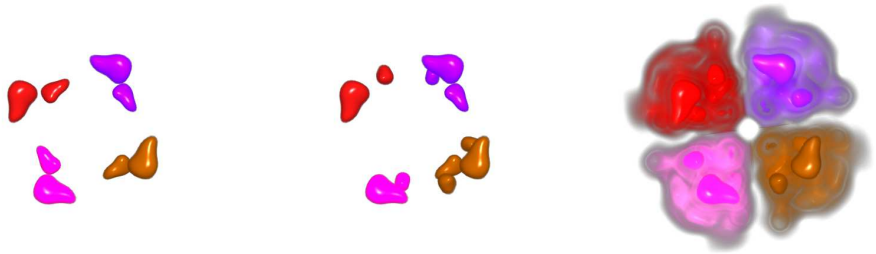
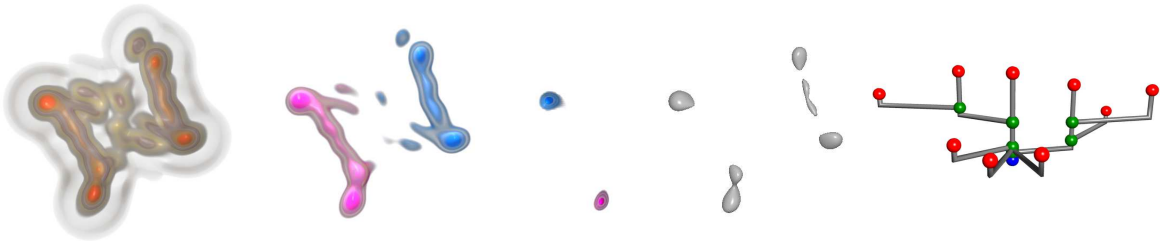


Figure 5.20: Effect of both insertion and deletion of seeds from the ideal seed set. (left) From the ideal seed set, (middle) five new seeds are inserted and two seeds are removed. The super-seeds formed represent the symmetry in the data and (right) the symmetric regions are detected correctly.

sets to be identified for each partial symmetry in the domain. The contour tree-based method also has the advantage that it can detect symmetry at multiple scales since the branch decomposition representation induces a natural hierarchy on the branches. Figure 5.22 shows additional results of symmetry detected by our method on data sets used in the contour tree-based symmetry detection method. It can be seen that our method is limited to detecting symmetries at the largest scale whereas the contour tree-based method can detect different partial symmetries as well as symmetries at different scales.

5.3.4 Performance

Table 5.1 reports the running time of our algorithm for the data sets shown in Figure 5.14. The time taken for building the graph includes computation of the Morse decomposition as well as the simplification of the initial extremum graph constructed. Earth Mover's Distance computation is fairly fast in practice. When the number of seeds is large, graph traversal dominates the computational cost of the algorithm since the saddles are visited multiple times by the graph traversal initiated from each seed. In the current implementation, the weights of the edges are computed on the fly during the graph traversal. The augmented edges that directly connect the shared saddles are not present in the extremum graph and their weights have to be computed during the traversal. This



(a) (b) (c) (d) (e)

Figure 5.21: A real-world data set illustrating improved symmetry detection using our method as compared to the contour tree-based method. (a) A data set (EMDB 5214) with 2-fold rotational symmetry. (b) Our method identifies the global symmetry whereas (c) symmetry detection using contour tree can only identify partial symmetry of two smaller regions. (d) Even though the data set is symmetric, the level-set components evolve differently and hence (e) the branch decomposition representation of the symmetric regions are also different. In the presence of significant noise, symmetry detection using contour trees perform poorly since the symmetric regions do not manifest as repeating subtrees in the branch decomposition representation. Our method, on the other hand, is not affected by the asymmetric evolution of the level set components since their geometric proximity is captured by the augmented extremum graph and the graph cut correctly partitions them into two symmetric regions.

additional computation increases the time spent for the graph traversal. We believe a more efficient implementation of the algorithm can significantly improve the running time. The running time does not include time for I/O.

5.4 Conclusions

In this chapter, we present an integrated geometric and topological approach for detecting symmetric regions in a scalar field. We believe that our method is a significant improvement over existing methods since it can robustly detect symmetry even in the presence of significant noise. The proposed method is computationally efficient. We show through experiments that our algorithm can detect symmetry under different types of transformations in real-world data sets.

Perhaps the biggest limitation of our method is that the symmetry detection critically depends on the selection of a meaningful set of seeds. A bad selection of seeds may lead

Table 5.1: Running time, measured in seconds, for various steps in the symmetry detection pipeline. All experiments were performed on a 2 GHz Intel Xeon processor with 8GB RAM.

EMDB data set#	#vertices	build graph	EMD computation	traverse graph
2094	100^3	5.4	1.5	3.2
1654	112^3	7.6	3.3	1.1
1706	130^3	8.5	0.04	0.5
1603	160^3	34.2	8.9	45.4
5331	240^3	106.5	19.8	77.6
1179	255^3	106.9	-	409.1

to incorrect formation of super-seeds, which in turn affects the quality of the detected symmetries. Though we describe the method used for seed selection, a robust and widely applicable method for automatic selection of seeds remains an open problem. One possible approach is to automate the seed selection procedure we describe in Section 5.2.2 using similarity of the level sets in the neighbourhood of the scalar value at extrema. In the next chapter, we pursue this idea further and develop a fully automatic symmetry detection method by clustering level set components based on their similarity. Another limitation of the current approach is that the simplification procedure is not symmetry-aware. An interesting open problem is to design a simplification method that ensures that the simplified Morse cells are appropriately distributed among the symmetric regions.

The limitations of our technique primarily arise from using only local information about symmetry. We believe that similar to methods that detect symmetry in geometric shapes [56, 64, 71, 74, 105], scalar field symmetry detection methods will also benefit from a clustering based analysis. Such an approach will help obtain more global information about the symmetry and may also lead to symmetry detection methods that are insensitive to missing regions and imperfections in the symmetry. In the next chapter, we describe a method based on such an approach for symmetry detection.

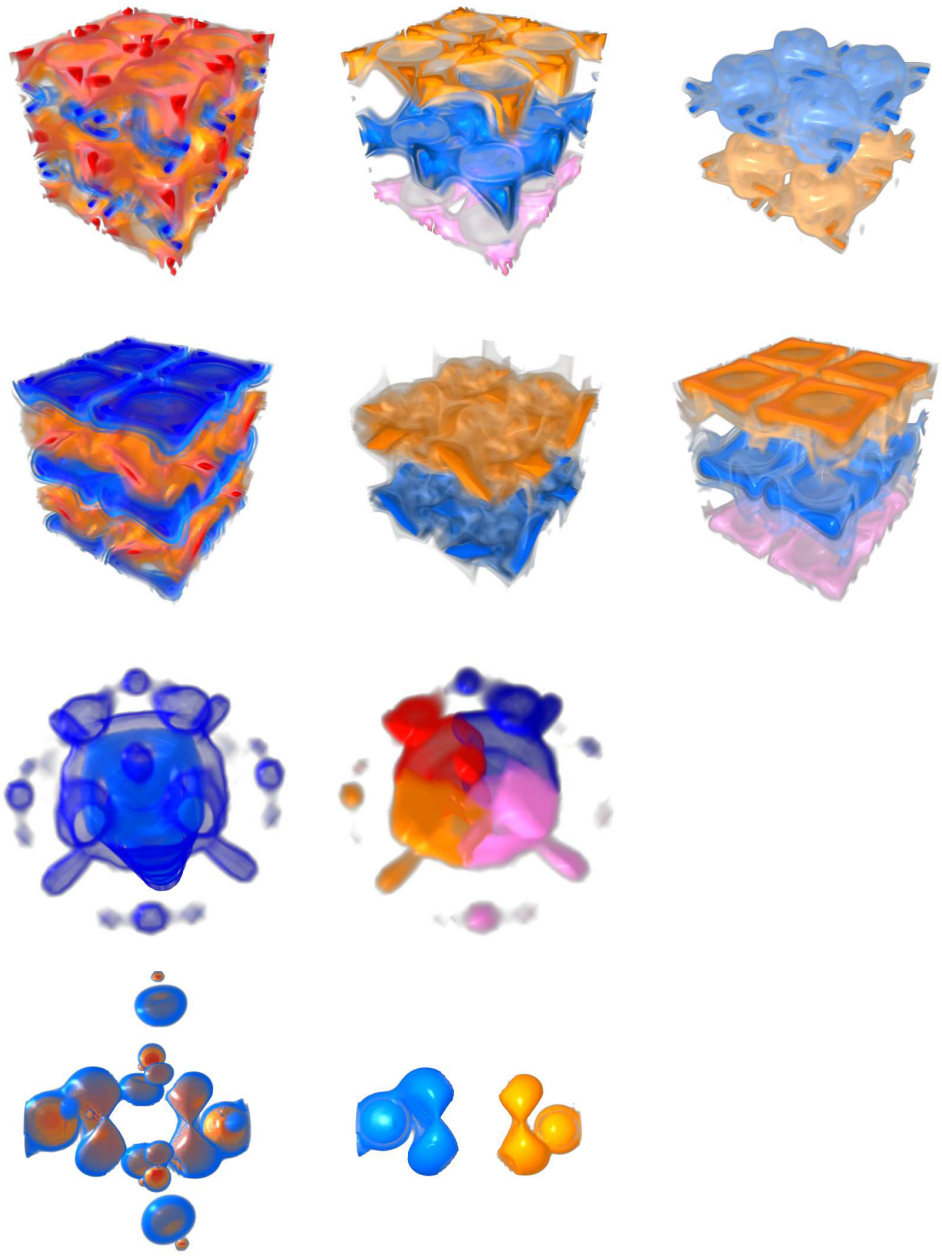


Figure 5.22: Disadvantages of the extremum graph-based method. First and second rows show volume rendering of the two vortex data sets and the symmetries detected by our method using the maximum graph and the minimum graph. Third and fourth row shows the fuel data set and the neghip data set and the partial symmetries detected in them using the maximum graph. Using augmented extremum graph, we can detect symmetry only at a single scale. On the other hand, the contour tree-based method can detect symmetries at multiple scales as well as different partial symmetries in the data.

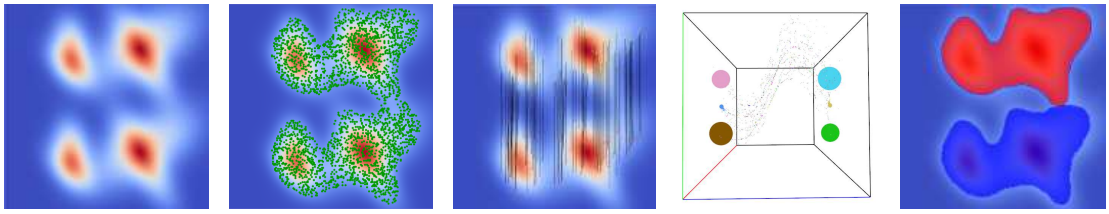
Chapter 6

Symmetry Detection Using Contour Clustering

The earlier methods we proposed compared candidate regions pairwise and used a similarity threshold to classify them into symmetric groups. Determining the similarity threshold is a challenge when using data sets with varying characteristics. Moreover, since the degree of symmetry exhibited is different for different symmetric regions, using a single global threshold is not always meaningful. In addition, these methods use abstract graph representations of the scalar fields for symmetry detection and use only limited geometric information in their analysis. Hence, they may not be suitable for applications that focus on the geometric properties of the symmetric regions in the domain.

The study of symmetry detection in shapes have established that clustering based analysis result in superior performance and robust identification of symmetry. These methods accumulate evidence of symmetry through geometric techniques and use a clustering stage to identify symmetric regions. We consider two approaches for accumulating symmetry information and clustering, namely, point based clustering and contour based clustering. Below, we briefly describe a method that uses the former approach for symmetry detection and the rest of this chapter describes in detail the latter approach.

Symmetry detection by clustering point pairs. This method extends an earlier work to detect symmetry in shapes by Mitra et al. [64] to scalar fields. A set of points



(a) (b) (c) (d) (e)

Figure 6.1: Illustration of symmetry detection by clustering point pairs on a synthetic data set. (a) The input scalar field. (b) Points sampled from the domain shown in green. (c) The sample points corresponding to the endpoints of each line segment are locally similar and hence paired together. For each pair, the transformation that maps the points in the pair is represented as a point in a transformation space. (d) The clusters in the transformation space are identified. (e) For each cluster, the corresponding symmetry transformation is spatially verified and the symmetric regions, shown in red and blue, are extracted.

are sampled from the domain and for each sample point a descriptor that captures the scalar field locally is computed. The descriptor consists of an invariant component and an alignment component. The invariant component of a point consists of properties of the field like scalar value, gradient magnitude, and curvature of the level set passing through the point. The alignment component of a point consists of vectors like the gradient vector and the principal curvature directions of the level set passing through the point. These vectors are used to define a local coordinate frame at each point. Sample points with similar invariant components are paired together and the transformation that maps the points in the pair is computed using their local coordinate frame. Since the scalar field is locally similar for the points in a pair, they are considered to be locally symmetric. Each pair then votes for the symmetry transformation that maps the points in the pair in the space of all transformations. Symmetry detection problem is then reduced to finding clusters in the high-dimensional transformation space. Each cluster corresponds to aggregation of votes for a particular symmetry transformation. These clusters are identified using a clustering technique. Finally, the symmetry transformation is spatially verified and the symmetric regions are extracted and reported. An illustration of this technique on a synthetic 2D scalar field data set is shown in Figure 6.1. Interested readers may refer to our work that uses point based clustering for more details [60].

