Computing Contour Trees for 2D Piecewise Polynomial Functions

A THESIS

SUBMITTED FOR THE DEGREE OF

Master of Science (Engineering)

IN THE

Faculty of Engineering

BY

Girijanandan Nucha

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

January, 2018

DEDICATED TO

My family

Acknowledgements

I thank my adviser Prof. Vijay Natarajan for the constant support and guidance throughout my M.Sc tenure. It has been a steep learning curve for me since I began my work under Vijay. His clarity of the problem and systematic way of conducting research has motivated me deeply. I sincerely appreciate his patience during difficult times. I also thank the chairman, Prof. Jayant Haritsa for creating a positive research environment. Courses at IISc were well organized and contained state of the art research. I enjoyed being taught by Prof. Vijay Natarajan , Prof. Sathish Govindarajan, Prof. Vittal Rao, Dr. Arpita Patra and Dr. Bhavana Kanukurthi and am thankful to them. Special thanks to Prof. Georges-Pierre Bonneau and Prof. Stefanie Hahmann from INRIA, France for the active collaboration and warm hospitality, when I was visiting Grenoble. I also thank IFCAM(CEFIPRA) for funding my visit to Grenoble. I thank Talha Bin Masood for helping me out when I was stuck and providing much needed data. Working with my lab mates Akash, Ramanpreet, Raghavendra and Adithya has been awesome. I thank my Mom, dad, sister and brother for the faith they had in me and finally I thank my soul-mate Shweta for being there for me always.

Abstract

Contour trees are extensively used in scalar field analysis. The contour tree is a data structure that tracks the evolution of level set topology in a scalar field. Scalar fields are typically available as samples at vertices of a mesh and are linearly interpolated within each cell of the mesh. A more suitable way of representing scalar fields, especially when a smoother function needs to be modeled, is via higher order interpolants. We propose an algorithm to compute the contour tree for such functions. The algorithm computes a local structure by connecting critical points using a numerically stable monotone path tracing procedure. Such structures are computed for each cell and are stitched together to obtain the contour tree of the function. The algorithm is scalable to higher degree interpolants whereas previous methods were restricted to quadratic or linear interpolants. The algorithm is intrinsically parallelizable and has potential applications to isosurface extraction.

Publications based on this Thesis

Girijanandan Nucha, Georges-Pierre Bonneau, Stefanie Hahmann, and Vijay Natarajan, *Computing contour trees for 2D piecewise polynomial functions*, Computer Graphics Forum (EuroVis 2017), 2017, In Press.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

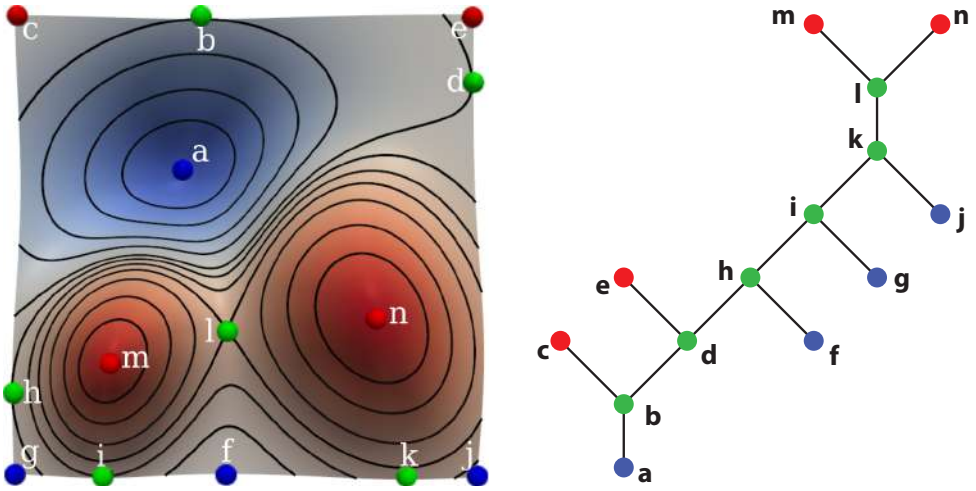
Scientists and engineers are increasingly using higher-order FEM simulations. Consider, as an example, the *h*p-adaptive variant [3] of finite element methods for which Nektar++ [5], Concepts [37], and Hermes [39] are three among many existing open source software packages and libraries. These methods rely on piecewise polynomial approximations, using elements (possibly curvilinear) of variable size h and polynomials of order p within an element. Higher order elements are suitable for efficient parallel implementations and allow for higher numerical accuracy and convergence than linear basis functions by either adaptively reducing the element's size h, by increasing the polynomial order p, or by combining both approaches. For an equivalent number of degrees of freedom, one can obtain the same level of accuracy with fewer elements.

Data in science and engineering applications is often available as a scalar field. **Scalar fields** are real-valued functions, which provides a way of visualizing the data.

In this thesis, we study piecewise higher-order scalar functions defined on planar or curvilinear elements representing 2-manifold geometries.

1.1 Motivation and related work

Whereas higher-order discretizations have become a widely accepted tool for many applications, visualization techniques have to adapt and better exploit the non-linear and often polynomial nature of the data sets. Standard visualization techniques such as contouring, volume rendering, and topology-based methods assume the basis functions to be linear. These methods first create compatible linear approximations of the geometry as well as of the higher-order data generally through adaptive tessellation [25, 35] in order to increase numerical accuracy and topological fidelity. Since the basis functions used to represent geometry and the attribute field functions are not necessarily the same nor of same polynomial order, accurate tessellation is a challenging task [38]. Further, the use of high



(a) (b)

Figure 1.1: (a) A 2D scalar field with two maxima (red) in the interior, two maxima on the boundary, one minimum (blue) in the interior, and three minima on the boundary. (b) Contour tree of the scalar field, whose nodes are exactly the critical points of the scalar field.

sampling density increases memory usage and may still introduce error.

In recent years, many visualization techniques for higher order fields have begun to emerge [13, 47, 30, 38, 28, 32, 31]. Whereas these contouring, particle tracking, and rendering techniques for higher-order data seek for improving numerical accuracy of the visualization, topological fidelity is equally important. Indeed, topology-based methods are important for analysis and visualization of scalar data, since they provide abstract representations of key features in the data.

Our focus is on the computation of contour trees. The contour tree captures significant topological features of a data set by computing the nested relationships between the connected components of level sets in a scalar field, as illustrated in Figure 1.1. Contour trees are widely applied in the context of volume visualization – for efficient computation of isosurfaces [43], transfer function design [21, 40, 45, 48, 18], and for effective and flexible exploration of isosurfaces [8]. The application of contour tree to volume data analysis such as feature extraction and tracking [4, 46, 19], symmetry and similarity detection [42, 36] is also clearly demonstrated.

Many efficient algorithms have been proposed for computing the contour tree for two- and three-dimensional scalar fields [15, 43, 41, 7, 11] and considerable efforts have been undertaken to develop parallel implementations [33, 27, 26, 1, 9, 24]. However, these methods typically suppose the data being sampled at the mesh/grid vertices and varying linearly along the edges. In this paper, we tackle the problem of computing contour trees specifically dedicated to higher-order interpolants without falling back to linear approximations of the data.

Dillard et al. [16] described a method to compute the contour tree for quadratic interpolants. They proceed by first tessellating a triangle in the input mesh into monotone triangles and then apply classical methods for contour tree computation. This method does however not scale to higher order elements because it requires a case analysis for computing the tessellation. This case analysis is cumbersome already for quadratic interpolants. Pascucci and Cole-McLaughlin [33] and Acharya and Natarajan [1] describe parallel algorithms to compute the contour tree for piecewise trilinear interpolants over a 3D grid. Minima and maxima are restricted to vertices of the grid and there are only four possible join/split tree configurations. Both methods compute the join and split trees for a single grid cell by looking up a case table and stitch them together. Carr and Snoeyink [6] propose an abstract framework for handling interpolants of arbitrary order and design a finite state automaton for computing the contour tree. This framework was employed either explicitly or implicitly for computing the contour tree in parallel for both trilinear interpolants [33, 1] and for piecewise linear interpolants [27, 26]. These algorithms used combinatorial routines to compute the tree corresponding to an individual cell. Such an approach is not feasible for higher order interpolants. Our method may be considered as the first concrete realization of this abstract framework for higher order interpolants.

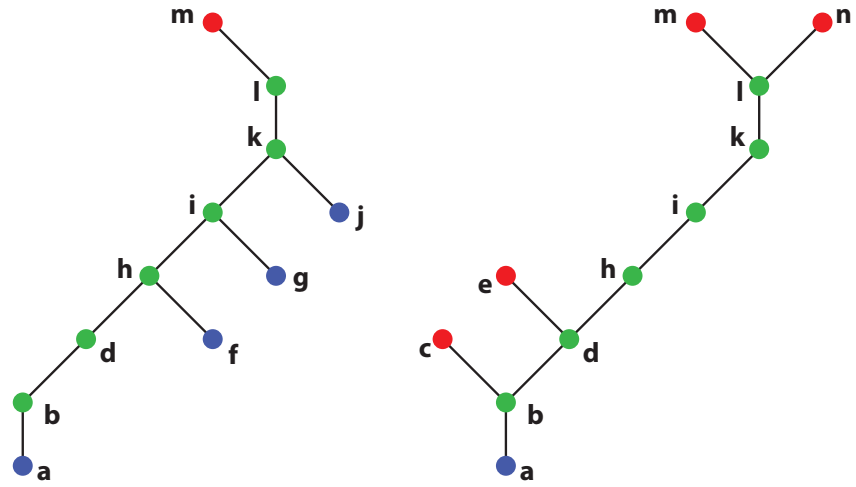
1.2 Summary of results

Our algorithm has two phases: local and global. The local phase computes the contour tree restricted to a single triangle by exploiting the monotone connectivity of critical points within an element. Inspired by the approach from Chiang et al. [11], we compute monotone paths on the polynomial function to connect the critical points inside an element. This leads to an advantageous dimension reduction of all involved sub-problems because the restriction of a bivariate polynomial function to a specific direction reduces to a univariate polynomial. The global phase stitches together the local trees and hence produces the contour tree.

This algorithm may be considered as a hybrid approach between the monotone path tracing algorithm of Chiang et al. [11] and the two-pass union-find based algorithm of Carr et al. [7]. Both algorithms have to be suitably extended to be made applicable to higher order polynomials. Our algorithm finally combines the advantages of both approaches. The algorithm of Chiang et al. is output sensitive and does not require processing all sample points. It is appropriate for our local tree computation because we enjoy the benefit of sampling the polynomial only along the monotone paths as opposed to sampling uniformly over all triangles. The global stitching procedure is a combinatorial and computationally efficient algorithm.

Our algorithm is designed to work for any higher-order interpolant. We demonstrate the effectiveness of our approach with an implementation for piecewise-polynomials of degree two,three and five. We also performed experiments on real-world scientific data obtained using COMSOL [12] and we

present our findings in Chapter 5. We back up our claims regarding the advantages of using higher order interpolants by evidences obtained from experiments performed on various synthetic datasets of different degrees of interpolation.



(a) (b)

Figure 1.2: Join tree (a) and split tree (b) of the function described in Figure 1.1. The contour tree is the union of the join and split trees.

1.3 Contributions

Following are the major contributions of the thesis.

- We propose a novel algorithm to compute contour trees for 2D piecewise polynomial functions of any degree of interpolation.
- We prove correctness and running time complexity for the proposed algorithm and showcase results obtained by a basic implementation.
- We demonstrate the advantages of higher order polynomial functions over their linear counterparts by a series of case studies and observations done over real and synthetic datasets respectively, of different degrees of interpolation.

1.4 Organization

We discuss the concepts and definitions required for understanding the thesis in Chapter 2. We present our proposed algorithm with apt illustrations, detailed pseudocode and rigorous analysis in Chapter 3.