Symmetry detection by clustering contours. Though it is clear from methods proposed in the geometry processing community that a clustering based analysis offers significant advantages in recognizing symmetry, we believe that unlike shapes, low-level information available at the sample points of the domain is not suited for symmetry identification in scalar fields. Scalar field data sets are typically represented using scalar values assigned to a discrete set of sample points that represent the domain under consideration. However, the domain and the scalar values are assumed to be continuous by interpolating the values at the sample points. Therefore, the sample points in a scalar field capture the lowest level of information. In practice, scientists are more interested in higher level features, extracted through methods like segmentation and isosurface extraction, for studying the underlying physical phenomena. Hence, symmetry identification methods that are based on local information available at the sample points encounter considerable difficulty in representing and extracting meaningful symmetric regions. Inspecting only local information at the sample points introduces additional challenges due to discretization errors since the symmetric counterpart for a given point may be an interpolated point. Moreover, these methods are computationally expensive since the number of sample points in scalar field data sets is typically orders of magnitude higher than that in geometric shape data sets.

In this chapter, we propose a novel symmetry detection method based on the idea of clustering contours. Isosurfaces are extensively used in studying scalar field data sets and contours, which are connected components of isosurfaces, capture information about a scalar field at a macroscale. Therefore, contours are more suitable for a clustering based analysis as opposed to sample points of the domain. It is easy to see that contours belonging to regions with symmetric scalar field distribution are also symmetric. Using an appropriate shape descriptor, our method maps contours to points in a descriptor space such that the distance between points in the descriptor space is a measure of similarity between the contours. As a result, points in the descriptor space representing symmetric contours lie in close proximity to each other and form clusters in the descriptor space. The region of the domain corresponding to each such contour can be extracted and these

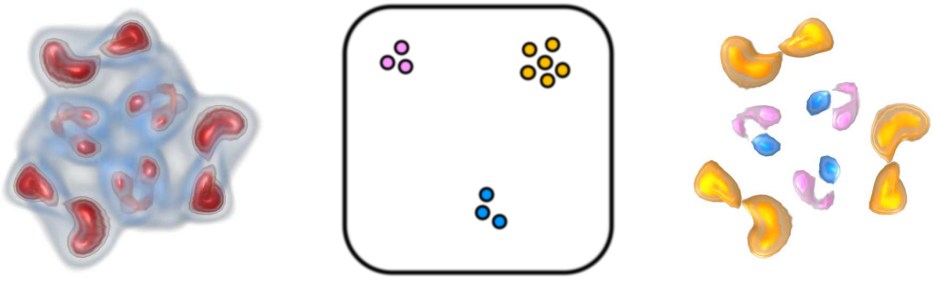


Figure 6.2: Clustering based analysis detects symmetry at different scales in a 3D cryo-electron microscopy image of AMP-activated kinase (EMDB-1897). (left) The three-fold rotational symmetry is apparent from the volume rendering. (center) Contours are represented as points in a high-dimensional shape descriptor space (illustrated in 2D). Symmetric contours form a cluster in the descriptor space and can be easily identified. Three such clusters are shown in gold, blue, and pink. (right) Three symmetric regions of different sizes, highlighted in gold, blue, and pink, detected by the method.

regions are reported as symmetric. Note that the choice of the shape descriptor is not fixed and depending on the noise characteristics and the definition of similarity relevant to the application of interest, an appropriate descriptor may be used. Our method uses topological information derived from the contour tree to infer importance of a feature and this allows the design of a feature-aware algorithm for symmetry identification. By selecting a contour from each arc of the contour tree, we are able to detect symmetry at different scales similar to the contour tree based approach. Noise in the data is seamlessly handled by the clustering framework through the choice of a robust shape descriptor. Thus, our approach combines the advantages of the earlier methods based on the contour tree and extremum graph. Figure 6.2 illustrates our approach on a 3D cryo-EM image of AMP-activated kinase (EMDB-1897) with three-fold rotational symmetry. Our method identifies symmetric regions of different scales. The large-scale features shown in gold and the small-scale features shown in blue and pink highlight the multiscale aspect of our approach.

The main contributions of this work are the following:

- A formulation of the problem of symmetry detection in scalar fields as a clustering problem in a shape descriptor space. This model provides a lot of flexibility in analyzing similarity of scalar fields as well as handling noise since it allows the

shape representation and the descriptor space to be varied.

- A novel representation of a scalar field as a collection of points where each point represents a contour in a contour shape descriptor space. Similarity between contours is naturally defined as the distance between points in this space. This is a generic representation of independent interest and we show its benefit in similarity analysis of scalar fields.

- A robust algorithm to detect symmetric regions at multiple scales. Though geometry based symmetry detection methods are typically computationally costly, we design an efficient algorithm that employs elegant optimizations by incorporating topological information about the contours using the contour tree.

6.1 Symmetry Detection via Contour Clustering

Methods based on clustering [56, 64, 65, 77, 104, 105] have shown superior performance in identifying symmetry in shapes. However, directly extending these methods to scalar fields is non-trivial. Consider a *scalar field*, $f : \mathbb{M} \to \mathbb{R}$, with subdomains $\mathbb{M}_1, \mathbb{M}_2 \subseteq \mathbb{M}$ Let c_1 and c_2 be contours of the same level set that belong to $\mathbb{M}_1$ and $\mathbb{M}_2$ respectively. It is easy to see that if $\mathbb{M}_1$ and $\mathbb{M}_2$ are symmetric then c_1 and c_2 are also symmetric. We make use of this property and detect symmetric subdomains by identifying symmetry of the contours belonging to the subdomains.

Let $\mathbb{C}$ be the set of all contours. Consider a function $g : \mathbb{C} \to \mathbb{R}^n$ such that $g(c) =$

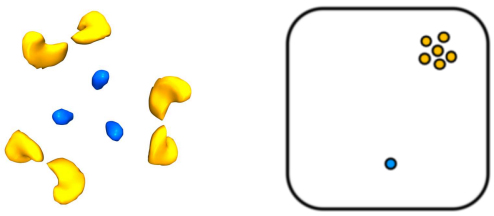


Figure 6.3: Mapping contours to points in a descriptor space. Symmetric contours shown in blue are mapped to the same blue point in the descriptor space. Six approximately symmetric contours shown in gold are mapped to six points that lie in close proximity to each other in the descriptor space.

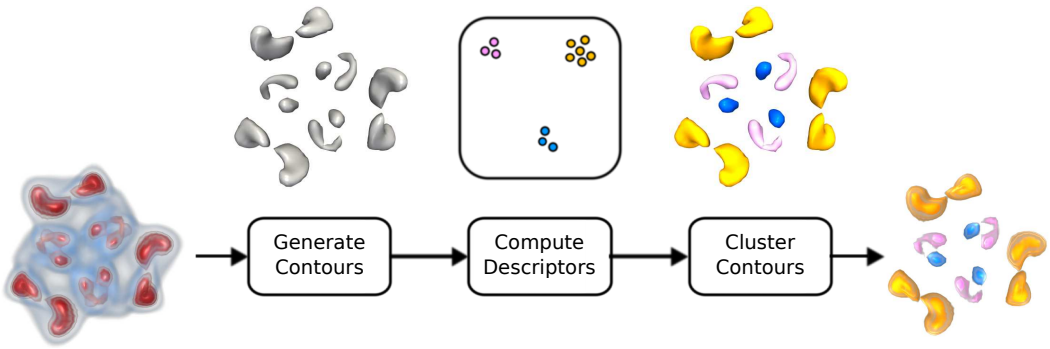


Figure 6.4: Symmetry detection pipeline. Contours are extracted from the scalar field and a descriptor is generated for each contour. A similarity score is estimated between pairs of contours based on the distance between the points in the descriptor space. Next, the set of symmetric contours are identified via clustering. Finally, the region of the domain to which each symmetric contour belongs is extracted and reported.

$g(T(c))$ where T is a transformation. In other words, g is a function that maps each contour to a point in a high-dimensional space such that a contour and its symmetric copies are mapped to the same point. The point to which a contour is mapped is called a *descriptor* and the high-dimensional space is called the *descriptor space*, see Figure 6.3. For illustration, the descriptor space is shown in 2D but the actual dimension of the space depends on the choice of the descriptor. The distance between contours in the descriptor space is a measure of their similarity. In practice, scalar fields do not exhibit perfect symmetry and therefore it is important to detect symmetry in an approximate sense. Ideally, deviation from perfect symmetry should be measured in the space of shapes but it is more convenient to measure deviations in the descriptor space. If contours c_1 and c_2 are not perfectly symmetric, then c_1 and c_2 will not be mapped to the same point in the descriptor space. The distance between the contours in the descriptor space, $\|g(c_1) - g(c_2)\|$, will be indicative of the deviation from perfect symmetry.

Definition (Symmetric Contours)**.** *Contours c_1 and c_2 are perfectly symmetric if $\|g(c_1) - g(c_2)\| = 0$, where $\|\cdot\|$ is a norm in the descriptor space. They are ϵ-symmetric if $\|g(c_1) - g(c_2)\| \leq \epsilon$, for $\epsilon > 0$.*

Shape descriptors have been extensively used in the geometry processing community for shape matching and there is a vast collection of research papers in this area [55, 76,

91, 99]. Note that different norms may have to be used for different descriptors. It is possible that a shape descriptor may incorrectly map contours c_1 and c_2 to the same point even when they have totally different shapes. A good shape descriptor should discriminate well between different shapes and minimize such incorrect mappings.

Figure 6.4 illustrates the main steps of our algorithm. Given a scalar field as input, we generate a set of contours. For each contour thus generated, a descriptor is computed. The descriptor for each contour can be considered to be a point in a high-dimensional descriptor space. The descriptor space is a transformation-invariant space, i.e., it reverses the effect of geometric transformation on contours. Thus, perfectly symmetric contours are mapped to the same point in the descriptor space. Imperfections in symmetry results in imperfections in the mapping. Since contours with similar shape have similar descriptors, the points in the descriptor space representing approximately symmetric contours will lie in close proximity to each other. Therefore, symmetric contours can be recognized by identifying clusters in the descriptor space. The volumetric regions represented by the contours within a cluster are then reported as symmetric regions.

6.1.1 Contour Generation

Our algorithm assumes that each region of interest in the domain is represented by a contour belonging to it. Hence, it is important to use a sampling strategy that generates a contour from each region of interest. The obvious method for generating contours is to sample isovalues uniformly from the range of the function values and extract contours corresponding to these isovalues. A coarse uniform sampling may not generate contours within a specific region and thus fail to recognize it as a symmetric region. On the other hand, a fine sampling may generate multiple contours within the same region and redundant computations. Ideally, each symmetric region should require only a single representative contour for its detection. The contour tree is a powerful tool that encapsulates information about the evolution of contours [19] and we leverage information obtained from the contour tree for optimal generation of contours. Each arc of the contour tree represents a family of contours that are nested one inside the other forming

offset surfaces similar to layers of onion peel. Hence, to capture the geometry of the region of the domain corresponding to an arc of the contour tree, we select only one contour from each arc of the contour tree.

Selecting a representative contour from each arc of the contour tree ensures that no regions are missed in the subsequent symmetry analysis. An arc in the contour tree may either represent a feature associated with a single extremum or a region formed by the merger of multiple features and hence associated with multiple extrema. As a result, our method can detect symmetry at multiple scales. However, for noisy scalar fields, a large number of arcs of the contour tree may correspond to noise. Selecting a contour from each arc of the contour tree will result in a significant amount of computational time spent in processing these noisy contours. To overcome this problem, we generate a contour from an arc of the contour tree only if the arc is deemed to represent a feature and not noise. The definition of noise is subjective and depends on the application. In this work, we consider an arc to be noise if the volume of the largest contour associated with the arc is below a user defined *noise threshold* δ. The volume of a contour is approximated as the number of vertices of the domain enclosed by the contour. This approximation assumes that the scalar field is continuously sampled. Since each arc of the contour tree can be associated with the set of vertices of the domain that comprise the subvolume corresponding to the arc, this estimation of the volume can be done efficiently [19]. In the absence of the metadata information provided by the contour tree, all contours would have had to be treated as equally important. In summary, an elegant optimization is achieved through contour tree driven sampling that both ensures that each region has a unique representative contour and avoids sampling of contours from noisy regions.

6.1.2 Contour Representation

Once a representative contour is generated from each region of interest, the next step is to generate its descriptor. The similarity score between a pair of contours is estimated using the distance between their descriptors. Designing shape descriptors for matching and retrieving similar shapes is a well studied area in the geometry processing community.