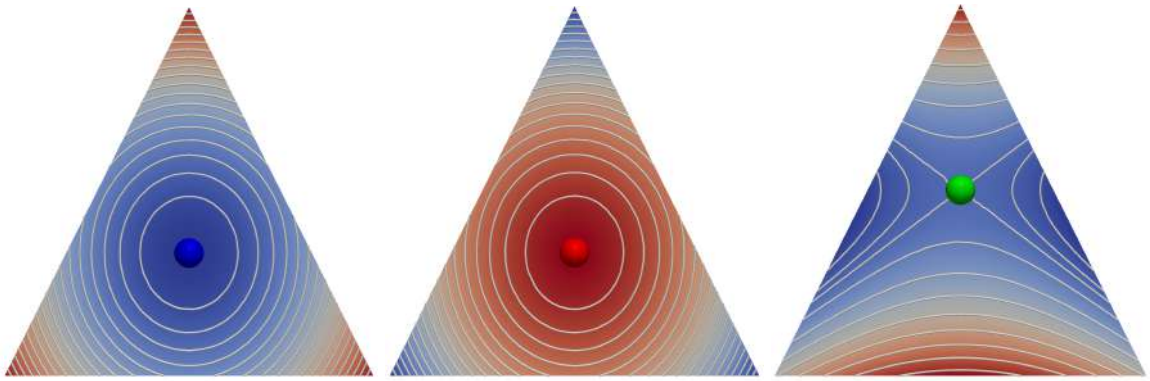
Chapter 4 gives insight into the implementation specific details such as, programming language, libraries and software tools used. Chapter 5 enumerates implementation details and experimental results. It also illustrates interesting case studies and observations. The gist of a prior work which tried to solve the same problem by analyzing all combinatorial cases of critical point types is presented in Chapter 6. Finally, Chapter 7 concludes the work presented in the thesis.

Chapter 2

Background

In this chapter, we introduce the necessary background needed to understand the rest of the thesis. Specifically, we define terms like level set, contour, critical point, Morse function, etc.

Let $\mathbb{M}$ be a d-manifold with or without boundary. Let $f : \mathbb{M} \longrightarrow \mathbb{R}$ be a smooth (C^∞ differentiable) real function. A point $p \in \mathbb{M}$ is called a **critical point** if $\nabla f(p) = 0$. Figure 2.1 shows minima/maxima/saddle occuring on the surface of a triangle with a bi-quadratic polynomial function defined on it. A minimum has the lowest function value in its local neighbourhood. Similarly, a maximum has the highest function value in its local neighbourhood.



(a) (b) (c)

Figure 2.1: (a) Minima. (b) Maxima. (c) Saddle

A critical point at which the Hessian matrix is non-singular is called a **non-degenerate** critical point. The function f is said to be a **Morse function** [29] iff:

- its critical points are non-degenerate and lie in the interior of $\mathbb{M}$,
- the critical points of f restricted to the boundary of $\mathbb{M}$ are non-degenerate,

- the critical values (values of f at critical points in the interior and the boundary of $\mathbb{M}$) are distinct.

A **level set** is the preimage $f^{-1}(c)$ of a real value c chosen from the range of f, which is called an **isovalue**. The connected components in a level set are called **contours**. The quotient space of $\mathbb{M}$ by the equivalence relation "a relates to b if a and b lie within the same contour" is a graph called the **Reeb graph** [34]. If the domain $\mathbb{M}$ is simply connected (genus 0) then the Reeb graph is acyclic and called the **contour tree**.

Nodes in a contour tree correspond to critical points of f. Maxima and minima are leaves of the tree, while index-1 and index-(d-1) saddles are degree-3 nodes. This follows from the fact that f is a Morse function and hence all critical points are non-degenerate. Two level set components merge at an index-1 saddle. Similarly a level set component splits into two at an index-(d-1) saddle. In order to explain the structure of the contour tree, it is convenient to use the metaphor of contours 'appearing', 'disappearing' or 'merging'. A contour appears when the isovalue increases past the critical value of a minimum. A contour disappears when the isovalue increases past the critical value of a maximum. When the isovalue increases past the critical value of a saddle, either one contour splits into two or two contours merge into one. Figure 1.1 shows a 2D function and the corresponding contour tree. Nodes colored red, blue, and green correspond to maxima, minima, and saddle, respectively. The contour tree provides the user with direct insight into the topology of all level sets and reduces the time required to understand the topological structure of the data. The **join tree** and **split tree** illustrated in Figure 1.2 are defined in a manner analogous to the contour tree, by considering respectively the sub-level sets (pre-image of $f^{-1}(-\infty, c]$) and the super-level sets (pre-image of $f^{-1}[c, +\infty)$). They are useful intermediate structures for computing the contour tree as explained in Section 3.2 and Section 3.6.

Chapter 3

Algorithm

In this chapter we describe an algorithm for computing the contour tree of a 2D piecewise polynomial function.

3.1 Input

The domain is represented by a triangle mesh of genus 0. We require the input data to be continuous across the domain so that the level sets are also continuous, see Figure 3.1. The polynomial interpolant is specified by a set of samples in each triangle. We use the word **patch** to refer to a triangle together with the polynomial interpolant as specified by the set of samples. A sample is specified by two parameter values for the location and one function value. The number of samples depends on the degree of the polynomial function. For example, a polynomial of maximal degree 2 is defined by 6 samples, whereas a maximal degree-3 polynomial requires 10 samples. If required, the monomial coefficients of the polynomial may be computed from the samples by solving a linear system. Typically, three samples are located at vertices of the triangle. The location of the remaining samples depends on the particular finite element implementation. Figure 3.1b shows a quadratic polynomial defined by six sample points (black). For comparison, the piecewise linear interpolant defined by the same set of six sample points is shown in Figure 3.1a. Figure 3.1d shows two quadratic patches defined by 6 sample points each. The functions are continuous across the common boundary as the continuous isolines indicate. The geometry of the element may also be modeled as a polynomial function, possibly of different order than the attribute data. In this case, a second set of samples is required, consisting of two parameter values and a 3D position per sample.

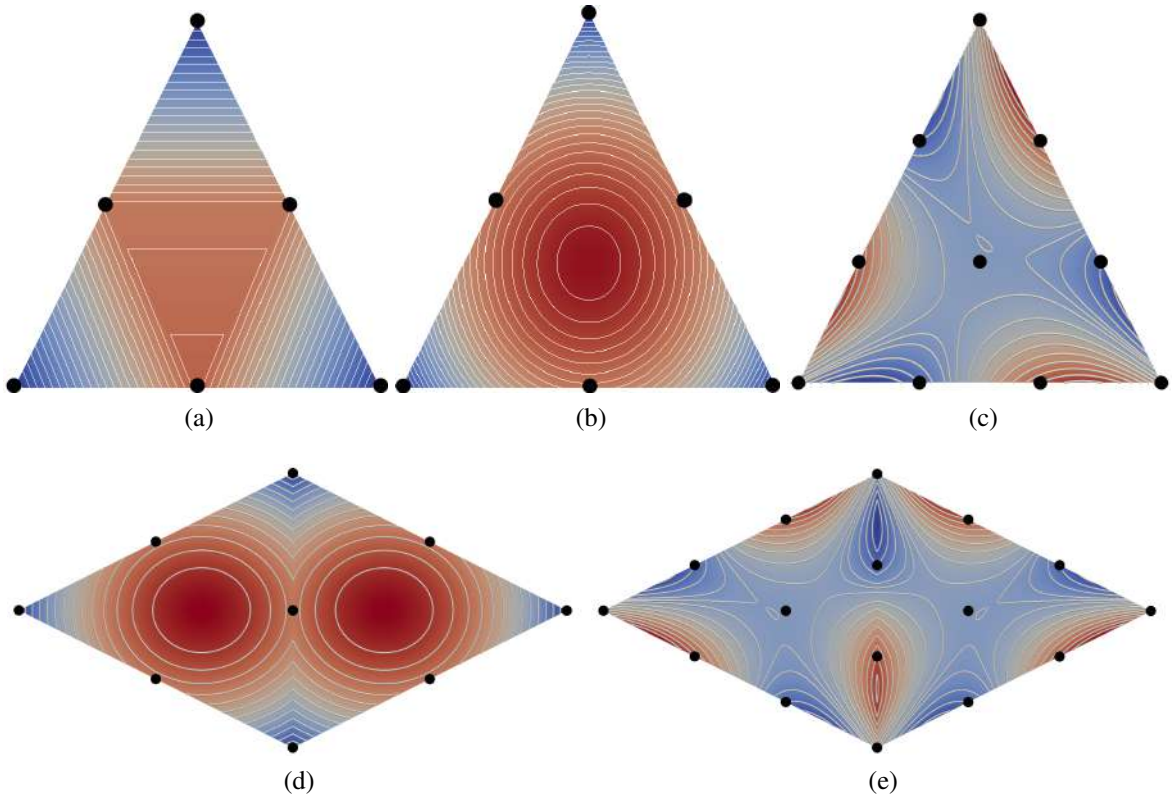


Figure 3.1: Linear and higher order interpolation over a triangle for a function sampled at multiple points (black). (a) Linear interpolation within each of the four triangles obtained by subdividing the input triangle. (b) Quadratic polynomial function interpolating the sample points. (c) Cubic polynomial function interpolating 10 sampling points. (d) Piecewise continuous quadratic polynomial sampled at 6 points each over two triangles. (e) Piecewise continuous cubic polynomial sampled at 10 points within each triangle. Isolines are continuous across common boundary.

3.2 Overview

The algorithm proceeds by building the local join and split tree of a patch independent of other patches. A **local join tree** captures the connectivity of sub-level sets of a patch, where topology change events are associated with local minima and join saddles. Similarly, the **local split tree** captures the connectivity of super-level sets of a patch. Topology change events here are associated with local maxima and split saddles. The local join trees of adjacent patches are stitched together to obtain the global join tree of the input scalar field. Similarly, the local split trees are stitched together into the global split tree. Finally, the global join and split trees are merged using an efficient tree merge procedure to obtain the desired contour tree of the input scalar field. Essentially, the algorithm contains four steps:

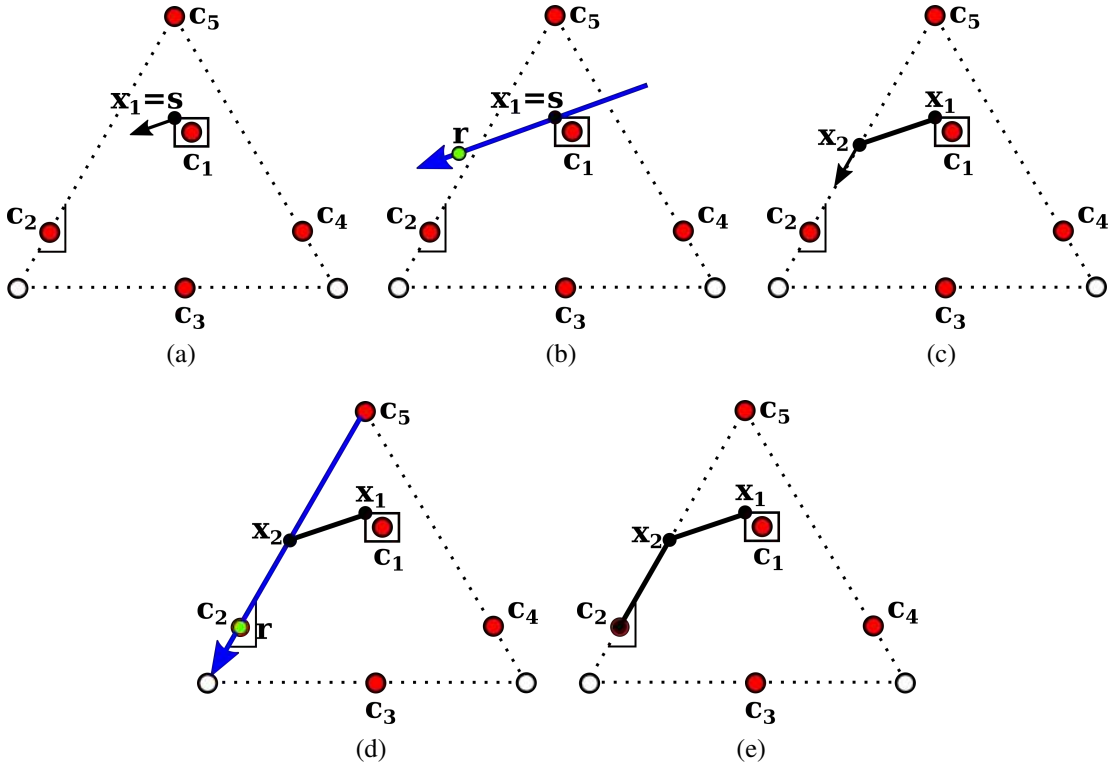


Figure 3.2: Illustration of path tracing for a monotone descending path.