The notion of similarity is subjective and varies from application to application. A major advantage of our method is that it is not restricted, in principle, to a particular choice of the shape descriptor. Instead, it offers flexibility in choosing the descriptor appropriate for an application. Hence, our method may be viewed as a generic framework for identifying similar regions in a scalar field. For example, if an application is interested in identifying similarity only with respect to rotation, a rotation invariant descriptor may be used. The only prerequisite on the descriptor is that it should be discriminative, i.e., similar contours should be mapped to points that are nearby in the descriptor space while contours that differ from each other should be mapped to far away points. Therefore, it is important to use shape descriptors with high precision and recall ratios [55] for applications that cannot tolerate false positives during shape retrieval.

6.1.3 Contour Clustering

After the descriptor generation stage, any standard clustering method may be used to locate the clusters in the descriptor space that represent symmetric contours. However, requiring the explicit generation of descriptors as a constraint limits the flexibility of using our approach as a generic framework for similarity detection. We observe that mapping of contours to points in the descriptor space is not a prerequisite for clustering. A similarity correspondence graph can be constructed from a set of contours by representing each contour as a node in the graph and inserting an edge between two nodes if the respective contours are similar. It is easy to see that the contours that are similar form a clique under this representation [56] and these cliques can be identified to detect a set of similar contours. Thus, given a procedure that assigns a similarity score between pairs of contours, further processing is performed solely on the graph and is independent of the actual definition of similarity. This allows considerable freedom in choosing a similarity measure that is relevant to an application. In particular, for symmetry identification, the distance between points in the descriptor space is used to assign the score between pairs of contours.

Given a set of contours marked as symmetric, the arc in the contour tree to which

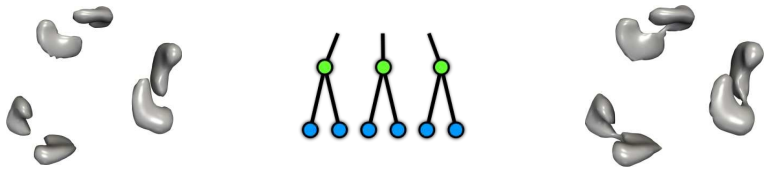


Figure 6.5: When contours merge, their shape change significantly. (left) Six contours before they merge. (center) Part of the contour tree depicting merging of pairs of contours. (right) Three contours after the merge.

each contour belongs to can be determined. The region of the domain enclosed by the largest contour of the arc can then be extracted and reported as a symmetric region. Although this works well in practice, note that the contour itself may have evolved due to the merging of contours nested within it. An application with stricter requirements on symmetry detection may need to also incorporate the symmetry of these nested contours in the algorithm. This presents a challenge in directly using clusters in the descriptor space for detecting symmetric regions because the descriptor space is not continuous with respect to the evolution of the shape of a contour during a level set sweep, see Figure 6.5. Recall that a cluster in the descriptor space represents a set of contours with the same shape. As illustrated in the left and right figures, the shape of individual nested contours before merge is different from the shape of the contour after the merge. Therefore the points corresponding to the contours before and after the merge may not be part of the same cluster.

To address this issue, we incorporate the similarity score between the children contours (contours before merging) into the calculation of the similarity score between a given pair of parent contours (contours after merging). Let p and q be two parent contours with children contours ${c_1}^p, \ldots, {c_n}^p$ and ${c_1}^q, \ldots, {c_m}^q$, respectively. Assume that the score between two children contours ${c_i}^p$ and ${c_j}^q$ is known. For contours with children, the procedure below can be applied bottom up to determine their score while for contours that do not have children, the score can be directly determined from the descriptor space. To calculate the similarity score between p and q, first the contribution from the children contours is determined. We construct a bipartite graph where nodes in the two partitions are ${c_1}^p, \ldots, {c_n}^p$ and ${c_1}^q, \ldots, {c_m}^q$, see Figure 6.6. An edge between ${c_i}^p$ and

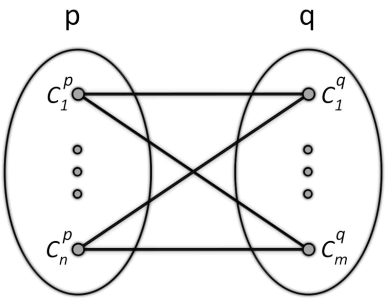


Figure 6.6: Maximum weight matching is used to determine the contribution of children contours towards the similarity score between the parent contours p and q. Each edge ${c_i}^p{c_j}^q$ is weighted with the similarity score between the child contours ${c_i}^p$ and ${c_j}^q$.

${c_j}^q$ is weighted with the score between ${c_i}^p$ and ${c_j}^q$. The maximum weight matching is computed to determine the similarity score between the children contours. The score between the contours p and q obtained directly from the descriptor space is added to the value of maximum weight matching to obtain the cumulative similarity score between p and q. An analogous procedure may be used for nested contours that split from a parent contour.

6.2 Implementation

We now elaborate on the implementation details of our symmetry identification algorithm. We describe the factors that determine the selection of isovalues, the particular shape descriptor that we use and its properties, and the clustering algorithm we employ.

6.2.1 Isovalue Selection

Each arc of the contour tree encodes the range of values of the scalar field restricted to the subdomain represented by the arc. The question that remains is which function value in this range should be used to generate the representative contour for the arc. To generate a large contour belonging to the subvolume represented by an arc, the isovalue selected should be close to the saddle value at which the contour merges into another contour. However, perturbation in function values due to noise may lead to instability

of the saddles in the contour tree. As a result, the function value at which the contours from symmetric regions merge and split may not be consistent. Figure 6.11(b) shows an example where, for the same isovalue, children contours of the green contours have already merged to form a single component while the blue contours are yet to merge. We require an isovalue that is both close to the saddle and generates isocontours that are consistent, with respect to merging. A simple strategy we adopt is to ensure that none of the contours merge by selecting an isovalue that is lower than the lowest join saddle at which the contours merge. We introduce a *stabilization parameter* α that determines the length of the interval of function values within which the saddle values are not consistent with respect to merging of contours. From the list of all saddles in the contour tree, we first remove all saddles belonging to arcs that are considered to be noise with respect to the noise threshold δ. Within the reduced list, we consider each join saddle s, where contours merge into one, and select an isovalue $w = f(s) - \alpha$ if there are no other saddles whose function value lie within the interval $[w, f(s)]$. Among the set of isovalues thus generated, if w is the highest isovalue that lies in between the function values at the end points of an arc, then w is chosen as the isovalue for the arc. An analogous procedure is used to select isovalues with respect to split saddles. A procedure for determining the value of α is described in Section 6.3.2.

6.2.2 Shape Descriptor

The choice of shape descriptor depends on the kind of similarity analysis required by an application. A function ϕ which satisfies the equation $\Delta\phi = -\lambda\phi$, where Δ is the Laplace-Beltrami operator, is called an eigen function of Δ and λ is called an eigen value [80]. We use the first ten non-zero eigen values of the Laplace-Beltrami spectra as the shape descriptor since noise in the shape has limited influence on the initial eigen values [55]. This descriptor has been used for shape matching and retrieval and is robust in the presence of noise, deformations in shape, and differences in the underlying triangulation [67, 78, 79]. Figure 6.7 illustrates the robustness of the Laplace-Beltrami shape descriptor when increasing levels of noise are added to a synthetic data set. Although it

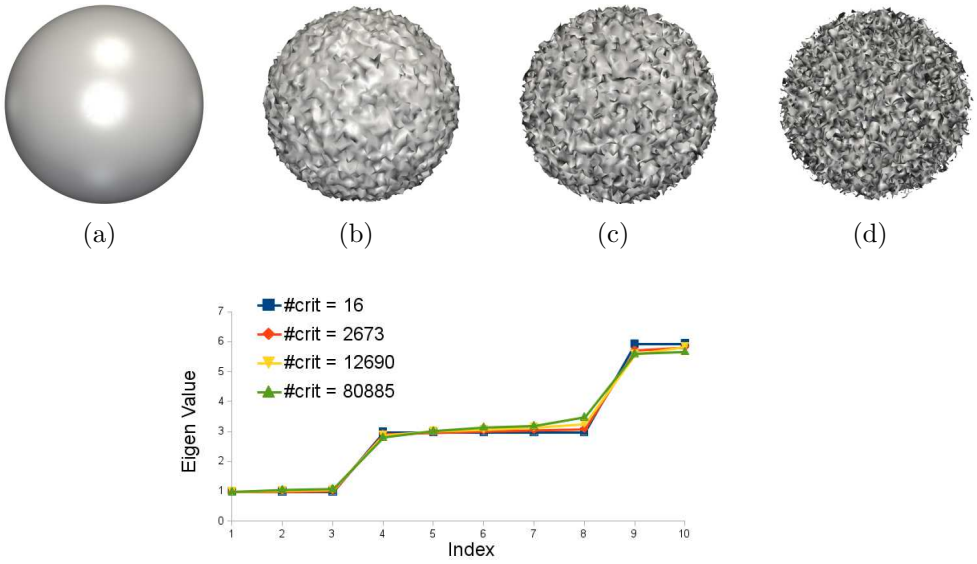


Figure 6.7: Robustness to noise. (a) An isosurface of a synthetic data set which has the shape of a sphere. The number of critical points for this data set is 16. (b)-(d) Adding increasing levels of noise to the data set deforms the isosurface. The number of critical points increase to 2673, 12690, and 80885. (e) For each data sets, the first ten non-zero eigen values of the Laplace-Beltrami spectra are normalized by dividing with the first non-zero eigen value and plotted as a 1D curve. The similarity of the curves shows that the similarity of the contours can be identified easily even in the presence of noise.

is possible that two different shapes may have the same spectra, it is very rare in practice [78]. The descriptor is discriminative with high precision and recall ratios [55]. We use the popular cotangent weighted scheme for computing the Laplace-Beltrami spectra [73]. Computation of the Laplace-Beltrami spectra on large meshes is costly and therefore we simplify the contour meshes so that the number of vertices is small. We simplify the mesh down to 1000 vertices in our experiments through edge collapses driven by the quadric error metric [44]. If the isosurfaces contain skinny triangles that result in numerical errors in the computation of the spectra, mesh quality aware isosurface generation or remeshing [85, 86] may have to be performed.

6.2.3 Clustering

We observe that clusters in the descriptor space are well separated and therefore employ a simple scheme based on nearest neighbor search for clustering. For a given contour c, other contours that are ϵ-symmetric with respect to c are determined by locating points in the descriptor space that lie within a sphere of radius ϵ centered at p_c, where p_c is the point in the descriptor space to which c is mapped. We limit the number of points in the search space by considering only those points that represent the contours that belong to the level set of c. The ideal metric for computing distances in the descriptor space may be determined using methods like metric learning [51]. However, we use Euclidean distances since it has yielded good results for shape retrieval with the Laplace-Beltrami spectra [55, 78]. The value of the approximation parameter ϵ is specified by the user.

6.3 Results and Discussion

In this section, we report the results of applying our method on different data sets and elaborate on comparison with the earlier topology based methods that we discussed, selection of parameters, and computational performance.

6.3.1 Comparison with Topological Methods

The segmentation of the domain utilized by our algorithm for locating symmetric regions is induced by topological features identified through the contour tree. We compare our method with existing symmetry detection methods that also segment the domain on the basis of topological features. While the contour tree based method relies on structural similarity between the subtrees for symmetry identification, the extremum graph based method depends on distances between the extrema evaluated on an augmented version of the extremum graph.