1. Compute critical points for each patch.
2. Generate local join tree and split tree for each patch.
3. Stitch local join trees together to obtain the global join tree. Stitch local split trees to obtain the global split tree.
4. Merge the global split and join trees together to obtain the contour tree.

Prior to processing a patch, we apply a rigid body transformation that moves the triangle to the XY-plane. This transformation simplifies future numerical computations without affecting the topology of the level sets and hence the local join and split trees.

3.3 Critical points of a patch

The critical points of a 2D polynomial are computed by solving for the roots of a polynomial system given by the two partial derivatives. Analytic methods are not available to compute roots of such a polynomial system, particularly for higher degrees. We use PHCpack [44] for computing the roots of the polynomial system. PHCpack uses homotopy continuation methods and provides exact roots

whenever they are computable and numerically approximate solutions when the root finding is intractable. The method reports points lying in the interior of the triangle. **Line-critical points**, defined as critical points lying on the triangle boundary, are identified by first computing the restriction of the polynomial to each bounding line. Note that the restrictions are univariate polynomials.

3.4 Local join and split trees

The critical points computed in the previous step together with the triangle vertices constitute the potential nodes of the local join and split tree of a patch. We next compute the arcs of these local trees. Below, we describe the algorithm for computing the local join tree (Algorithm 1). The local split tree is computed using a similar procedure. The algorithm assumes that the polynomial f defined on a triangle together with the set of its critical points is available as input.

Most methods to compute the join tree for piecewise linear functions explicitly track the connected components of sub-level sets during a sweep over the domain and process all vertices of the input mesh [7, 19]. This approach does not extend well to higher order interpolants due to two reasons. First, the critical points of a piecewise linear function are necessarily located at vertices of the input mesh and hence it is sufficient to process vertices of the mesh. However, the critical points of a higher order polynomial interpolant may lie in the interior of the triangle. Second, computing the level sets for higher order interpolants is challenging both in terms of the computational cost and numerical accuracy. We instead employ an approach that directly computes the downward arcs incident on a join tree node. The potential nodes of the join tree are processed in increasing order of function value and the downward arcs incident on it are identified by following monotone descending paths from the corresponding critical point / triangle vertex. A **monotone descending path** (MDP) is a path in the patch along which the value of f decreases monotonically. The path originates at a critical point / triangle vertex and terminates at a different critical point / triangle vertex or merges into another descending path. The MDP terminates in the sub-level set component that contains its origin. Hence, the MDP helps determine the downward arcs from the corresponding join tree node.

The critical points and triangle vertices are processed in increasing order of value of f. We maintain a forest of join trees containing all critical points and vertices processed so far. When the next critical point or vertex c is processed, the forest is updated to include c. The local neighborhood of c is classified into regions where f assumes values lower or higher than $f(c)$. This classification determines the number of MDPs traces from c. If an MDP terminates at a critical point c' then we insert an arc from c to the root of the tree containing c'. Alternatively, if the MDP intersects a previously computed MDP, say originating at c'', then we insert an arc from c to the root of the tree containing c''. The forest of trees is stored as a union find data structure [14]. After all critical points and vertices are processed, the forest consists of a single tree rooted at the global maximum of the patch. This tree

is the local join tree of the patch.

Monotone path tracing. Computing the exact geometry of a monotone path is computationally challenging. However, it is sufficient for our purposes to compute sample points on the path. We represent an MDP as a piecewise linear curve and store it as a finite collection of points $[x_0, x_1, \ldots, x_k]$. Here, x_0 is the source critical point, x_k is the terminal point, and $f(x_{i+1}) < f(x_i)$. We now describe how an MDP is traced. Ensuring numerical accuracy while reducing computational costs makes this a challenging problem. In particular, (a) appropriate number of MDPs needs to be traced from a critical point / triangle vertex, (b) the tracing procedure should ensure the monotone property, and (c) the terminal point should be recognized correctly.

We reduce all numerical computations to root finding on univariate polynomials, thereby simplifying the computation. We use GNU Scientific Library for processing the univariate polynomials [22]. We first compute disjoint axis-aligned bounding boxes for all the critical points and vertices, to facilitate the identification of the number of MDPs originating at them. Chattopadhyay et al. [10] show that these boxes always exist. In practice, we compute boxes with diagonal length smaller than d_{min}, the minimum distance between critical points / triangle vertices. Assuming that f is a Morse function, the local neighborhood may be partitioned into sectors with values of f alternating between higher and lower than $f(c)$. We compute the restriction of f to the boundary of the bounding box R_c. This restriction is a univariate piecewise polynomial function f_{R_c}. Roots of $f_{R_c} - f(c)$ partition the boundary of R_c into segments. If there are more than four roots then we compute a smaller bounding box by halving the diagonal length. Choose a point s within each segment and initialize an MDP $[x_0 = c, x_1 = s]$ if $f(s) < f(c)$. Next, compute the restriction of f to the ray along the negative gradient direction at s. This restriction is also a univariate polynomial, say f_s. The minimum of f_s closest to s is inserted as the next point x_2 of the MDP. This process is repeated to compute subsequent points x_i on the MDP. This iterative procedure terminates either when x_i lies in the interior of the bounding box of a critical point / triangle vertex c' or when $x_{i-1}x_i$ intersects a previously computed MDP. The iterative procedure indeed computes a monotone descending path because the gradient at x_i is non-zero and hence $f(x_{i+1}) < f(x_i)$. So, it always terminates by reaching the interior of the bounding box of either a minimum or another critical point.

Example. Figure 3.2 illustrates the tracing of an MDP p from a critical point c_1 in a triangle. All the critical points of the patch are shown in red. The bounding box is shown only for critical points c_1 and c_2 to reduce clutter. The direction of steepest descent is shown using the arrow glyph. The polynomial f defined on the triangle is restricted to a ray along the steepest descent (Figure 3.2(b), shown in blue). Let r be the closest minimum of the resulting univariate polynomial. In this case, the point r lies outside the boundary of the patch. The univariate polynomial decreases monotonically until r. So, the point of intersection of the ray with the triangle boundary is chosen as the next point,

x_2 on the MDP. When an MDP reaches the triangle boundary, it is restricted to the boundary for simplicity. The steepest descent direction on the boundary is computed at x_2, shown using an arrow glyph in Figure 3.2(c). Next, the univariate polynomial along the blue ray is computed followed by locating the closest minimum r, see Figure 3.2(d). The tracing stops because r lies within the bounding box of c_2. The resulting path is $p = [x_0 = c_1, x_1 = s, x_2, x_3 = c_2]$.

3.5 Global join and split trees

The local join trees are stitched together resulting in the global join tree. Similarly the global split tree is computed by stitching the local split trees. The stitching procedure is similar to the one proposed by Acharya and Natarajan [1]. We describe it here for completeness, see also Algorithm StitchJoinTrees. The sorted list of nodes of two input join trees are merged into a single sorted list. Duplicate nodes from the boundary of the two corresponding sub-domains lie adjacent to each other in the sorted list. An edge is inserted between every pair of duplicate nodes, resulting in a single connected graph that may contain cycles. The stitched join tree is computed by sweeping the graph in increasing order of function values and tracking connected components of subgraphs using a union-find data structure. Repeated application of the stitching process on all local join trees and on intermediate stitched trees results in the global join tree.

Algorithm 1: BUILDLOCALJOINTREE

Input: Polynomial function f defined on a triangle that lies on the XY plane
Input: Set C of critical points(CP) and triangle vertices. Min distance, d_{min}, between CPs.
Output: Local join tree JT

/* Let P denote the set of monotone descending paths(MDP). Each $p \in P$ is of the form $[x_0, x_1, \ldots, x_k]$, where $x_0, x_k \in C$. */
/* Let R_c denote the bounding box(BB) of a point $c \in C$. */
/* Let S denote the set of starting points of MDPs. */
/* Let UF denote a union-find data structure to store collection of points in C. */

Initialize the node set of JT to C
Initialize $UF \longleftarrow \emptyset$, $P \longleftarrow \emptyset$
for each $c \in C$ in ascending order of function value **do**
    NewSet$(\{c\}, UF)$
    Compute an axis parallel bounding box, R_c with c as center and diagonal length d_{min}
    Compute roots of f restricted to the boundary of R_c
    Compute the collection S of points s between pairs of adjacent roots on the boundary of R_c that satisfy the condition $f(s) < f(c)$
    for each $s \in S$ **do**
        Start a monotone descending path p, initialize it to $[c]$
        Set $x_1 \longleftarrow s$, $i \longleftarrow 0$
        repeat
            Set $i \longleftarrow i+1$
            Append x_i to p
            Compute f_i, the restriction of f to the ray along the negative gradient of f at x_i
            Set x_{i+1} as the minimum of f_i that is closest to x_i
        until x_{i+1} lies within $R_{c'}$ for some $c' \in C$ OR the line segment $x_i x_{i+1}$ intersects a path $p' \in P$;
        if x_{i+1} lies within $R_{c'}$ **then**
            Append c' to p
            Add p to P
            Add edge cc' to JT
            Union(c, c', UF)
        end
        if $x_i x_{i+1}$ intersects a path $p' \in P$ at a point x **then**
            Append x to p
            if x is not a point in the representation of p' **then**
                Insert x into p' at the appropriate location
            end
            Add p to P
            Let c' denote the source critical point for p'
            if $c \neq$ Find(c', UF) **then**
                Add edge $(c,$ Find$(c', UF))$ to JT
                Union(c, c', UF)
            end
        end
    end
end
Return JT

Algorithm 2: StitchJoinTrees [1]

Input: Join trees JT_1 and JT_2 for two sub-domains that have a common boundary
Input: List of nodes of the two trees N_1 and N_2 sorted in ascending order of function value
Output: Stitched join tree JT

Initialize $JT \longleftarrow JT_1 \cup JT_2$
$UF \longleftarrow \emptyset$
$N \longleftarrow Merge(N_1, N_2)$ **for** $(i \longleftarrow 1 to |N|-1)$ **do**
    if v_i and v_{i+1} are redundant nodes on the common boundary **then**
        NewSet(v_i, UF)
        NewSet(v_{i+1}, UF)
        Union(v_i, v_{i+1}, UF) making v_{i+1} as the head
        JT.v_i.Parent $\longleftarrow v_{i+1}$
        Add v_i to JT.v_{i+1}.ChildrenList
    end
    for (each child c_j of v_i) **do**
        if c_j is present in UF **then**
            if v_i is present in UF **then**
                NewSet(v_i, UF)
            end
            Delete c_j from JT.v_i.ChildrenList
            $c' \longleftarrow FIND(c_j, UF)$
            if $v_i \neq c'$ **then**
                JT.c'.Parent $\longleftarrow v_i$
                Add c' to JT.v_i.ChildrenList
                Union(c', v_i, UF) ensuring v_i as the head
            end
        end
    end
end
Return Stitched join tree JT

3.6 Contour tree

The global join and split trees are merged using Algorithm 3 resulting in the global contour tree [7]. This merge step is also described in previous work. For completeness, the merging procedure is described in Algorithm MergeJoinAndSplitTree in the supplementary material. The algorithm maintains a set L of leaf nodes in the join and split trees and processes them in sequence. If the current leaf node under consideration, say l, is an unprocessed non-root node then the edge between l and its parent from the appropriate tree is added to the resulting contour tree. After processing, l is deleted both from the list L and the join/split tree that contained it. If this deletion results in the parent of l to

become a leaf node , then that node is inserted into L.

Algorithm 3: MERGEJOINANDSPLITTREE [1]

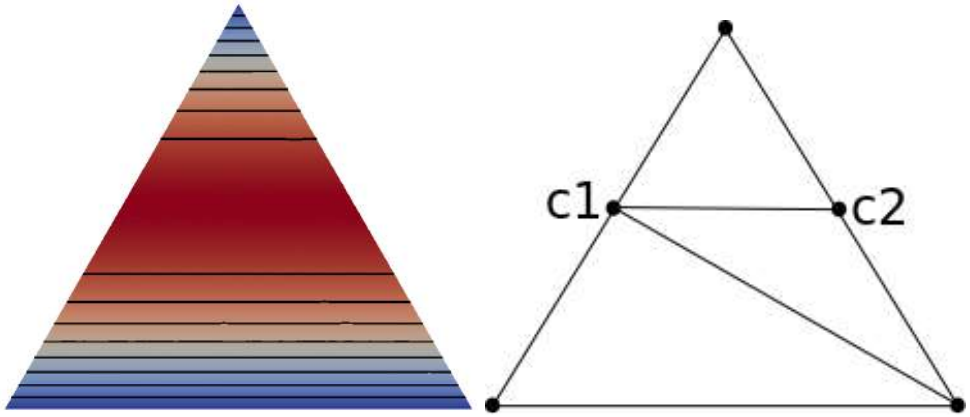
Input: Global join tree JT and split tree ST

Output: Contour tree CT

```
L ⟵ Set of leaves in JT and ST
while L ≠ φ do
    if l is a leaf in JT or ST then
        Process l and remove it from L
        T = tree in which l is a leaf
        while l ≠ T.root and l is not processed do
            n = l
            l = parent vertex of l in T
        end
        Remove n from T and L
        Add arc(n,l) to CT
        if l is either a leaf in JT or ST then
            Add l to L
        end
    end
end
```

3.7 Degeneracies

A patch that contains at least one non-isolated critical point is said to be degenerate. Figure 3.3(a) shows a degenerate function defined on a triangle. All points on the line c_1c_2 are local maxima. The function in this case is not Morse and hence the above algorithm does not apply. In particular, individual critical points cannot be isolated and the MDP tracing fails due to the presence of zero gradient regions. We compute the local join and split tree for the degenerate patch by subdividing it into smaller triangles and assuming linear interpolation within each smaller triangle. The degenerate patch is processed as follows. First, compute the line-critical points of the patch and insert them as vertices. Next, compute a triangulation of the set of triangle vertices and the newly inserted vertices. This triangulation subdivides the interior of the degenerate patch into smaller triangles. For example, Figure 3.3(b) shows the decomposition of the patch into three triangles. Assume linear interpolation within each triangle and compute the local join tree and local split tree of the patch using the sweep algorithm [7]. Inserting the line-critical points ensures that the subsequent stitching step applies to all patches, degenerate or otherwise, without modification.



(a) (b)

Figure 3.3: (a) A degenerate patch. (b) Subdividing the patch into triangles after inserting all line-critical points. The local join tree and local split tree is computed by assuming piecewise linear interpolation within each smaller triangle.

Notice that we only compute line critical(LC) points in case of degenerate patch. We claim that, any degeneracy in the polynomial function on the patch has to extend till the boundary of the triangle, which is in turn captured by the LC points as shown in Lemma 3.1 and Lemma 3.2.

Lemma 3.1 *If a bivariate polynomial function f is constant over a non-empty open subset Ω of $\mathbb{R}^2$, then f is constant over $\mathbb{R}^2$.*

Proof: Let $\mathbf{x}_0 \in \Omega$. The Taylor expansion of the polynomial function f at x_0 is

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \sum_{|\mathbf{k}|>0}^{\mathbf{k}=\mathbf{N}} \partial^{\mathbf{k}} f(\mathbf{x}_0) \frac{(\mathbf{x}-\mathbf{x}_0)^{\mathbf{k}}}{\mathbf{k}!},$$

where we use the multi-index notation, and $\mathbf{N} = (N_1, N_2)$ is the degree of f. Since f is constant within a neighborhood of $\mathbf{x}_0$, we must have $\partial^{\mathbf{k}} f(\mathbf{x}_0) = 0, \forall \mathbf{k}$. From the Taylor expansion of f, we conclude that f is constant over $\mathbb{R}^2$. □

Lemma 3.2 *If a bivariate polynomial function f is constant over a non-empty line segment I in $\mathbb{R}^2$, then f is constant over its supporting line $l(I)$.*

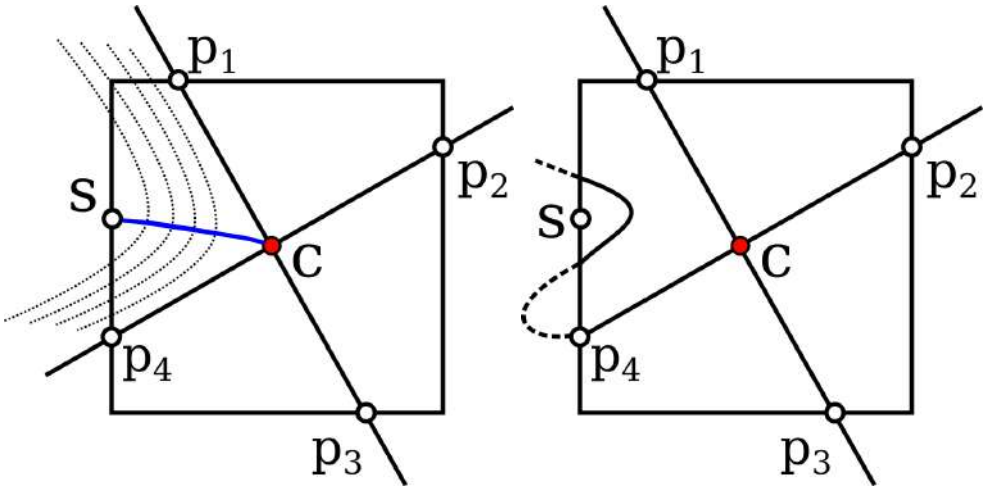
Proof: Let $\mathbf{x}_0$ and $\mathbf{x}_1$ be two distinct points in I. Define a univariate polynomial g such that $g(s) = f(\mathbf{x}_0 + s(\mathbf{x}_1 - \mathbf{x}_0))$ for $s \in \mathbb{R}$. The polynomial g is constant over a non-empty open interval. An analogous argument as in the proof of the previous lemma implies that g is constant over $\mathbb{R}$. Therefore, f is constant over the line $l(I)$. □

3.8 Correctness

We claim that the tree computed by the above algorithm is the contour tree of the piecewise-polynomial input. If the MDPs are computed accurately from all critical points of a patch then Algorithm 1 indeed computes the local join tree [11]. The algorithm traces all MDPs from a critical point as shown in Lemma 3.3. Degenerate patches are also processed correctly. If a patch contains a non-isolated critical point then the degenerate region extends to the boundary as shown in the appendix, see Lemma 3.1 and 3.2. If the degenerate region is a curve then the corresponding line-critical points are included, else the entire patch is flat. In either case, all critical points are included into the join tree and the degenerate patch is processed correctly.

Lemma 3.3 *Let f be a bivariate polynomial Morse function and let c be a critical point of f. Algorithm 1 traces at least one monotone descending path from each connected component of the lower neighborhood N_c^- of c.*

Proof:



(a) (b)

Figure 3.4: (a) Bounding box R_c for a critical point c showing its neighborhood N_c. (b) A box containing c, which is larger than the required bounding box.

The bounding box is chosen such that no other critical point lies within it. First, assume that the bounding box R_c is small enough that the level set $f^{-1}(c)$ intersects the boundary of R_c exactly zero times (for minima and maxima) or four times (for saddle critical points). In this case, Algorithm 1 clearly traces 0, 1, and 2 MDPs for minima, maxima, and saddle critical points, respectively. Further,

the path is indeed an MDP because there exists a descending path from c to the point s on the boundary, namely the gradient descent path (see Figure 3.4(a)).

Let us now consider the case when the above assumption is not true. If c is a minimum or maximum and $f^{-1}(c)$ intersects the boundary of R_c then there exists another critical point within R_c, a contradiction.If c is a saddle point, then a configuration as shown in Figure 3.4(b) may arise. In this case, $f_{R_c} - f(c)$ will have more than four roots and the algorithm finds a smaller bounding box. □

3.9 Analysis

We now analyze the run time of the contour tree algorithm beginning with Algorithm 1. Let n_c denote the number of critical points in the patch. Sorting the critical points takes $O(n_c \log n_c)$ time. Assuming that the function is Morse, a constant number of MDPs are traced for each critical point resulting in $O(n_c)$ Find and Union calls. This takes $O(n_c \alpha(n_c))$ time. Let n_p denote the maximum number of segments in an MDP. Checking for intersection with previously computed MDPs takes $O(n_c^2 n_p^2)$ time and is the costliest step in the MDP tracing procedure. So, Algorithm 1 takes $O(n_c^2 n_p^2)$. Let n_t denote the number of triangles in the input. Stitching the local join trees takes results in $O(n_t n_c)$ Find and Union calls, which takes $O(n_t n_c \alpha(n_t n_c))$ time. Computing the global split tree also takes the same time. The global join and split tree can be merged in linear time on the total number critical points, which is $O(n_t n_c)$. For a degree d polynomial, $n_c \leq d-1$. So, the total time to compute the contour tree is dominated by the MDP tracing, which is $O(n_t d^2 n_p^2)$.

Chapter 4

Implementation Details

In this chapter, we discuss some of the implementation specific details like, the coding language used, libraries used and tools used to visualize and generate data.

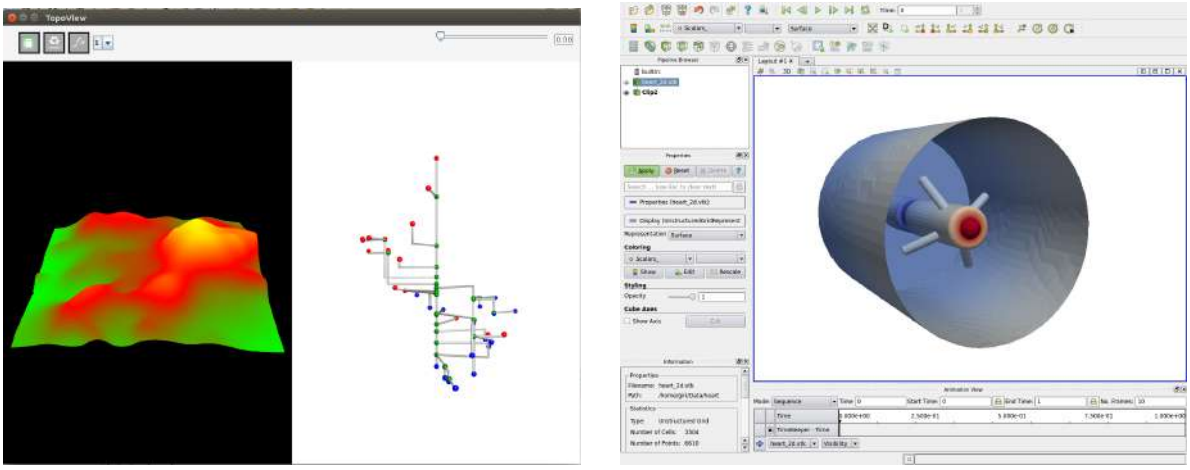
Computing local join/split trees for a patch, stitching the join/split trees and finally merging global join/split tree, all these procedures are implemented in C++. We use Python3.4 for generating scripts for extracting 2D surface out of 3D datasets, reading datasets and file format conversion. We list below, the libraries that we used along with their purpose. We analyzed and performed experiments on these libraries to check sanity and the suitability for our application.