As we pointed out, noise in the data that destroy the repeating structure of the subtrees of the contour tree poses challenges in determining symmetry using the contour tree. The extremum graph based method is better at handling noise and report

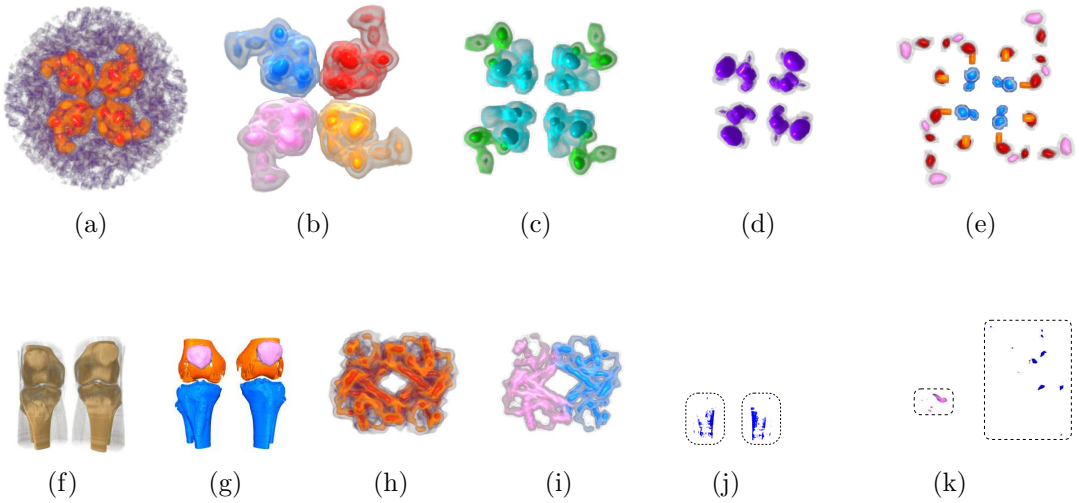


Figure 6.8: Comparison with topological methods for symmetry detection. (a) Volume rendering of a cryo-EM data set (EMDB-1654), with noisy regions depicted in violet. (b)-(e) Symmetric regions at multiple scales detected by our method. The extremum graph based method reports symmetry only at the largest scale. (f) Volume rendering of a CT scan of a pair of knees. (g) Bilateral symmetry of the bones detected by our method, shown in orange, blue, and pink, even in the presence of missing regions and noise in the contours of the data set. (h) Volume rendering of the electron density of a hemoglobin molecule (PDB-ID 1HGA) that exhibit 2-fold rotational symmetry. (i) The two symmetric regions are detected by our method even though they lie in close proximity to each other. (j) Flat regions corresponding to the global maximum of the knee data set and (k) asymmetric distribution of the extrema of the molecule data set, shown within dotted boxes, pose challenges in determining the seed set for the extremum graph based method.

symmetry detected at the largest scale for several cryo-EM data sets. The contour tree based method, on the other hand, can detect symmetry at different scales. Our method combines the advantage of these two methods. The first row in Figure 6.8 shows the result of our method on one of the noisy data sets, EMDB-1654, which was used with the extremum graph based method. In addition to detecting symmetry at the largest scale, our method is able to identify smaller symmetric regions at different scales. Figure 6.9 shows the projection of the descriptors onto 2D using multidimensional scaling and illustrates that the clusters representing symmetric regions are well separated and can be easily identified. In contrast, the extremum graph based method requires user

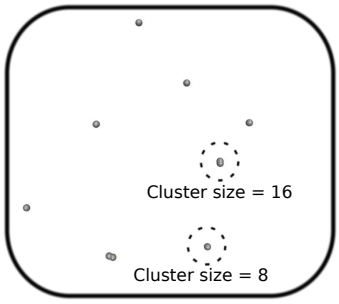


Figure 6.9: Projection of 93 points in the descriptor space to 2D using multidimensional scaling for the data set EMDB-1654. Observe that the clusters are well separated. For the two clusters within the dotted circles, their sizes are annotated below the cluster.

guidance for carefully selecting a set of well separated and symmetrically distributed extrema, called the seed set, for symmetry detection. Typical cryo-EM data sets is devoid of noise in the high and low density regions and therefore the symmetry of the extrema belonging to these regions can be easily identified for choosing a seed set.

Our method does not make such assumptions and we show the results of our method on two data sets where selection of seed set is non-trivial. Figure 6.8(f) shows a volume rendering of a CT scan of a pair of knees. Observe the dark regions in the bone where the scalar values are noisy. Our method identifies the symmetry between the contours belonging to three portions of the bones in the left and the right knee, shown in orange, blue, and pink in Figure 6.8(g). An inspection of these contours shows that the dark regions of the bone in the volume rendering denote missing regions and noise in the contours. Despite these missing regions and noise, our method successfully identifies the symmetry of the bones. The isosurface for this data set at the global maximum in Figure 6.8(j) shows that the global maximum is degenerate and forms a flat region. Figure 6.8(h) shows an electron density field derived from the atomic coordinates of a hemoglobin molecule using EMAN software [58]. Our method identifies two symmetric regions, shown in blue and pink in Figure 6.8(i), despite these two regions being in close proximity to each other. The isosurface for an isovalue close to the global maximum, see Figure 6.8(k), shows that the extrema belonging to the high density regions are not symmetrically distributed within the two regions. In both these cases, selection of a set of extrema as seed set is not an easy task because of the flat regions and the asymmetric

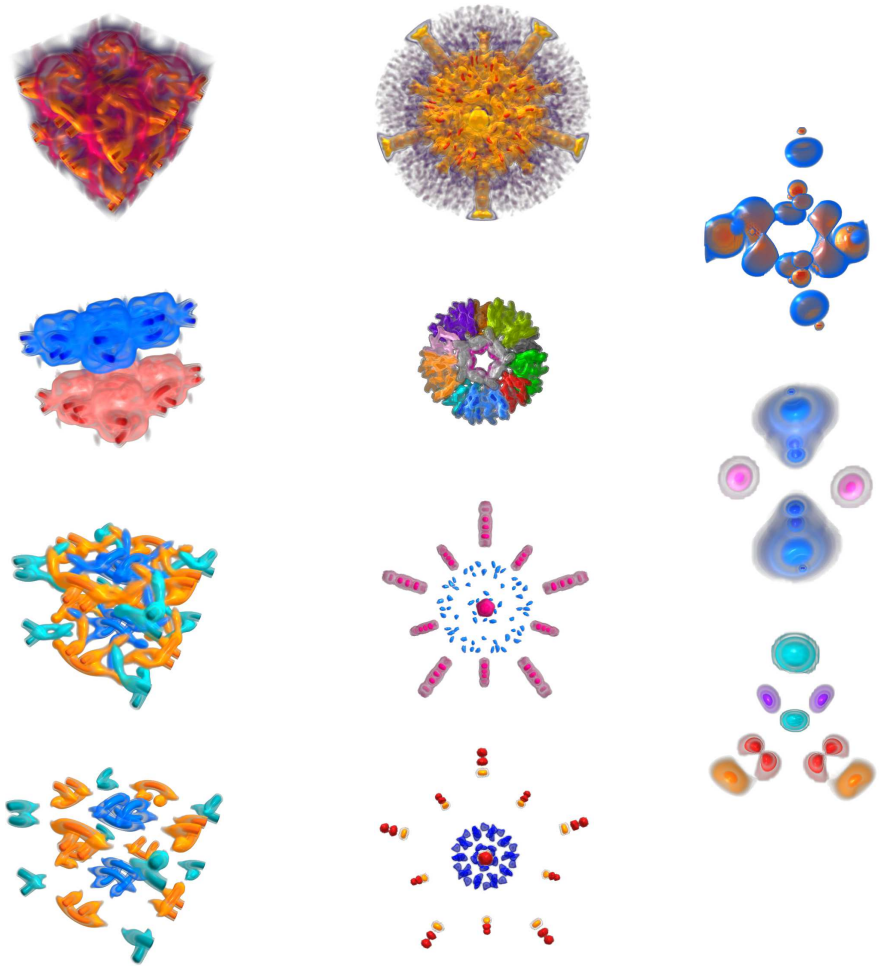


Figure 6.10: Multiscale symmetry detected on (left column) Vortex (middle column) EMDB-1179 and (right column) Neghip data sets. The topmost figure in each column shows a volume rendering of the data set and different symmetric regions detected are shown below it.

distribution of the extrema. Results on additional data sets are shown in Figure 6.10.

6.3.2 Parameter Selection

The approximation parameter ϵ and the noise parameter δ are external to our algorithm. Hence, the stability parameter α is the only parameter that is specific to our method.

Stabilization parameter α. The value of the stabilization parameter affects the detection of features that are very transient with respect to the evolution of contours during a level set sweep. Choosing too small a value for the stabilization parameter may result in

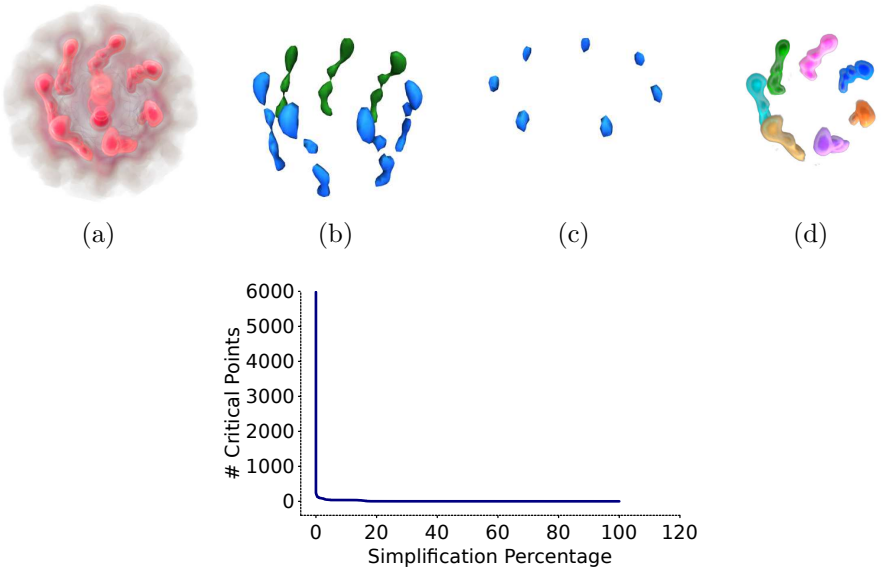


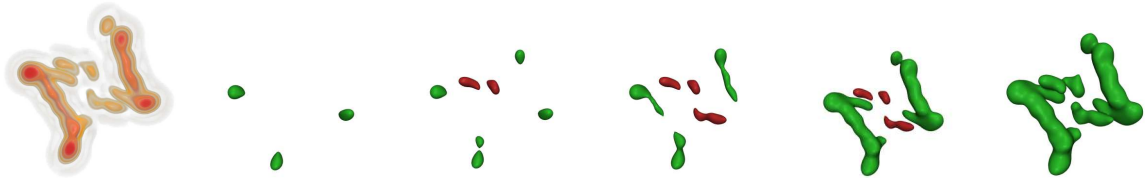
Figure 6.11: Influence of stabilization parameter, α, on symmetry detection. (a) Volume rendering of a cryo-EM data set (EMDB-1292) shows long tube-like structures that exhibit 7-fold rotational symmetry. (b) Setting α to 0.2% of the length of the range of scalar values generates unstable contours within the tube-like regions. The inconsistency in the merging of the blue and green contours indicates this instability. (c) Setting α to 1% discards the unstable contours. Instead, the small blue contours are generated at a more stable isovalue. (d) The 7-fold symmetry of the tube-like regions are identified in both cases since these larger features are not sensitive to the exact value of α. (e) The plot of the number of critical points with increasing simplification shows a significant drop at a low value of α. We set α to a value immediately after this drop.

only partial detection of such unstable symmetric regions while choosing too high a value may result in ignoring these unstable regions in the symmetry analysis. We illustrate this with a cryo-EM data set (EMDB-1292) with 7-fold rotational symmetry, see the volume rendering in Figure 6.11(a). In an attempt to capture the symmetry of the small unstable regions within the long tube-like regions that form the 7-fold symmetry, we set α to 0.2% of the total range of scalar values. However, only four of the seven symmetric regions generate contours belonging to the small regions within the tube-like regions, shown in blue in Figure 6.11(b). Due to the instability of the saddles, the contours belonging to the remaining three regions, for the same isovalue, have already merged, as shown by the green contours.

To determine a suitable value for α, we adopt the same heuristic used earlier to stabilize the branch decomposition representation of the contour tree. We plot the drop in the number of critical points with increasing simplification of the contour tree, see Figure 6.11(e). Initially, there is a sudden drop due to the unstable critical points, after which the curve tapers off. Based on this plot, we select a value immediately after the initial drop. Setting α to 1% discards the unstable isovalue generated earlier. Figure 6.11(c) depicts the 7-fold symmetry of the small blue contours from the top portion of the tube-like regions identified using the new isovalue generated. The symmetry of the remaining small regions are not detected since the higher value of α discards the unstable contours in this region. Note that the exact value of α is crucial only while determining symmetry of the contours belonging to very small arcs of the contour tree. These contours are very transient and quickly merge or split into another contour during a level set sweep. For both values of α, the symmetry of the tube-like regions is detected correctly at a different isovalue that is more stable, see Figure 6.11(d). This is because these contours belong to longer arcs and are insensitive to the exact value of α.

The inability to detect symmetric regions due to unstable contours is a limitation of our implementation. This limitation arises from the constraint that for the same isovalue, contours should be consistent with respect to merging. A solution is to relax this constraint and allow a small interval of isovalues from which stable contours may be generated independently by employing a geometric criterion like maximally stable contour [61]. However, when contours at different isovalues are used to detect symmetry, the variation in the isovalues should also be considered in determining the quality of the symmetry. For this purpose, we are currently exploring ways to include the differences in the isovalues into the distance measure between contours in the descriptor space.