- The open source C++ library Eigen [23] is used for finding the solution of a system of linear equations
- The GNU Scientific Library [22] is used for finding the roots of a univariate polynomial
- PHCpack [44] for finding roots of a polynomial system.

To visualize the final computed contour tree and to generate and visualize contour tree for piece-wise linear approximations, we use a in-house developed tool called as TopoView [2]. It shows the tree in a hierarchical fashion, called as branch decomposition view, from which the significance of a critical point can be made out. Figure 4.1a shows a screenshot of the tool. Left half is meant to render the data and right half is for showing contour tree.

A powerful data visualization and exploration tool Paraview [17] is used for generating the result images. It provides a vast range of filters, which aid in generating meaningful and visually appealing images. Some of the commonly used filters are listed below.

- **Contour** is used for showing contour curves.
- **Glyph** is used for highlighting critical points using spheres.



(a) (b)

Figure 4.1: (a) Topoview tool. (b) Paraview tool.

- **Tube** is used for highlighting contour curved using cylinders.
- **Clip** is used to clip away parts of the dataset to show culled regions.

A screen shot of the paraview tool showing a synthetic data is shown in Figure 4.1b. Clip filter is used to clip away the occluding surface.

Chapter 5

Experimental Results

In this chapter, results of computational experiments done on datasets with 2D piecewise-polynomials of degree two,three and five are presented. The contour tree computed by the algorithm contains all vertices of the input mesh as nodes. Hence, the output is the so-called *augmented contour tree*. In our implementation, the degree-2 nodes are pruned away. They may be retained if required.

5.1 Conductor

The thermal conductor dataset represents the temperature distribution at the surface of an electronic component computed by solving a multi-physics simulation using COMSOL [12]. The elements are polynomials of maximal degree 2. The six samples are located at the triangle vertices and at the mid-point of the edges, as shown in Figure 3.1b. For this dataset, the geometry of the elements is also modeled as a quadratic polynomial that accurately fits the curved surface. Figure 5.2a-5.2c shows the temperature field, its critical points together with a set of contours, the contour tree computed using the proposed algorithm. A topological feature is represented by a min-saddle or max-saddle arc / path in the contour tree. The size of a topological feature is measured using the notion of topological persistence [20], which is equal to the difference in function value at the pair of critical points.

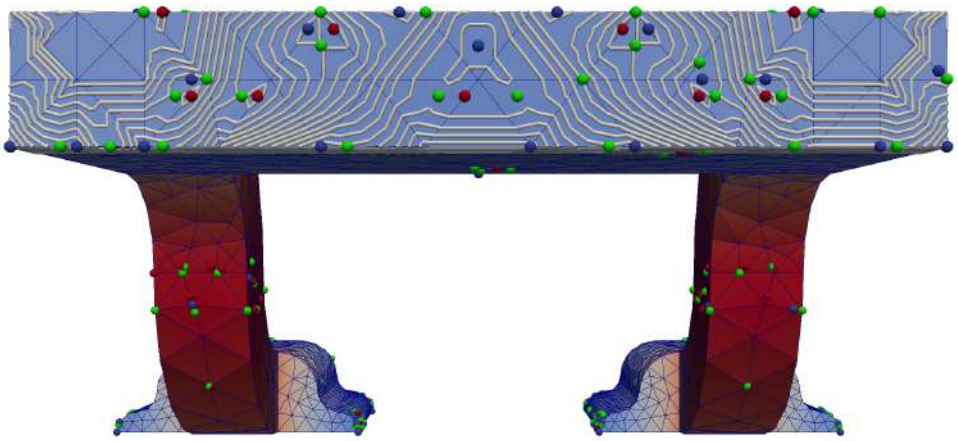


Figure 5.1: Top of the conductor contains multiple degeneracies. The algorithm identifies all the critical points and handles the degenerate patches gracefully.

The contour tree contains 530 nodes, several of them corresponding to small-sized topological features. Given a persistence threshold the contour tree may be simplified by iteratively removing arcs incident on leaf nodes [8]. Figure 5.2d shows the contour tree simplified using a persistence threshold of 0.1%. The temperature field contains multiple degeneracies, particularly on the top plate as shown in Figure 5.1. Our algorithm handles all degenerate patches gracefully and computes the contour tree. The simplified tree shows the two significant maxima located at the center of the two legs surrounded by a few low persistence maxima. Note that the tree contains two similar subtrees as expected. The subtrees correspond to the symmetric legs of the conductor.

5.2 Heater

The heater dataset is a sum of Gaussian function sampled on the surface of a heater. Each triangle in the mesh representing the heater surface contains ten sample points. Three samples are located at the vertices, two within each edge subdividing it into equal sized segments and one at the barycenter. A polynomial of maximal degree 3 interpolates the ten samples. Figure 5.3a-5.3d shows the cubic function, its critical points together with a set of contours, the contour tree computed using our algorithm, and the contour tree simplified using a persistence threshold of 0.5%. Gradient computation and root finding are costlier for the cubic interpolant. However, the use of univariate polynomials to trace monotone paths and the use of bounding boxes helps resolve several numerical issues. This data again contains multiple degenerate patches. All degeneracies are gracefully handled.

5.3 Sphere

Cubic-sphere dataset is generated by defining a sum of sine field on a sphere triangulation, as shown in Figure 5.4a. It is a piecewise-cubic dataset with 1056 triangles. Figure 5.4b shows the contours and critical points for Figure 5.4a. As the function is symmetric and almost all of the critical points are of high persistence, contour tree, having 136 nodes,in Figure 5.4c has equal number of maxima and minima. On 0.5 percent simplification, the contour tree reduces to a tree with 110 nodes. Most of the nodes are retained because of their significant persistence, Figure 5.4d.

5.4 Two triangle

A toy dataset having a piecewise polynomial function of order 5 defined on two adjacent triangles sharing an edge is shown in Fig 5.5a. For each triangle, there are total 21 samples. Three on the triangle vertices, four samples each on the triangle edges and six samples inside the triangle. This dataset has been generated just to showcase the scalability of our algorithm across degree of interpolation. To compute the contour tree, we make 2 calls to local tree computation procedure and a single stitch procedure. Contour lines and the critical points of the dataset are shown in the image Fig 5.5b, observe that, there are 11 critical points inside the triangle boundary and are detected with high accuracy by PHCpack routines, contour line behaviour confirms this. The computed contour tree for this dataset has 37 critical points and all the critical points are of high persistence. Fig 5.5c shows the computed contour tree mapped on to the dataset and Fig 5.5d shows the branch decomposed contour tree.

5.5 Case-studies

We present three additional case studies in Figure 5.6, Figure 5.7 and Figure 5.8. where we compare the contour tree computed for the higher order interpolants with the contour tree of PL-approximations of the datasets. The PL approximations are produced by applying a uniform refinement on the input triangulation and linearly interpolating the polynomials on the refined triangles.

5.5.1 Conductor

Figure 5.6 compares the contour tree of a piecewise quadratic function with the contour tree of piecewise linear (PL) approximations. The PL approximations are computed by applying a uniform refinement on the input triangulation and by linearly interpolating the polynomials on the refined triangles. There are large differences between the contours of the original data and those of the PL approximations even after a high level of refinement. A close-up view shows how the topology of the contours, and thus the contour trees, differ. The visual comparison indicates that a higher order interpolant can capture all topological features even if the surface is discretized using fewer number of triangles. The

PL approximation is not able to accurately capture the level set topology even with a 15-fold increase in number of triangles. For example, the number of detected critical points in PL approximation with 4298 triangles, shown in Figure 5.6b are less compared to the piecewise-quadratic case, shown in Figure 5.6e. Although number of critical points in PL approximation with 17192 triangles, shown in Figure 5.6c are at least as much as the number of critical points in piecewise-quadratic case, notice the error in critical point position as compared to the true positions. PL approximation with 68768, shown in Figure 5.6d, captures all of the critical points as that of its piecewise-quadratic counterpart. Some new low persistent minima are reported due to the degeneracy caused by smaller triangles. Also notice that the critical point positions in Figure 5.6d are still different from the true positions. Change in the topology of the contours, and thus also the contour trees shown in Figure 5.6g-Figure 5.6j differ between the PL approximation and the original polynomial datasets. Notice that the section of the contour tree shown in Figure 5.6(i) contains more number of maxima than Figure 5.6(j). However, these additional maxima are introduced due to flat regions and have zero persistence.

5.5.2 Heater

Figure 5.7 compares the contour tree of the piecewise cubic function defined on the heater dataset with the contour tree of its PL approximations. The contours of the PL approximation is significantly different from that of the piecewise cubic function defined on an equal number of triangles. A 10-fold increase in number of triangles seems to be necessary to obtain similar contours. Notice the large differences in critical point position and number in PL approximation with 12438 triangles, shown in Figure 5.7b as compared to the piecewise-cubic counterpart in Figure 5.7d. PL approximation with 111942 triangles captures the critical point position better but misses out on some of the close-by critical points, in Figure 5.7c, a close-by maxima is missed.

5.5.3 Sphere

A Similar case study has been performed on cubic-sphere dataset as well in Figure 5.8. Figures 5.8a,5.8d and 5.8g show the function, contour lines and critical points, and contour tree for a PL approximation with 1056 triangles. Note the significant differences in the position and number of critical points. A PL approximation with 9504 triangles,Figures 5.8b,5.8e and 5.8h gives matching results as that of its piecewise-cubic counter part with 1056 triangles Figures 5.8c,5.8f and 5.8i. They have same number of critical points but there are differences in the positions of these critical points.

5.6 Running Time

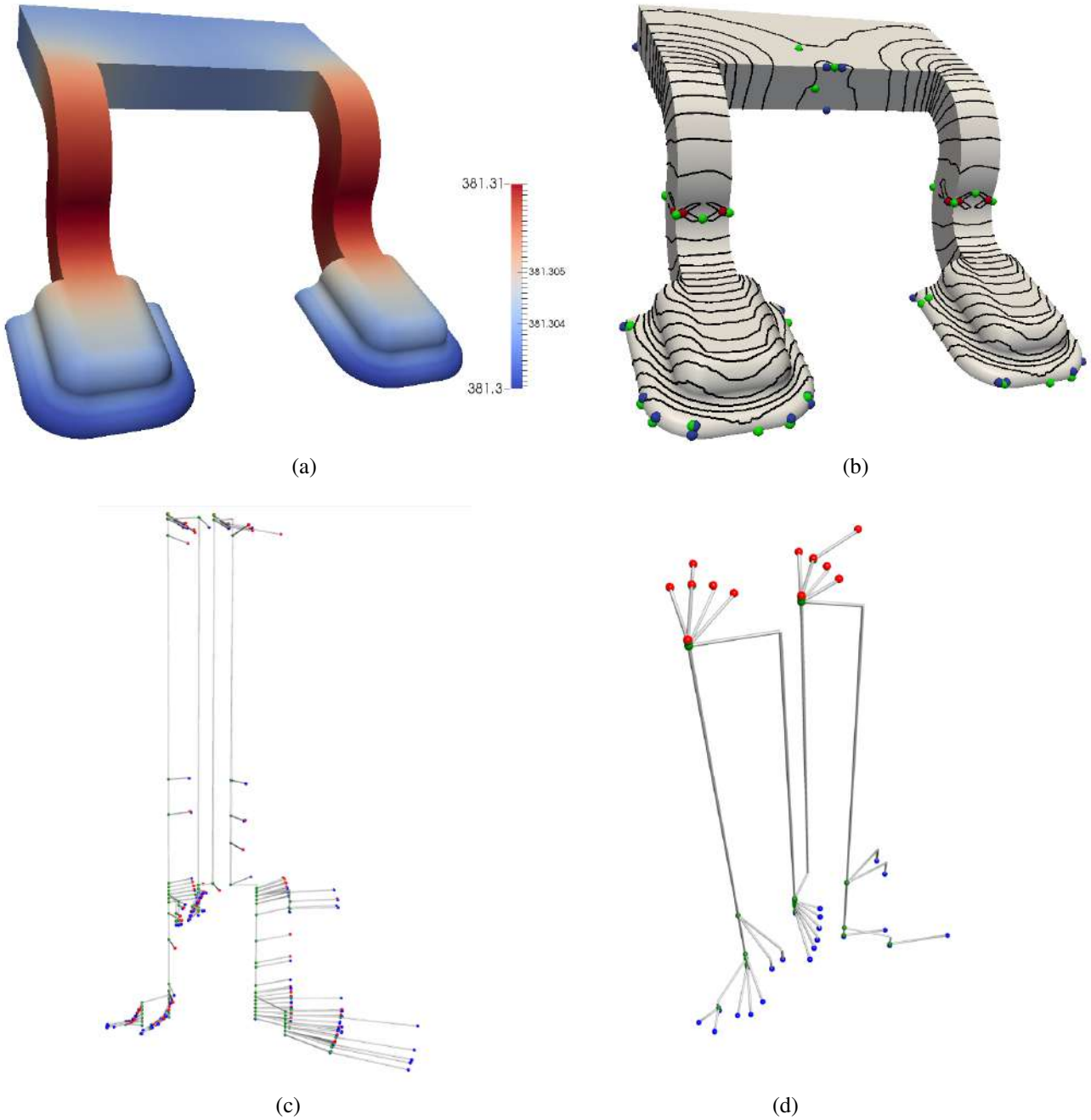
The focus of the current implementation was on correctness and validation of the proposed algorithm and not on efficiency. We present in Table 5.1, the running time observed with the current sequential

implementation. The datasets were run on a HP workstation with Intel Xeon 2 X 2 GHz processor with 4 GB of RAM. Algorithmic and code optimizations may result in significant reduction in running times. For example, several steps of the algorithm can be parallelized, structure of the code can be changes to avoid writing to files. We plan to do this in future.

Table 5.1: Running Time

Dataset Name	**Degree of Interpolation**	**No. of Vertices**	**No. of Triangles**	**Running Time(s)**	**No. of nodes in CT (Before Simplification)**	**No. of nodes in CT (After Simplification)**	**Simplification %**
Conductor	2	8598	4298	962.2	530	67	0.1
Gauss simp40	2	1600	722	56.7	28	16	0.5
Heater	3	78427	12438	10678.1	268	178	0.5
Sphere	3	4754	1056	206.9	136	110	0.5
2-Triangle	5	36	2	0.193	38	38	0.5

As evident by the Table 5.1, runtime is directly proportional to the size, degree of interpolation and degeneracy in the dataset.



(a) (b)

(c) (d)

Figure 5.2: Degree of Interpolation : 2. (a) Temperature distribution as a quadratic function on the surface of a thermal conductor. (b) Critical points and contour lines. (c) Contour tree without simplification consisting of 530 nodes. (d) Contour tree with 67 nodes obtained after simplifying using a 0.1 percent persistence threshold.

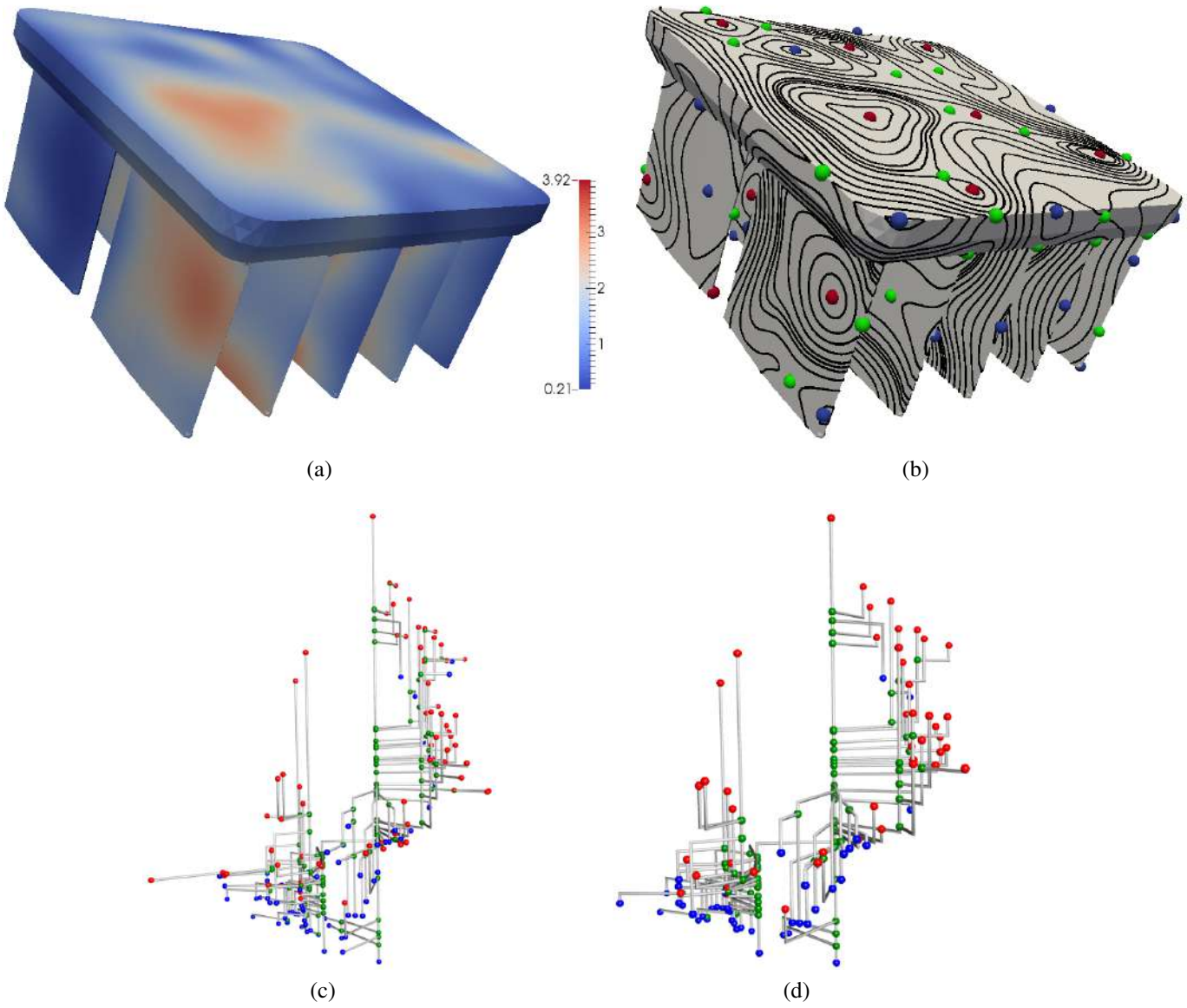


Figure 5.3: Degree of Interpolation : 3. (a) A cubic function defined on a heater geometry. (b) Critical points and contour lines. (c) Contour tree without any persistence simplification having 268 critical points. (d) Contour tree with 178 nodes obtained after simplifying using a 0.5 percent persistence threshold.

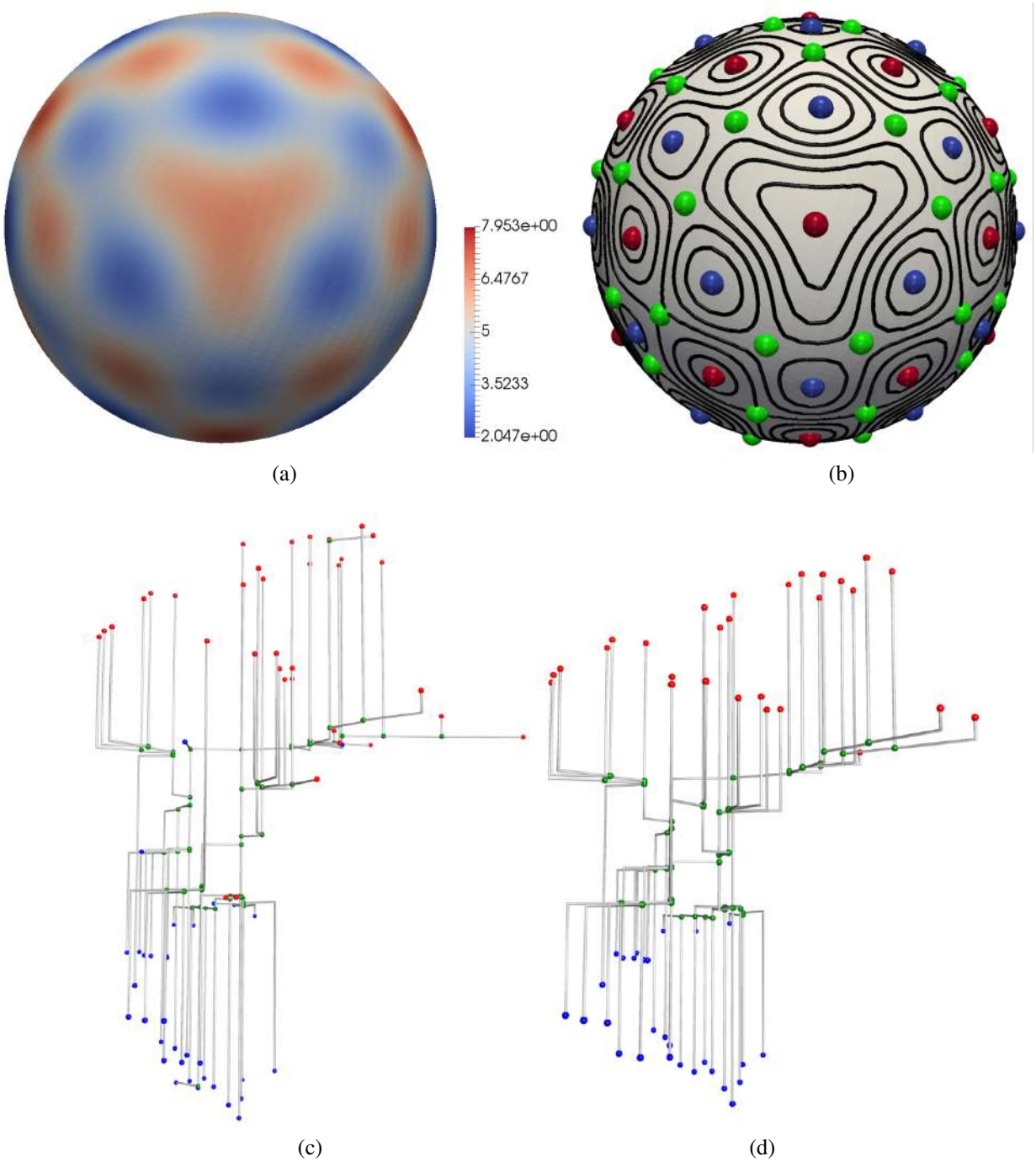
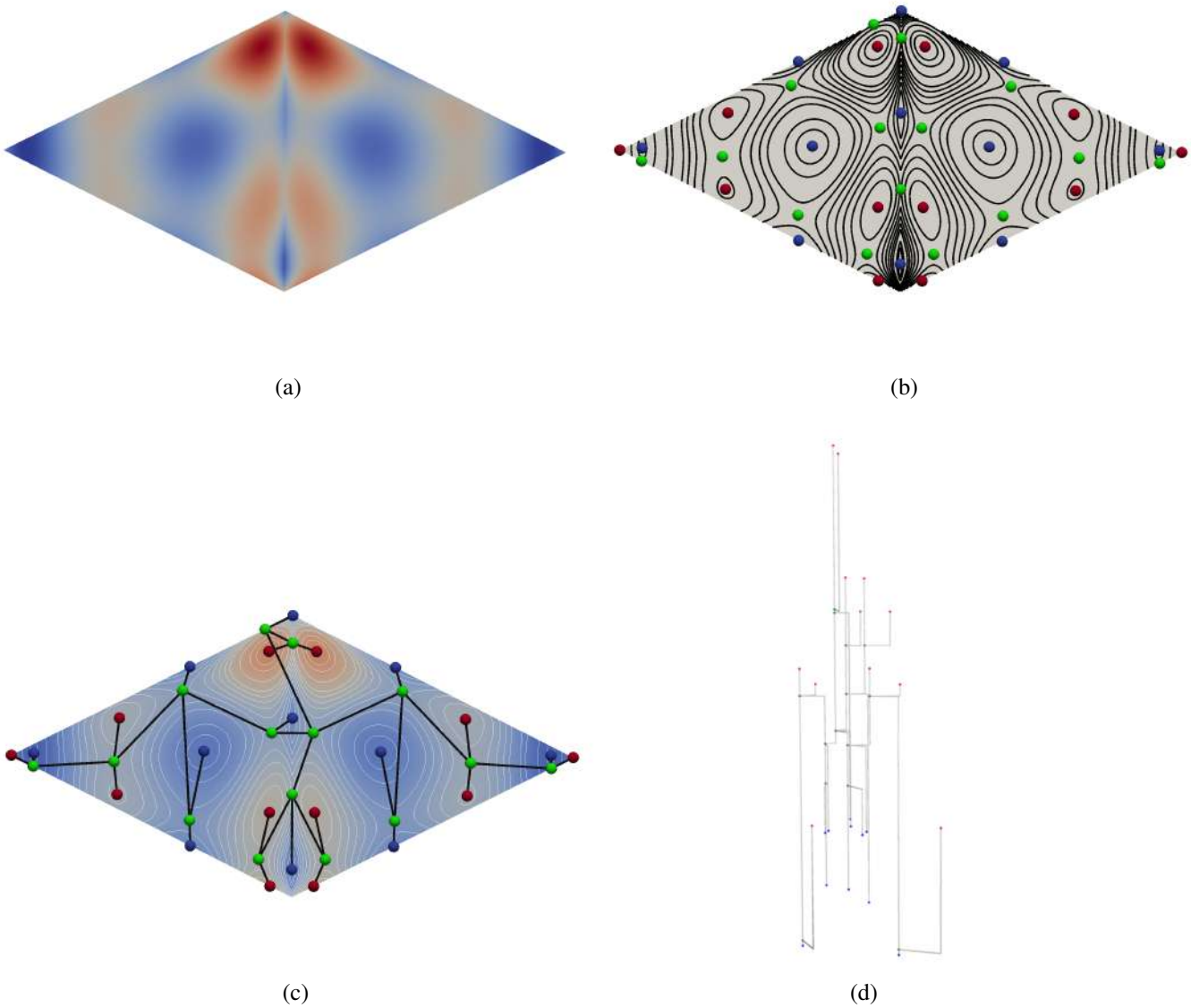