Approximation parameter ϵ. Changing the value of ϵ directly affects the output of our method since ϵ controls the quality of the symmetry detected. We believe that it is best to let the user choose the value of ϵ for a data set, on the basis of the output it generates, instead of determining a specific value through a heuristic procedure. Note that the first and the second stages of our algorithm, namely, generation of contours and

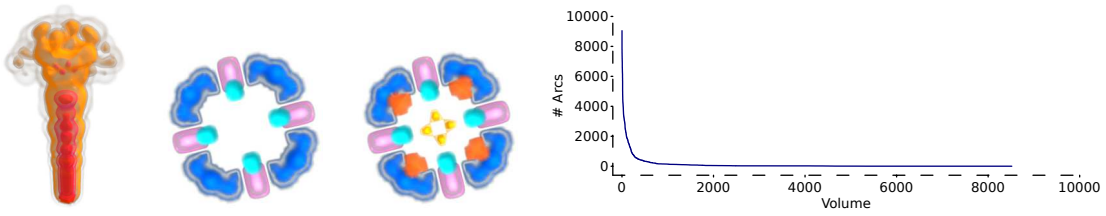


(a) (b) (c) (d) (e) (f)

Figure 6.12: Influence of approximation parameter, ϵ, on symmetry detection. (a) Volume rendering of a cryo-EM data set (EMDB-5214) with 2-fold symmetry where the left and the right regions show deviations from perfect symmetry. (b) The green oval contours that are almost perfectly symmetric are identified when $\epsilon = 0.1\%$. (c) Increasing ϵ to 1% results in the addition of two more smaller contours of oval shape and a new set of symmetric contours shown in maroon. (d) At $\epsilon = 4\%$ a new maroon contour, which is only partially symmetric with respect to the existing maroon contours, is located. Similarly, the green contours also deviate from perfect symmetry. (d) Larger green regions are detected at $\epsilon = 8\%$ and (e) ϵ =10%. The occluded contours are not shown.

computation of descriptors, are independent of the value of ϵ. Nearest neighbor search can be performed efficiently. Therefore, if the first and second stages of our method are precomputed, it is possible to efficiently regenerate the output as ϵ is varied. This makes it possible to provide the user with a slider interface to visualize the effect of varying ϵ and choose the ideal value for a data set.

Figure 6.12 illustrates the effect of varying ϵ on a cryo-EM data set (EMDB-5214). Even though the global symmetry is evident, carefully observe the variation between the left and the right regions. These variations affect the quality of symmetry and is captured by different values of ϵ in our experiment. We determine the maximum distance between a pair of points in the descriptor space and specify different values of ϵ as a percentage of the maximum distance. For $\epsilon = 0.1\%$, the oval contours that are highly symmetric, shown in Figure 6.12(b), are detected. As the value of ϵ is increased, more approximately symmetric contours are detected as shown in Figure 6.12(c)-(f). In each figure, contours belonging to the same cluster are shown with the same color. Note that the same data set was used earlier to emphasize the advantage of the extremum graph based method over the contour tree based method. While the extremum graph based method is able to detect the global symmetry by virtue of being geometry-aware, the contour tree based method fails to do so due to variations in the features of the left and the right regions.



(a) (b) (c) (d)

Figure 6.13: Influence of noise parameter, δ, on symmetry detection. (a) Volume rendering of the Fuel data set, which exhibits 4-fold rotational symmetry at the top of the long tube-like region. (b) Three symmetric regions reported when δ is set to 200 vertices. (c) Lowering δ to 20 vertices results in the detection of two additional symmetric regions that are smaller. (d) For the Fuel data set, the plot of the number of arcs in the contour tree with a specific value of volume as the value of the volume increases. The curve shows a significant drop initially due to the arcs corresponding to noise. A value immediately after this drop is chosen as the value of δ. Plots corresponding to the other data sets behave similarly.

Our method is able to quantify the effect of these variations on the output quality. Moreover, it can recognize partial symmetry at different scales in addition to identifying the global symmetry.

Noise parameter δ. We determine the number of arcs of the contour tree with volume above a particular value and plot the drop in the number of arcs with increasing value of volume. For the Fuel data set in Figure 6.13(a), this plot is shown in Figure 6.13(d). The plot shows a sudden drop in the number of arcs initially, similar to the graph used for determining the stabilization parameter α. This drop is due to a large number of small sized features in the data which are considered to be noise. We choose a value immediately after this sudden drop. When δ is set to 200 vertices for the Fuel data set, three sets of symmetric features, each exhibiting 4-fold rotational symmetry are identified, see Figure 6.13(b). Lowering δ to 20 vertices identifies two more smaller symmetric regions as illustrated in Figure 6.13(c). Thus, δ acts as a parameter that controls the size of features analyzed for multiscale symmetry detection. Setting δ to an even lower value generates contours of very small size whose descriptors cannot be reliably compared due to numerical errors. In comparison, the contour tree based method is able to easily capture the symmetry of smaller symmetric regions for this data set.

Table 6.1: Running time, measured in seconds, for the different steps in the symmetry detection algorithm. All experiments were performed on a 2 GHz Intel Xeon processor with 8GB RAM. The time taken for computing contour tree is denoted by t_{ct}, generating contours by t_{cg}, and computing contour descriptors by t_{cd}. The number of contours processed is denoted by #ctrs and the number of critical points by #crit.

Data set	#vertices	$t_{ct}(s)$	$t_{cg}(s)$	$t_{cd}(s)$	#ctrs	#crit
1HGA	$72 \times 72 \times 72$	1.3	7.7	28.7	78	4698
EMDB-1654	$112 \times 112 \times 112$	4.6	1.0	8.7	92	55762
Vortex	$253 \times 253 \times 253$	82.0	19.9	163.7	119	1184142
EMDB-1179	$255 \times 255 \times 255$	67.4	10.6	84.9	339	338688
Knee	$379 \times 229 \times 305$	120.4	94.6	429.1	442	1222491

They use a different model for determining noise based on persistence, which is the difference in function value between the extremum and the saddle of a topological feature. Although our implementation may be modified to incorporate persistence based noise determination, it is not possible to use a single model that works across data sets with different noise characteristics. Therefore, we suggest that the model for determining noise be adapted based on the application under consideration.

6.3.3 Performance

Table 6.1 lists the running time of our algorithm on different data sets. The time required to compute the contour tree depends on the size of the data set and the number of critical points present in it. The contour generation stage includes the time taken for the simplification of large meshes. If the variations in the size of the contour is discounted, the contour generation and the descriptor computation stages are linear in the number of contours processed. Figure 6.14 shows the scaling of computational performance with respect to the number of the contours. In our implementation, symmetry with respect to contours that evolve from maxima and minima are identified in two separate passes and the running time listed above is for a single pass. The clustering stage is very quick and requires less than a second. The current implementation is not very efficient and offers a lot of scope for improvement. However, we believe that even with our suboptimal implementation, the running times that we report are reasonable for a geometry based

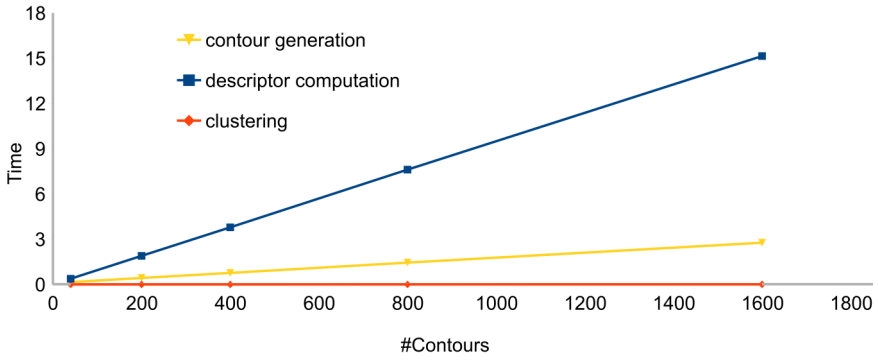


Figure 6.14: Computational performance scales linearly with respect to the number of contours. The time taken, in seconds, for contour generation, descriptor computation, and clustering are plotted when the number of contours processed are 40, 200, 400, 800, and 1600. The plot shows linear scale up in the computation time for contour generation and descriptor computation. The nearest neighbor search takes only a fraction of a second. The contours are generated from the data set EMDB-1654. To limit the variation in performance due to differences in the size of the contours, this experiment was performed with contours belonging to a single scale.

algorithm that identifies symmetry at multiple scales.

6.4 Conclusions

In this chapter, we present a novel symmetry detection method based on the idea of clustering contours. We show that mapping contours to points in a descriptor space is a powerful representation for performing similarity analysis on scalar fields. One of the main limitations of our method is that symmetry identification is restricted to regions obtained from the segmentation of the domain induced by the contour tree. We plan to extend our method to handle other kinds of segmentation by designing an appropriate surface representation for the segments. A related issue is that our method does not consider symmetry within a contour. A pre-processing step that segments contours into smaller partial surfaces may be designed to solve this problem. The current implementation does not incorporate geometric stability criteria for selecting representative contours. It is beneficial to employ such a criteria that selects a stable contour from each arc of the contour tree to ensure consistency in the shape of the contours extracted. We

believe that the descriptor space representation of scalar fields will spur research in determining structurally similar features in collections of related data sets like time-varying and ensemble data and plan to investigate this in future.

Chapter 7

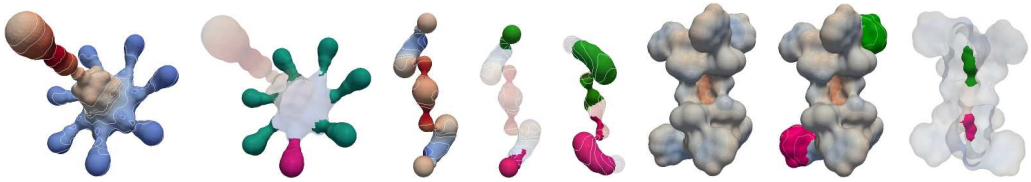
Applications

Symmetry in scalar field data provides important insights about the properties of the underlying scientific phenomena. Therefore, visualizing symmetric or repeating patterns in the data often help the domain scientists in understanding and characterizing the features in the data. Additionally, symmetry identification helps in improving traditional techniques for visualization of scalar field data. For example, symmetry information may be used in selection of meaningful cross-section planes for slicing volume data, better visualization of features by applying different rendering techniques that show complementary information in the symmetric regions, and selection of view directions that removes redundancies in the data [43]. In this chapter, we describe novel applications that use symmetry information to enhance visualization of scalar field data and to facilitate their exploration.

7.1 Query Driven Exploration

As the size and complexity of data generated through imaging and simulations grows, newer paradigms for effective exploration and interaction with the data are required. Query driven exploration is one such paradigm that facilitates navigation through complex data and is gaining popularity over the last few years [8, 57]. Symmetric regions in a data set facilitate query driven exploration of the data. For example, a user can select

a subdomain of interest as the query object and query for all subdomains that are similar to it. All the selected subdomains could then be optionally shown or hidden. Such tools are popular in image manipulation software and extending them to scalar field visualization will help users effectively explore their data. Based on the query object selected, we describe two related applications, namely, query segment driven exploration and query contour driven exploration.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 7.1: Query based exploration on electrostatic potential defined on molecular surfaces and channels. The query regions are shown in pink while the results are shown in green. (a) Electrostatic potential on the surface of the channel network in a membrane protein. (b) Query region and the result of the query. Six channels are symmetric to the query. (c) Electrostatic potential on the channel network in Gramicidin molecule. (d) The query region and the result. (e) A similar result for another query. (f) Electrostatic potential on the molecular surface of Gramicidin. (g) The query region along with the result. (h) A scenario where the query and the result are completely occluded by the molecular surface. To visualize the occluded regions, the context is shown in a translucent wire frame.

7.1.1 Query Segment Driven Exploration

A region of interest that is segmented from the domain may be employed as the query region and the symmetry information may be utilized to identify other regions similar to the query. To facilitate exploration using regions of the domain as query, the domain needs to be segmented. For the illustrations in this section, we use Morse decomposition to segment the scalar field [21, 25, 38]. In Figure 7.1, query-based exploration of the electrostatic potential fields defined on molecular surfaces and channels is shown. Figure 7.1(a) shows the channel network of a protein (PDB-ID: 2OAU) where the small bulb-like structures exhibit 7-fold rotational symmetry. One of the bulbs, shown in pink,

is selected as the query. The remaining six channels similar to the query are successfully detected and shown in green in Figure 7.1(b). Similar results are shown in Figure 7.1(c)-(e) for a channel through Gramicidin (PDB-ID: 1GRM). We also show results on the molecular surface of Gramicidin in Figure 7.1(f)-(h) for two different queries. The first query, shown in Figure 7.1(g) detects only one region symmetric to it with respect to the scalar field, even though that residue repeats eight times in the molecule, because the scalar field distribution of the rest of the residues are different. The second query, shown in Figure 7.1(h), highlights that the query can extract even regions which are occluded.

7.1.2 Query Contour Driven Exploration

To facilitate exploration using regions of the domain as query, the domain needs to be segmented and this is often a challenge. Instead, it is easy to visualize the evolution of contours during a level set sweep and select a contour of interest. We demonstrate that our symmetry detection method can be used to explore data sets using a contour as the query object.

A simulation of the hurricane Isabel[1] [52], which struck the west Atlantic in 2003, was performed on a $500\times500\times100$ grid over 48 time steps. Figure 7.2(a) shows a volume rendering of the pressure field in the first time step. The low pressure region is shown in red. A domain expert with the knowledge that isocontours in the low pressure region roughly has the shape of a hemisphere can use a synthetically generated shape, shown in Figure 7.2(b), as the query object. We generate different contours of the volume and identify the contour in the volume that is similar to the query object. This can be done efficiently using the contour clustering method by searching in descriptor space for the nearest neighbor of the query contour. We identify the contour shown in Figure 7.2(c) through this search. Note that the size and the exact shape of the query object may be different from that of the contour. However, our method is robust to small variations and therefore the query successfully identifies a contour from the low pressure region.

[1]Hurricane Isabel data was produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR and the U.S. National Science Foundation (NSF).

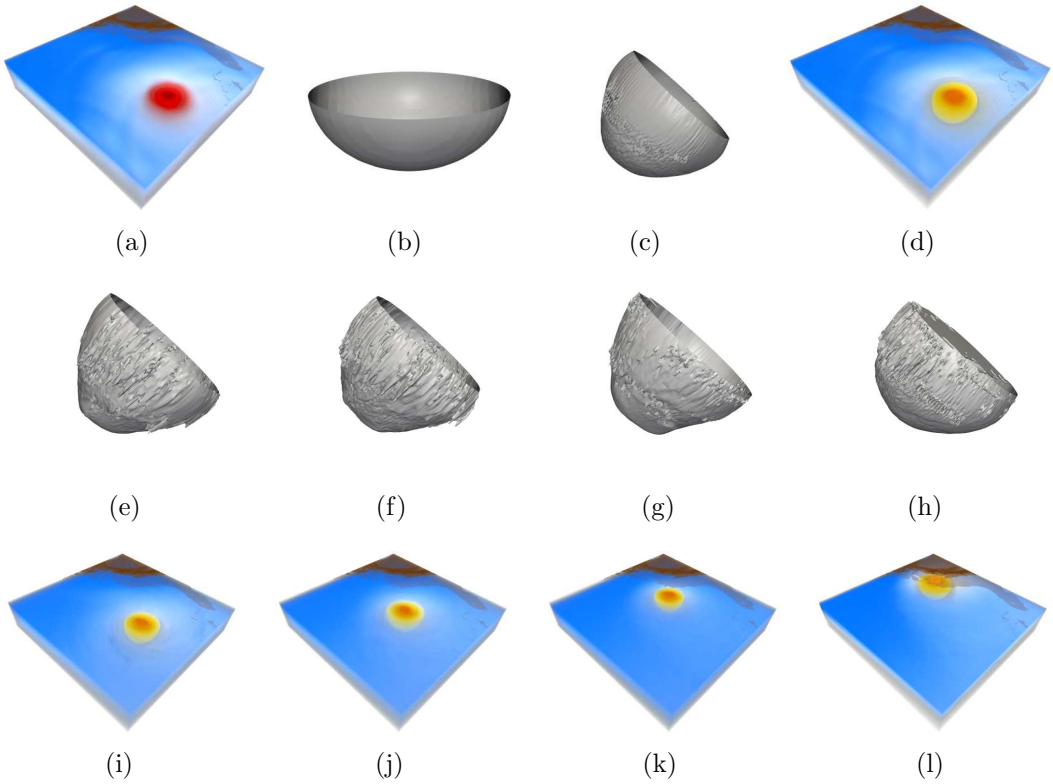


Figure 7.2: Query driven exploration using contours. (a) Volume rendering of the first time step of the hurricane Isabel data set. Low pressure region shown in red. Querying with (b) a synthetically generated half-ellipsoid finds (c) an isocontour with similar shape from the low pressure region. (d) The corresponding volumetric region shown in yellow. (e)-(h) The isocontour from the low pressure region of the first time step is employed as the query contour and the result on four different time steps, 12, 24, 36, and 48 is shown. Note that even though the contours contain noise, they are located correctly due to the robustness of our method. (i)-(l) The corresponding volumetric regions shown in yellow.

The volumetric region of the domain from which the contour is generated is extracted and displayed in the context of the rest of the volume in Figure 7.2(d).

We describe the above experiment to demonstrate the power of the descriptor space as a geometry based representation of the scalar field. But, we concede that using a synthetic shape as query object may not always be possible. However, the user could select one of the contours from a region of interest as the query object. It is straightforward to see that this approach may be used to query for symmetric regions within the same data set. What is more interesting is that the same approach can be extended to search through multiple data sets. We use the contour shown in Figure 7.2(c) from the

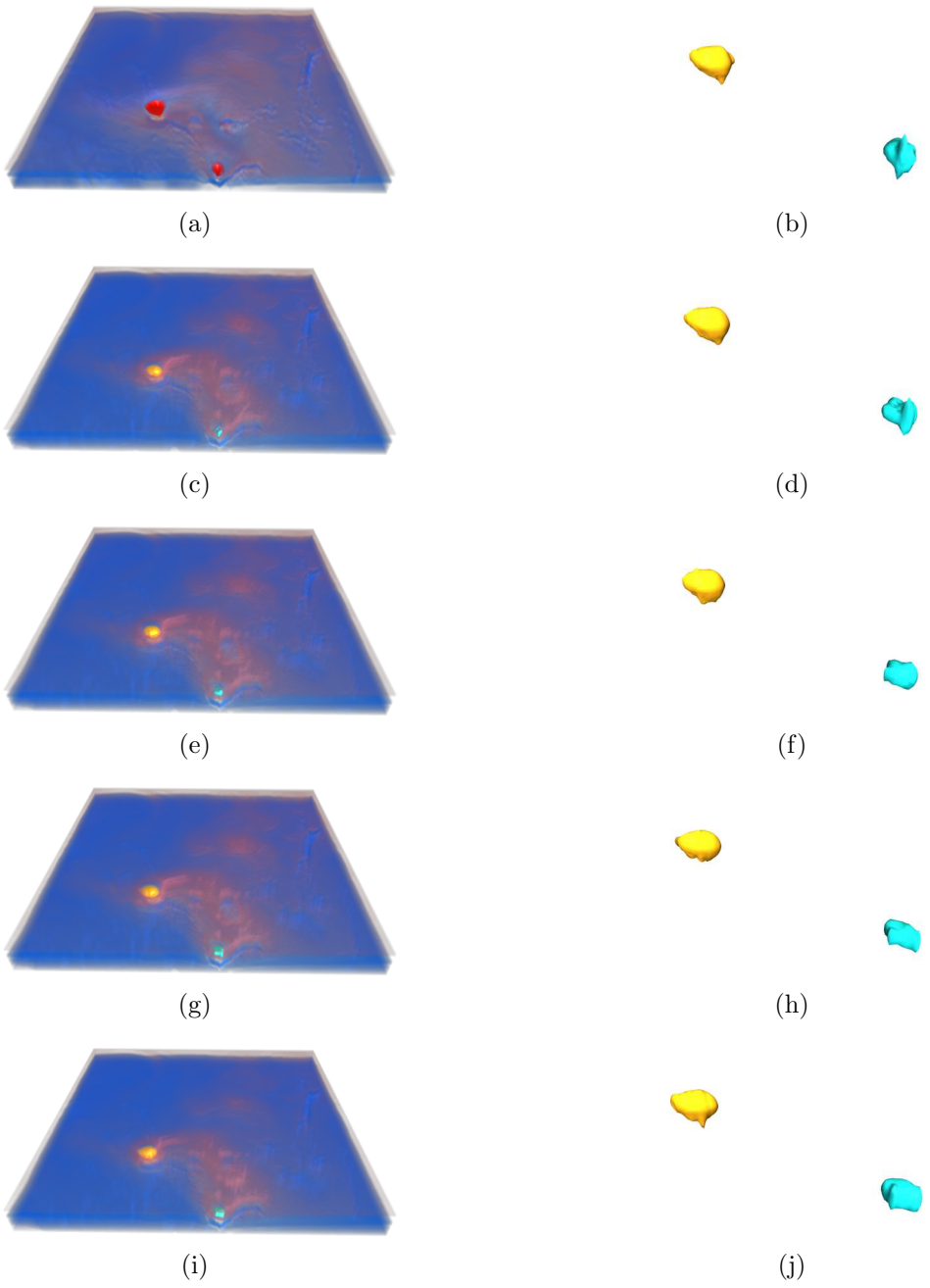


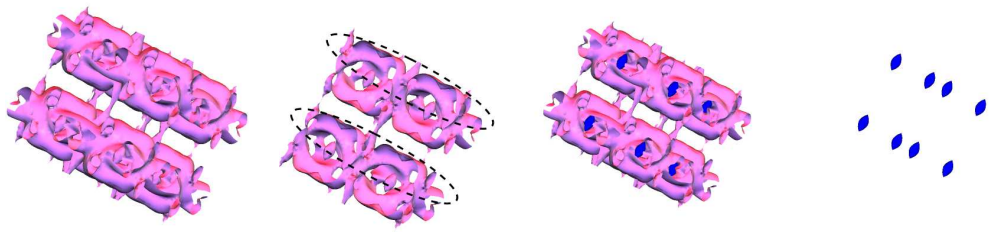
Figure 7.3: Query contour driven exploration using multiple contours on a weather simulation data set of size $306 \times 285 \times 27$. (a) Volume rendering of the pressure field from the first time step of the simulation shows two depressions in the Pacific ocean in red. (b) Two query contours selected from the low pressure region of the first time step shown in gold and cyan. (c)-(j) The result of the query on four subsequent time steps that are one hour apart shows the two low pressure regions detected in gold and cyan. The left and the right columns show the volumetric region extracted and the contour located, respectively.

first time step as the query object to search through the other time steps. The contours detected as a result of executing the query on four different time steps, 12, 24, 36, and 48, are shown in Figure 7.2(e)-(h). Observe that the contours show variations in their shape and have considerable amount of noise. Also, as opposed to the query contour, a significant portion of the mouth of the contour is closed in Figure 7.2(h) as the hurricane makes landfall. The query is successful even with these challenges and emphasizes the robustness of our method. Similar to Figure 7.2(d), the volumetric region of the domain can be extracted in each case as shown in the last row. We also show results on another weather simulation data set using two contours in Figure 7.3.

7.2 Symmetry-aware Isosurface Extraction

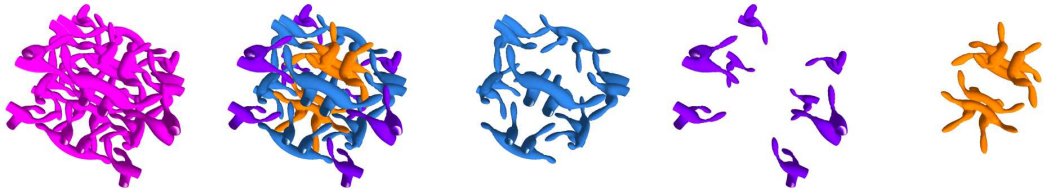
Volumetric data is often analyzed by extracting different isosurfaces and studying their properties. The number of components in the isosurface for a given isovalue can be quite large. Data exploration via isosurfaces often becomes difficult because the larger components may occlude the smaller components. 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 Figure 7.4(a)-(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. Traditional isosurface visualization does not distinguish between the different isosurface components. Symmetry information allows classification of isosurface components into different groups based on the symmetry of the components. Highlighting the isosurface components based on the group to which they belong results in enhanced visualization of the isosurface. Each group can be inspected independently and this avoids problems of occlusion and visual clutter while visualizing them, see Figure 7.5.



(a) (b) (c) (d)

Figure 7.4: 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) Symmetry information computed can be used to highlight the oval-shaped components in blue, or (d) hide rest of the components that occlude the oval-shaped components.



(a) (b) (c) (d) (e)

Figure 7.5: 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 based on symmetry. 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.

7.3 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 the scalar value at each point of the volume. Such transfer functions will not be able to highlight distinctly 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 [102, 112]. However, these methods do not detect repeating patterns in the domain while assigning different

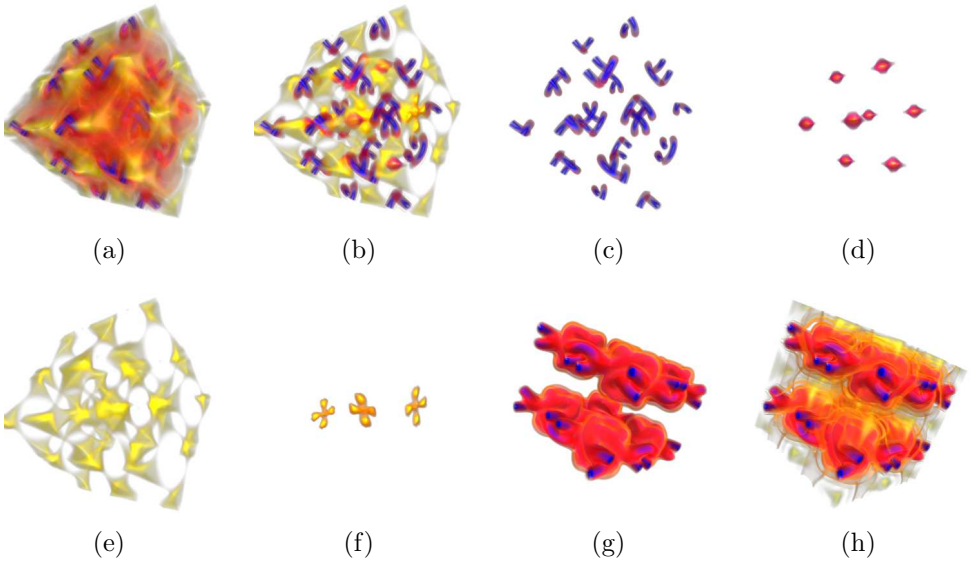


Figure 7.6: Symmetry-aware transfer function design. (a) Volume rendering of a 3D Taylor-Green vortex flow simulation data set. Traditional volume rendering makes it difficult to distinctly highlight different features belonging to the same function range. (b) Symmetric or repeating patterns are detected 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.