Figure 5.4: Degree of Interpolation : 3. (a) A sum of sine function defined on a sphere (b) Contour lines and critical points of 5.4a. (c) Contour tree shown on the dataset without any simplification. It has 136 nodes. (d) Contour tree after 0.5 percent simplification. It has 110 nodes



(a) (b)

(c) (d)

Figure 5.5: Degree of Interpolation : 5. (a) Toy dataset showing a piecewise-polynomial function of order 5 defined on two triangles sharing an edge. (b) Contour lines and critical points of 5.5a. (c) Contour tree shown on the dataset. (d) Contour tree shown in branch decomposition form, it has 37 nodes

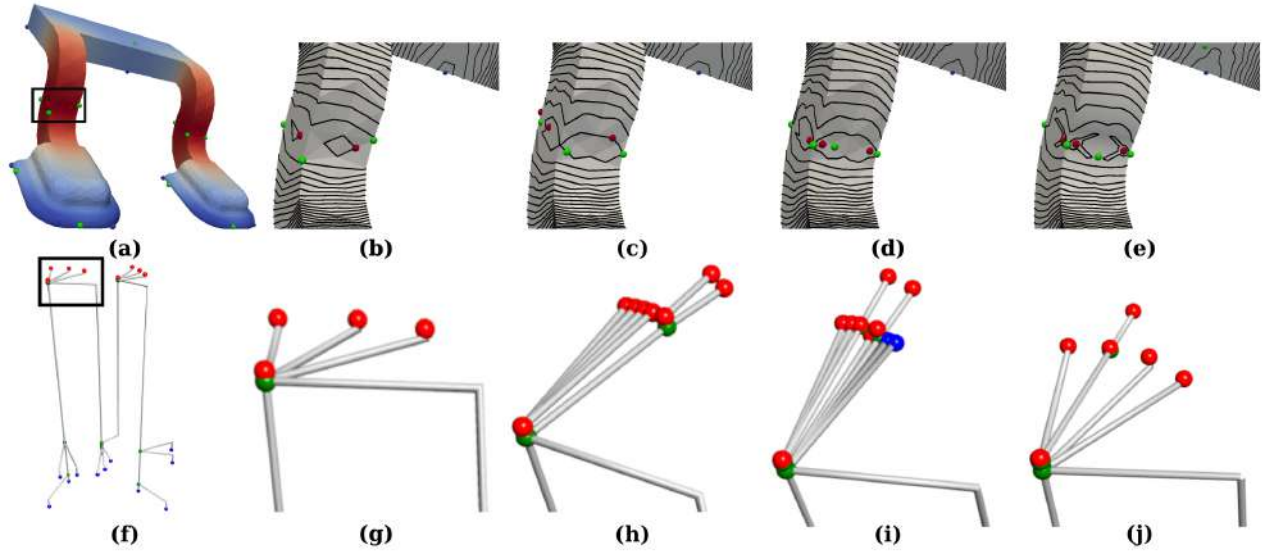


Figure 5.6: (a) Piecewise linear (PL) approximation of the temperature scalar field defined in the thermal conductor dataset shown in Fig. 5.2. (f) Contour tree for the PL approximation. (b-d) Region of thermal conductor showing critical points and contours for successive PL subdivisions of conductor containing 4298, 17192, and 68768 triangles, respectively. (g-i) Corresponding nodes and arcs in the contour tree. (e) Region of conductor showing critical points and contour lines for piecewise quadratic function with 4298 triangles. (j) Nodes and arcs from contour tree computed using proposed method for the piecewise quadratic function

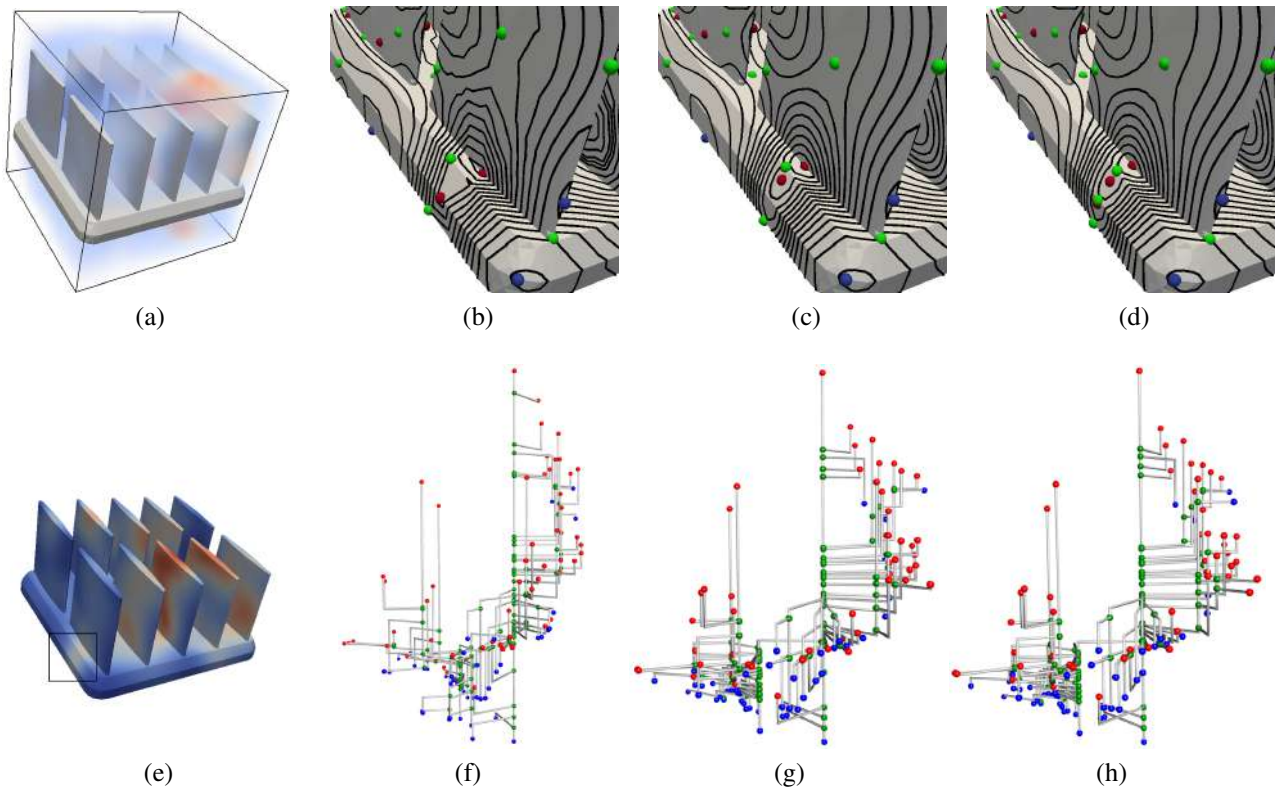
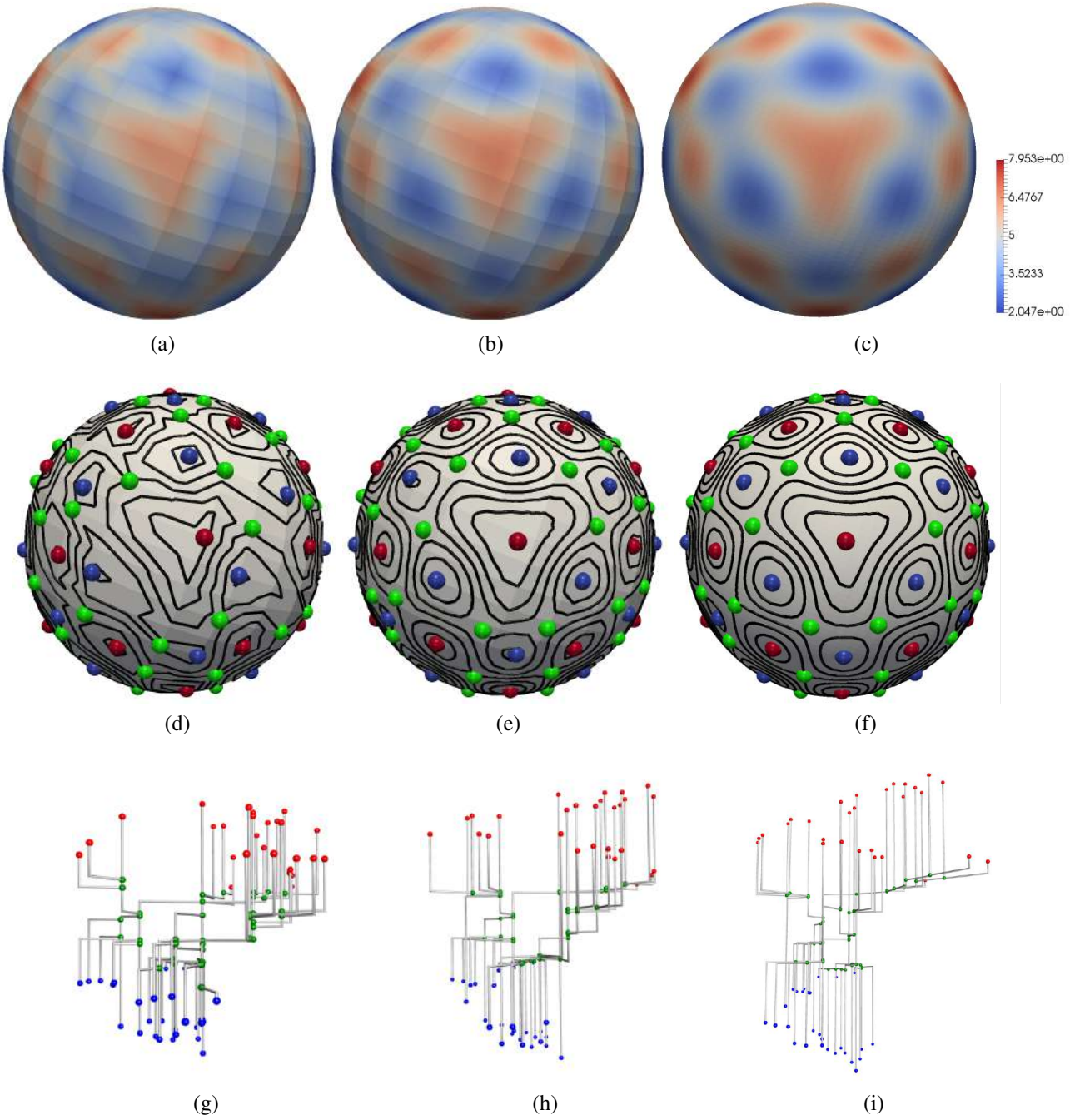