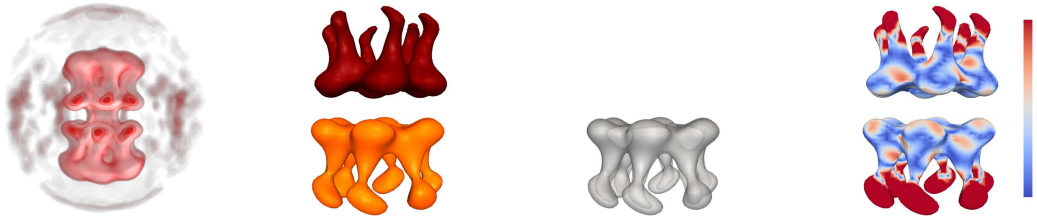
transfer functions. On the other hand, symmetry information we compute can be used to classify regions into different groups based on symmetry. Distinct symmetries can be assigned different transfer functions while repeating regions within the same symmetry can be assigned the same transfer function, thus highlighting repeating patterns in the domain.

Figure 7.6(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 7.6(b). These repeating regions can also be examined in isolation, see Figure 7.6(c)-(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 a group and a low opacity transfer function to the rest of the volume, see Figure 7.6(g)-(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.

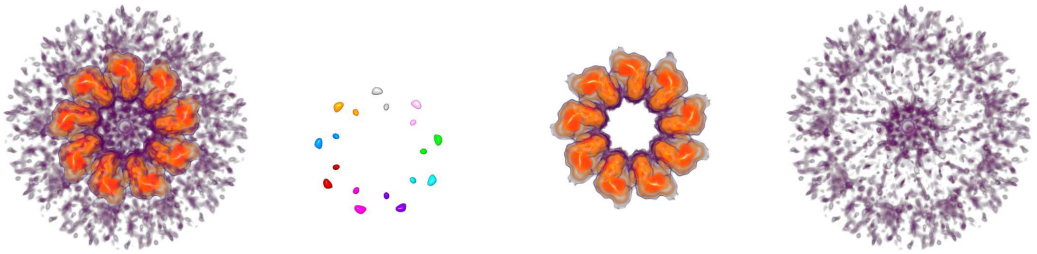
7.4 Asymmetry Visualization

Asymmetry in scalar fields often reveals crucial characteristics of the underlying physical phenomenon. For example, asymmetry between the left and the right breasts in mammogram images is an indicator of breast cancer [90]. Given a set of contours $c_1, \ldots, c_n$, reported as symmetric, we determine asymmetry of contour c_k, $1 \leq k \leq n$, with respect to the set of remaining contours $S = \{c_1, \ldots, c_{k-1}, c_{k+1}, \ldots, c_n\}$. For this, we first determine the symmetry transformation that maps each $c_i \in S$ onto c_k and spatially align c_i with c_k by applying the transformation. The transformation may be determined using the ICP (Iterative Closest Point) algorithm. These spatially aligned meshes in S are then glued together to generate a single mesh, say c_{avg}, with respect to which asymmetry of c_k is determined. For this purpose, we use the `Metro` tool [23], which evaluates the distance from the vertices of the mesh c_k to the mesh c_{avg} by computing for each vertex in c_k the closest point in c_{avg}. A large distance means that there was no point corresponding to it in the other contours, an indication of asymmetry. Note that the `Metro` tool cannot compute distances between contours directly from their original spatial coordinates. It requires both the meshes to be spatially aligned. The distance computed at each vertex using `Metro` is stored as a scalar field of c_k and the procedure is repeated for each contour. The asymmetry in the contours can then be located by visualizing the distance fields. Figure 7.7 shows a cryo-EM data set in which the top and bottom regions are symmetric. The procedure above is applied to two contours marked as symmetric by our method. Observe that the tip of the long club-like portion of the top contour is thin while that of the bottom contour is fat. This asymmetry can be distinguished from the dark red regions that depict vertices with large distance values.



(a) (b) (c) (d)

Figure 7.7: Asymmetry visualization. (a) Volume rendering of a cryo-EM data set (EMDB-1134) depicts two symmetric regions. (b) Two symmetric contours extracted by our algorithm shown in maroon and orange. The tip of the long club-like portion of the contours at the top and bottom is asymmetric. (c) The top contour is aligned with the bottom contour and a distance field is computed. (d) Visualization of the distance field. The dark red regions are asymmetric.

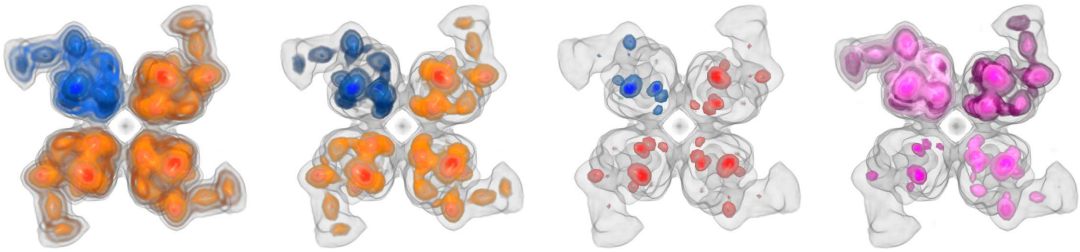


(a) (b) (c) (d)

Figure 7.8: Proximity-aware volume visualization. (a) A volume rendered image of cryo-EM data set EMDB-1603 and (b) its seed set. (c) The seed cells together with the Morse cells that lie in its proximity are extracted. The volume can also be cropped automatically to include only the selected features, thus reducing the size of the volume from $160 \times 160 \times 160$ to $92 \times 93 \times 50$. (d) The remaining Morse cells which correspond to noise in the data. The transfer function used for all the figures is identical.

7.5 Proximity-aware Feature Visualization

In this section, we describe an application that uses distances computed through augmented extremum graph traversal although it is not directly related to detecting symmetry. Distances from the seed cells to the remaining Morse cells computed during the augmented extremum graph traversal can be used for proximity-aware selection and visualization of features. A user can specify the proximity he is interested in and only those Morse cells that satisfy the proximity criteria, measured in terms of the distance from the seed cells will be selected and extracted. Thus a given set of features that the



(a) (b) (c) (d)

Figure 7.9: Symmetry-aware editing and rendering. (a)-(c) Two internal layers are peeled off from the segment shown in blue to reveal the features in the interior and the corresponding actions are automatically performed on the other three symmetric segments. (d) The symmetric regions are rendered differently by peeling different internal layers and shows complementary information in a single view.

user deems as important (seed cells) can be visualized together with only those features that lie in its proximity. We use this technique for separating the features from noise as shown in Figure 7.8(c) and Figure 7.8(d).

7.6 Symmetry-aware Editing and Rendering

Exploration of features in 3D scalar field data often involves significant effort from the user to interact with the volume and focus on the features of interest. Symmetry-aware segmentation obtained by our method can offer considerable assistance to users in performing such time consuming interactions. User can interact with one of the segments by applying different volume editing operations and the same operations can be automatically applied to the remaining segments. We show an illustration of this technique in the context of peeling features in a volume as shown in Figure 7.9(a)-(c). Observe that because of the symmetry in the data, each of the segment presents the same information. Instead, complementary information from different symmetric segments may be presented in a single view by rendering each segment differently [43]. We show this in Figure 7.9(d) where each segment is rendered at a different level of peeling.

7.7 Conclusions

In this chapter, we describe different applications of detecting symmetry in scalar fields. While some of the applications like symmetry-aware transfer function design, symmetry-aware isosurface extraction, and symmetry-aware editing and rendering enhance traditional visualization methods, the other applications like query driven exploration, asymmetry visualization, and proximity-aware visualization open up new avenues for the analysis and exploration of scalar fields. Until now, not much work has been done in developing applications of symmetry detection to domain specific problems. One of the main reasons behind this was the lack of robust and computationally efficient methods for symmetry detection. The methods proposed in this thesis overcome these challenges and we hope this will lead to new applications of symmetry detection for domain specific problems.

Chapter 8

Conclusions

Designing efficient and robust methods that detect symmetry in scalar fields in a feature-aware manner is a challenging problem. This thesis proposes three methods to address these challenges. Symmetry is detected by identifying similar subtrees of the contour tree, comparing distances using augmented extremum graph, and clustering contours in a high-dimensional shape descriptor space. We show through different applications that extracting symmetry information benefits visualization and exploration of scalar fields.

Although this thesis specifically addresses the problem of symmetry detection in scalar fields, it is easy to see that the methods we propose can be extended to analyze similarity between different data sets. Structural comparison of features is very useful in analyzing the behavior of the underlying scientific phenomenon in collections of related data sets like time varying data and ensemble data. For example, in the case of time varying data generated at a high frequency, there will be considerable redundancy in the data. It is a challenge to identify redundant time steps and remove them so that only key time steps are used for further analysis. A related problem is identifying if features in a time step repeat at periodic intervals. For ensemble data, it is important to characterize the correlation between the different runs of an experiment. Structural comparison between the features generated by each run is of great importance in such an analysis. We believe that the efficient and robust methods for similarity comparison that we have proposed in this thesis will immensely help in addressing these challenges.

Big data analysis is an emerging field in scientific data analysis. There is a lot of focus in developing new paradigms for exploring such data. We believe that methods like query driven exploration that we described in Section 7.1 will play a vital role in the visualization and exploration of big data. Similarity information extracted from a large dataset may also lead to research in compactly representing such data. It is however a major challenge to develop scalable methods for extracting similarity information from large datasets.

It is encouraging to note that in recent years several methods have been proposed for similarity analysis in scalar fields [3, 4, 82]. We hope this thesis will inspire the development of more such methods. While we have discussed some of the applications of our methods, we believe that applications of similarity analysis in scalar fields is still a largely untapped area. It is only a matter of time before new applications of similarity information are developed for better visualization and exploration of scalar fields.

References

[1] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity, and symmetries of geometric objects. *Discrete Comput. Geom.*, 3(3):237–256, 1988. (page 6).

[2] M. J. Atallah. On symmetry detection. *IEEE Trans. Computers*, 34(7):663–666, 1985. (page 6).

[3] U. Bauer, X. Ge, and Y. Wang. Measuring distance between Reeb graphs. In *Proc. Symposium on Computational Geometry*, pages 464–473, 2014. (pages 10, 104).

[4] K. Beketayev, D. Yeliussizov, D. Morozov, G. Weber, and B. Hamann. Measuring the distance between merge trees. In *Topological Methods in Data Analysis and Visualization*, pages 151–165. 2014. (pages 10, 104).

[5] A. Bermanis, A. Averbuch, and Y. Keller. 3-d symmetry detection and analysis using the pseudo-polar Fourier transform. *International Journal of Computer Vision*, 90(2):166–182, 2010. (page 7).

[6] 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. (page 7).

[7] A. Berner, M. Wand, N. Mitra, D. Mewes, and H. Seidel. Shape analysis with subspace symmetries. *Computer Graphics Forum*, 30(2):277–286, 2011. (page 7).

[8] J. Beyer, A. Al-Awami, N. Kasthuri, J. W. Lichtman, H. Pfister, and M. Hadwiger. Connectomeexplorer: Query-guided visual analysis of large volumetric neuroscience data. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2868–2877, 2013. (page 91).

[9] 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. (page 17).

[10] S. Biasotti and S. Marini. Sub-part correspondence using structure and geometry. In *Proc. Eurographics Italian Chapter Conference*, pages 23–28, 2006. (page 7).

[11] 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. (page 17).

[12] M. Bokeloh, A. Berner, M. Wand, H. Seidel, and A. Schilling. Symmetry detection using feature lines. *Computer Graphics Forum*, 28(2):697–706, 2009. (pages 7, 8, 38).

[13] G. Boothroyd, P. Dewhurst, and W. Knight. *Product Design for Manufacture and Assembly, Third Edition.* CRC Press, 2010. (page 1).

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

[15] P. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. A topological hierarchy for functions on triangulated surfaces. *IEEE Trans. Vis. Comput. Graph.*, 10(4):385–396, 2004. (page 36).

[16] 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. (pages 15, 17).

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

[18] 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. (page 15).

[19] H. Carr, J. Snoeyink, and M. van de Panne. Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. *Computational Geometry*, 43(1):42–58, 2010. (pages 15, 73, 74).

[20] F. Cazals, F. Chazal, and T. Lewiner. Molecular shape analysis based upon the Morse-Smale complex and the Connolly function. In *Proc. Symposium on Computational Geometry*, pages 351–360, 2003. (page 36).

[21] F. Chazal, L. J. Guibas, S. Y. Oudot, and P. Skraba. Analysis of scalar fields over point cloud data. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 1021–1030, 2009. (page 92).

[22] M. Chertok and Y. Keller. Spectral symmetry analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(7):1227–1238, 2010. (page 7).

[23] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998. (page 99).

[24] 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. (page 18).

[25] C. D. Correa, P. Lindstrom, and P.-T. Bremer. Topological spines: A structure-preserving visual representation of scalar fields. *IEEE Trans. Vis. Comput. Graph.*, 17(12):1842–1851, 2011. (pages 3, 10, 11, 14, 36, 39, 92).

[26] M. de Graef and M. E. McHenry. *Structure of Materials*. Cambridge University Press, 2007. (page 1).

[27] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. C. Hart. Spectral surface quadrangulation. *ACM Trans. Graph.*, 25(3):1057–1066, 2006. (page 36).

[28] H. Doraiswamy and V. Natarajan. Output-sensitive construction of Reeb graphs. *IEEE Trans. Vis. Comput. Graph.*, 18(1):146–159, 2012. (page 34).

[29] H. Doraiswamy and V. Natarajan. Computing Reeb graphs as a union of contour trees. *IEEE Trans. Vis. Comput. Graph.*, 19(2):249–262, 2013. (page 34).

[30] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010. (page 11).

[31] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Local and global comparison of continuous functions. In *IEEE Visualization*, pages 275–280, 2004. (page 9).

[32] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete & Computational Geometry*, 30(1):87–107, 2003. (page 13).

[33] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002. (pages 13, 43).

[34] EMDataBank. `http://www.emdatabank.org/`. (pages 16, 56).

[35] D. Ghosh, N. Amenta, and M. M. Kazhdan. Closed-form blending of local symmetries. *Computer Graphics Forum*, 29(5):1681–1688, 2010. (page 7).

[36] 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. (page 7).

[37] A. Gyulassy, M. Duchaineau, V. Natarajan, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann. Topologically clean distance fields. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1432–1439, 2007. (page 36).

[38] A. Gyulassy and V. Natarajan. Topology-based simplification for feature extraction from 3d scalar fields. In *IEEE Visualization*, pages 535–542, 2005. (page 92).

[39] M. Haidacher, S. Bruckner, and M. E. Gröller. Volume analysis using multimodal surface similarity. *IEEE Trans. Vis. Comput. Graph.*, 17(12):1969–1978, 2011. (page 9).

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

[41] 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. (pages 9, 17).

[42] U. Homberg, D. Baum, S. Prohaska, U. Kalbe, and K. J. Witt. Automatic extraction and analysis of realistic pore structures from μCT data for pore space characterization of graded soil. In *Proc. 6th Int. Conference on Scour and Erosion (ICSE-6)*, pages 66–73, 2012. (page 36).

[43] Y. Hong and H.-W. Shen. Parallel reflective symmetry transformation for volume data. *Computers & Graphics*, 32(1):41–54, 2008. (pages 8, 91, 101).

[44] H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *IEEE Visualization*, pages 59–66, 1999. (page 79).

[45] X. Y. Jiang and H. Bunke. A simple and efficient algorithm for determining the symmetries of polyhedra. *CVGIP: Graph. Models Image Process.*, 54(1):91–95, 1992. (page 6).

[46] G. Johansson, K. Museth, and H. Carr. Flexible and topologically localized segmentation. In *EuroVis*, pages 179–186, 2007. (page 17).

[47] 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. (pages 7, 8).

[48] M. M. Kazhdan, T. A. Funkhouser, and S. Rusinkiewicz. Symmetry descriptors and 3D shape matching. In *Proc. Symposium on Geometry Processing*, pages 117–126, 2004. (page 7).

[49] J. Kerber, M. Wand, J. Krüger, and H. Seidel. Partial symmetry detection in volume data. In *Vision, Modeling, and Visualization*, pages 41–48, 2011. (page 8).

[50] V. G. Kim, Y. Lipman, X. Chen, and T. A. Funkhouser. Möbius transformations for global intrinsic symmetry analysis. *Computer Graphics Forum*, 29(5):1689–1700, 2010. (page 40).

[51] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2013. (page 80).

[52] B. Kuo, W. Wang, C. Bruyere, T. Scheitlin, and D. Middleton. IEEE Visualization 2004 contest data set. `http://vis.computer.org/vis2004contest/data.html`. (page 93).

[53] D. Laney, P. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Trans. Vis. Comput. Graph.*, 12(5):1053–1060, 2006. (page 36).

[54] T. Lewiner, H. Lopes, and G. Tavares. Applications of Forman's discrete Morse theory to topology visualization and mesh compression. *IEEE Trans. Vis. Comput. Graph.*, 10(5):499–508, 2004. (page 36).

[55] Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoué, H. Van Nguyen, R. Ohbuchi, et al. A comparison of methods for non-rigid 3d shape retrieval. *Pattern Recognition*, 46(1):449–461, 2013. (pages 73, 75, 78, 79, 80).

[56] Y. Lipman, X. Chen, I. Daubechies, and T. Funkhouser. Symmetry factored embedding and distance. *ACM Trans. Graph.*, 29(4):103, 2010. (pages 7, 65, 71, 75).

[57] Z. Liu, B. Jiang, and J. Heer. *imMens*: Real-time visual querying of big data. *Computer Graphics Forum*, 32(3):421–430, 2013. (page 91).

[58] S. J. Ludtke, P. R. Baldwin, and W. Chiu. Eman: semiautomated software for high-resolution single-particle reconstructions. *Journal of Structural Biology*, 128(1):82–97, 1999. (page 82).

[59] 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. (page 7).

[60] T. B. Masood, D. M. Thomas, and V. Natarajan. Scalar field visualization via extraction of symmetric structures. *The Visual Computer*, 29(6-8):761–771, 2013. (page 68).

[61] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image Vision Computing*, 22(10):761–767, 2004. (page 85).

[62] 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. (page 6).

[63] N. Mitra, M. Pauly, M. Wand, and D. Ceylan. Symmetry in 3D geometry: Extraction and applications. In *EUROGRAPHICS State-of-the-art Report*, pages 29–51, 2012. (pages 2, 7).

[64] N. J. Mitra, L. J. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM Trans. Graph.*, 25:560–568, 2006. (pages 7, 65, 67, 71).

[65] N. J. Mitra, L. J. Guibas, and M. Pauly. Symmetrization. *ACM Trans. Graph.*, 26(3), 2007. (pages 7, 71).

[66] S. Nagaraj, V. Natarajan, and R. S. Nanjundiah. A gradient-based comparison measure for visual analysis of multifield data. *Computer Graphics Forum*, 30(3):1101–1110, 2011. (page 9).

[67] M. Niethammer, M. Reuter, F.-E. Wolter, S. Bouix, N. Peinecke, M.-S. Koo, and M. E. Shenton. Global medical shape analysis using the laplace-beltrami spectrum. In *Medical Image Computing and Computer-Assisted Intervention*, pages 850–857, 2007. (page 78).

[68] 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.*, 17(11):1547–1559, 2011. (page 17).

[69] 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. (pages 13, 48).

[70] 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. (page 34).

[71] 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. (pages 7, 65).

[72] O. Pele and M. Werman. Fast and robust earth mover's distances. In *IEEE International Conference on Computer Vision*, pages 460–467, 2009. (page 48).

[73] U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993. (page 79).

[74] J. Podolak, A. Golovinskiy, and S. Rusinkiewicz. Symmetry-enhanced remeshing of surfaces. In *Proc. Symposium on Geometry Processing*, pages 235–242, 2007. (pages 7, 65).

[75] 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. (page 7).

[76] Z. Qin, J. Jia, and J. Qin. Content based 3d model retrieval: A survey. In *International Workshop on Content-Based Multimedia Indexing*, pages 249–256, 2008. (page 73).

[77] D. Raviv, A. Bronstein, M. Bronstein, and R. Kimmel. Full and partial symmetries of non-rigid shapes. *International Journal of Computer Vision*, 89(1):18–39, 2010. (pages 40, 71).

[78] M. Reuter, F.-E. Wolter, and N. Peinecke. Laplace–beltrami spectra as shape-dna of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006. (pages 78, 79, 80).

[79] M. Reuter, F.-E. Wolter, M. Shenton, and M. Niethammer. Laplace–beltrami eigenvalues and topological features of eigenfunctions for statistical shape analysis. *Computer-Aided Design*, 41(10):739–755, 2009. (page 78).

[80] S. Rosenberg. *The Laplacian on a Riemannian Manifold*. Cambridge University Press, 1997. (page 78).

[81] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, Nov. 2000. (page 48).

[82] H. Saikia, H. Seidel, and T. Weinkauf. Extended branch decomposition graphs: Structural comparison of scalar data. *Computer Graphics Forum*, 33(3):41–50, 2014. (pages 10, 104).

[83] D. Schneider, C. Heine, H. Carr, and G. Scheuermann. Interactive comparison of multifield scalar data based on largest contours. *Computer Aided Geometric Design*, 30(6):521–528, 2013. (page 9).

[84] 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. (page 9).

[85] J. Schreiner, C. Scheiclegger, and C. Silva. High-quality extraction of isosurfaces from regular and irregular grids. *IEEE Trans. Vis. Comput. Graph.*, 12(5):1205–1212, 2006. (page 79).

[86] J. Schreiner, C. E. Scheidegger, S. Fleishman, and C. T. Silva. Direct (re)meshing for efficient surface processing. *Computer Graphics Forum*, 25(3):527–536, 2006. (page 79).

[87] S. Shafii, S. Dillard, M. Hlawitschka, and B. Hamann. The topological effects of smoothing. *IEEE Trans. Vis. Comput. Graph.*, 18(1):160–172, 2012. (page 10).

[88] 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. (page 7).

[89] R. Stevenson and J. Hall. *Human Malformations And Related Anomalies*. Oxford University Press, 2006. (page 1).

[90] D. Tahmoush and H. Samet. An improved asymmetry measure to detect breast cancer. In *Proc. SPIE 6514, Medical Imaging 2007: Computer-Aided Diagnosis*, 2007. (page 99).

[91] J. W. Tangelder and R. C. Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia tools and applications*, 39(3):441–471, 2008. (page 73).

[92] D. M. Thomas and V. Natarajan. Symmetry in scalar field topology. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2035–2044, 2011. (pages 3, 5, 10, 38).

[93] D. M. Thomas and V. Natarajan. Detecting symmetry in scalar fields using augmented extremum graphs. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2663–2672, 2013. (pages 4, 5, 10).

[94] D. M. Thomas and V. Natarajan. Multiscale symmetry detection in scalar fields by clustering contours. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2427–2436, 2014. (pages 4, 5, 10).

[95] S. Thrun and B. Wegbreit. Shape from symmetry. In *Proc. IEEE International Conference on Computer Vision*, pages 1824–1831, 2005. (page 7).

[96] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Topology driven 3D mesh hierarchical segmentation. In *Proc. Shape Modeling International*, pages 215–220, 2007. (page 17).

[97] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Partial 3D shape retrieval by Reeb pattern unfolding. *Computer Graphics Forum*, 28(1):41–55, 2009. (page 17).

[98] D. M. Ushizima, D. Morozov, G. H. Weber, A. G. C. Bianchi, J. A. Sethian, and E. W. Bethel. Augmented topological descriptors of pore networks for material science. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2041–2050, 2012. (page 36).

[99] O. Van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or. A survey on shape correspondence. *Computer Graphics Forum*, 30(6):1681–1707, 2011. (page 73).

[100] 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 *Proc. Symposium on Computational Geometry*, pages 212–220, 1997. (pages 3, 10, 11, 15).

[101] 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. (page 17).

[102] 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. (pages 17, 18, 97).

[103] 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. (page 6).

[104] K. Xu, H. Zhang, W. Jiang, R. Dyer, Z.-Q. Cheng, L. Liu, and B. Chen. Multi-scale partial intrinsic symmetry detection. *ACM Trans. Graph.*, 31(6):181, 2012. (pages 7, 71).

[105] 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. (pages 7, 65, 71).

[106] I. Yamazaki, V. Natarajan, Z. Bai, and B. Hamann. Segmenting point-sampled surfaces. *The Visual Computer*, 26(12):1421–1433, 2010. (page 36).

[107] R. Zakia. *Perception and imaging*. Focal Press, 2001. (page 52).

[108] E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. *ACM Trans. Graph.*, 24(1):1–27, 2005. (page 17).

[109] X. Zhang. Complementary shape comparison with additional properties. In *IEEE Proceedings of Volume Graphics*, pages 79–86, 2006. (page 9).

[110] X. Zhang, C. L. Bajaj, and N. A. Baker. Affine invariant comparison of molecular shapes with properties. In *ICES Tech Report*, 2005. (page 9).

[111] 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. (page 9).

[112] 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. (pages 17, 97).

[113] A. J. Zomorodian. *Topology for Computing*. Cambridge University Press, 2005. (page 11).