Figure 5.7: (a) Geometry of Heather dataset along with a 3d Gaussian field. (e) Heather dataset showing the mapped 3d Gaussian field on to its surface. (b) Part of heather dataset, showing critical points computed using piecewise linear approximation having 12438 triangles.(f) Corresponding contour tree containing 210 critical points. (c) Part of heather, showing critical points computed using a piecewise linear approximation having 111942 triangles. (g) Corresponding contour tree with 208 critical points. (d) Part of heather showing critical points computed using a piecewise cubic function with 12438 triangles. (h) Corresponding contour tree containing 212 critical points.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.8: (a) A sum of sine function defined on a piecewise-linear sphere with 1056 triangles. (d) Contour lines and critical points of 5.8a. (g) Contour tree for 5.8a containing 105 nodes. (b) Same function defined on a piecewise-linear sphere with 9504 triangles. (e) Contour lines and critical points of 5.8b. (h) Contour tree for 5.8b containing 110 nodes. (c) Same function defined on a piecewise-cubic sphere with 1056 triangles. (f) Contour lines and critical points of 5.8c containing 110 nodes. (i) Contour tree for 5.8c.

Chapter 6

Contour tree computation for piecewise-quadratic functions using case analysis method

In this chapter, we present another method of computing contour trees for a triangle, on which a bivariate quadratic polynomial function is defined. This method relies on the fact that the input function is a Morse function. An exhaustive combinatorial analysis helps identify the appropriate contour tree for each patch. This method does not extend easily to higher order interpolants and is superseded by the MDP based method presented in the earlier chapters.

In the section 6.1 we describe the method for computing the contour tree in detail, and in section 6.2 we present some of the results obtained by an efficient implementation.

6.1 Method

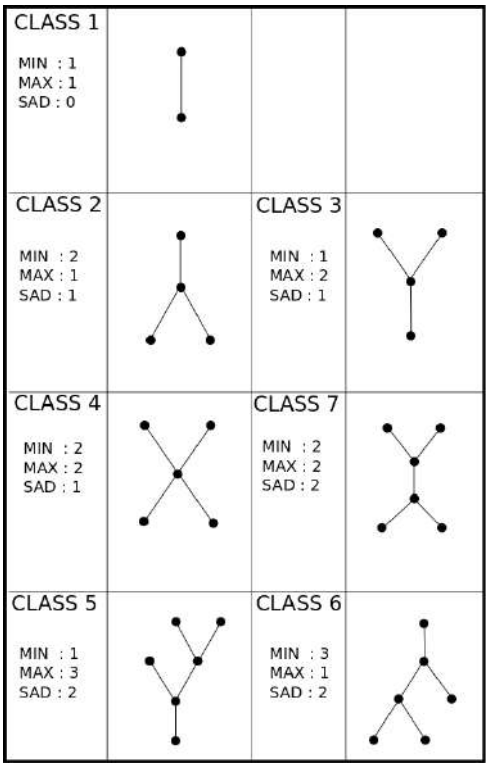
Our method has three major steps, they are described in detail in below subsections.

6.1.1 Enumerating all possible classes of contour trees for a patch.

Since we are assuming that the function defined on the patch is a Morse function, in the contour tree we can only have degree 1 (maxima and minima) and degree 3 (saddle) nodes. We need an upper bound on the number of nodes in the contour tree. We do that in the following lemma.

Lemma 6.1 *There can be at most 7 critical points for a patch.*

Proof: Function defined on the patch is bi-quadratic, hence it can have at most 1 surface critical point. We will address this surface critical point as face criticality. Bi-quadratic function when restricted to a line segment, will be a quadratic function in 1 variable and it could have at most 1 critical



(a)

Figure 6.1: Possible contour trees for a piecewise-quadratic patch.

point. Since we have 3 edges, there could be at most 3 such critical points. We refer such critical points as line criticality. Lastly, restricting the function to the triangle may cause vertices of the triangle to behave as critical points. Hence in total we could have 7 critical points at most for the patch. Since only critical points appear in contour tree, the contour tree for the patch can have 7 vertices (1FC+3LC+3V). With these constraints, we classify all possible types of trees in 7 classes as shown in Figure 6.1. Notice that CLASS 4 violates 'only degree 3 saddles allowed' condition, but this is an exception since we are considering contours local to a patch. A degree 4 saddle would in fact be a degree 3 saddle of the global function. This type of contour tree is obtained in the presence of a face saddle. □

6.1.2 Enumerating all possible cases of face and line criticality [F,L]

A face criticality can either be present or absent. If present, it can be a maximum , a minimum or a saddle. Same way, the number of line criticality can be 0,1,2 or 3. In total there are around 40 different combinatorial cases to be considered as shown in Figure 6.2.

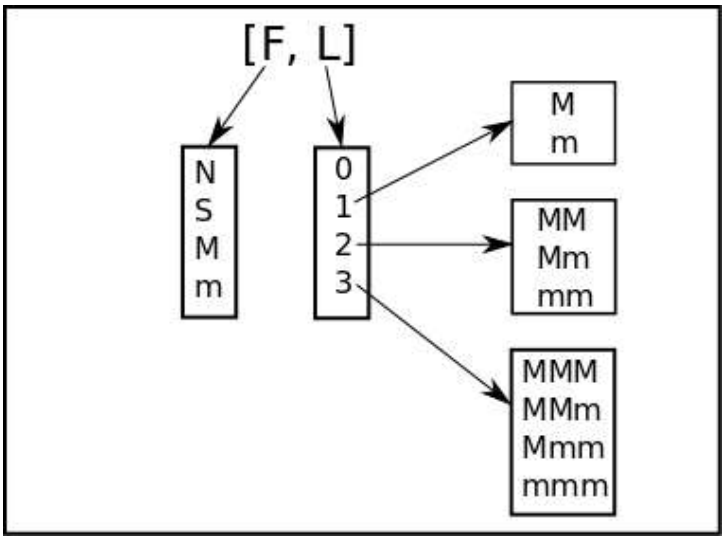


Figure 6.2: Enumeration of face and line criticality

6.1.3 Analyzing each case to assigning contour tree class

Each case is analyzed and mapped to corresponding contour tree class as shown in Figure 6.3. Stages 2 and 5 in Figure 6.3 make use of a set of rules. A FC appears as same type of criticality in CT, but same doesn't hold for LC. A maximum LC can appear as a maximum, saddle or regular vertex when evolution of contours is considered for the whole patch, we address this as global behaviour. Stage 3 deals with this by enumerating all possible choices for types of LCs. Once we decide on global behaviour of LC, in stage 4 we compute number of triangle vertices which are regular and make use of this fact along with set of rules to decide the correct CT class for the case. A decision tree showing the paths from each case to their mapped contour tree classes is designed and each case is processed accordingly. Figures 6.4 and 6.5 show the complete decision tree. CT 1 means contour tree class 1, CT 2 means contour tree class 2 and so on. NP means not possible. The representation [M,Mmm] means the FC is a maximum, one LC is a maximum, and two LCs are minima.

Following are the rules, which are used for mapping a case to a contour tree class.

1. In the contour tree, sum of vertices should be even. $N_{max} + N_{min} = N_{saddle} + 2$. This constraint is a direct consequence of the Morse function. Proved in Lemma 6.2.

2. If a vertex of the triangular patch is a saddle, then the patch has zero face critical points and one line critical point.

3. A line criticality can't appear as opposite type of criticality in contour tree.

4. Maximum number of nodes in a CT for a case [X , Y] is $X + Y + 3$.

5. Consider triangular patch in case [0, 3], with six critical points. The only possible contour tree for the patch is the one where all three vertices are minima (maxima), two LCs are saddle and one LC is a maximum (minimum).

6. There can be at most one LC which can behave as a regular point globally.

7. If all LCs are maxima (minima) then all triangle vertices are minima (maxima).

8. In cases [0, Mmm] and [0, MMm], among the LC type which dominates, one LC behaves as a saddle globally.

9. When a FC is a maxima (minima) then no LC or vertex could be maxima (minima) in the contour tree.

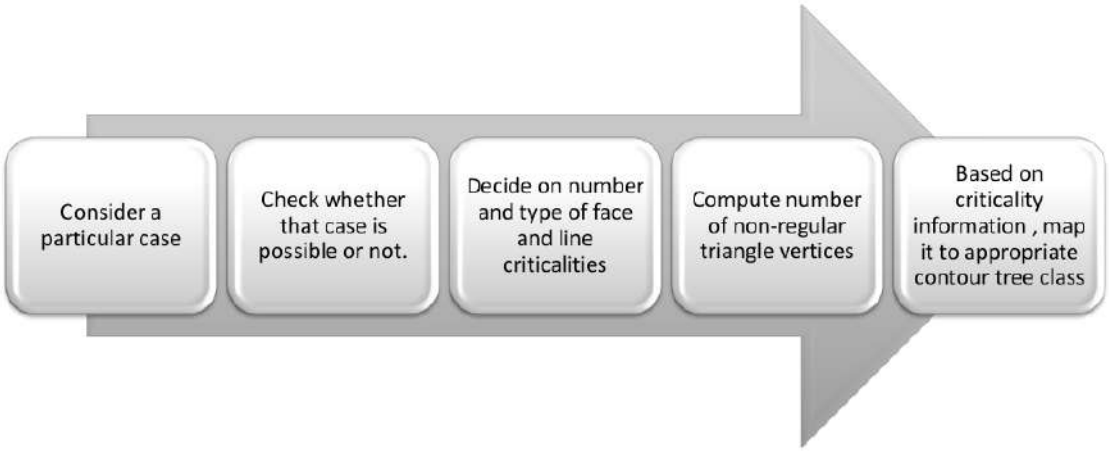


Figure 6.3: Steps to decide contour tree

Lemma 6.2 *In a contour tree of a Morse function f, $N_{max} + N_{min} = N_{saddle} + 2$*

Proof: Since contour tree is a graph, sum of degrees of the vertices is equal to twice the number of edges.

Sum of degrees of vertices = 2 * (Number of edges)
deg(Maxima) + deg(Minima) + deg(Saddle) = 2 * (Number of edges)
Since maxima and minima are of degree 1 and saddle is of degree 3,
1*(N_max) + 1*(N_min) + 3*(N_saddle) = 2 * (Number of edges)
Since contour tree is a tree, Number of edges is one less than sum of all vertices.
1*(N_max) + 1*(N_min) + 3*(N_saddle) = 2 * ($N_max + N_min + N_saddle - 1$)
Hence, $N_{max} + N_{min} = N_{saddle} + 2$ □

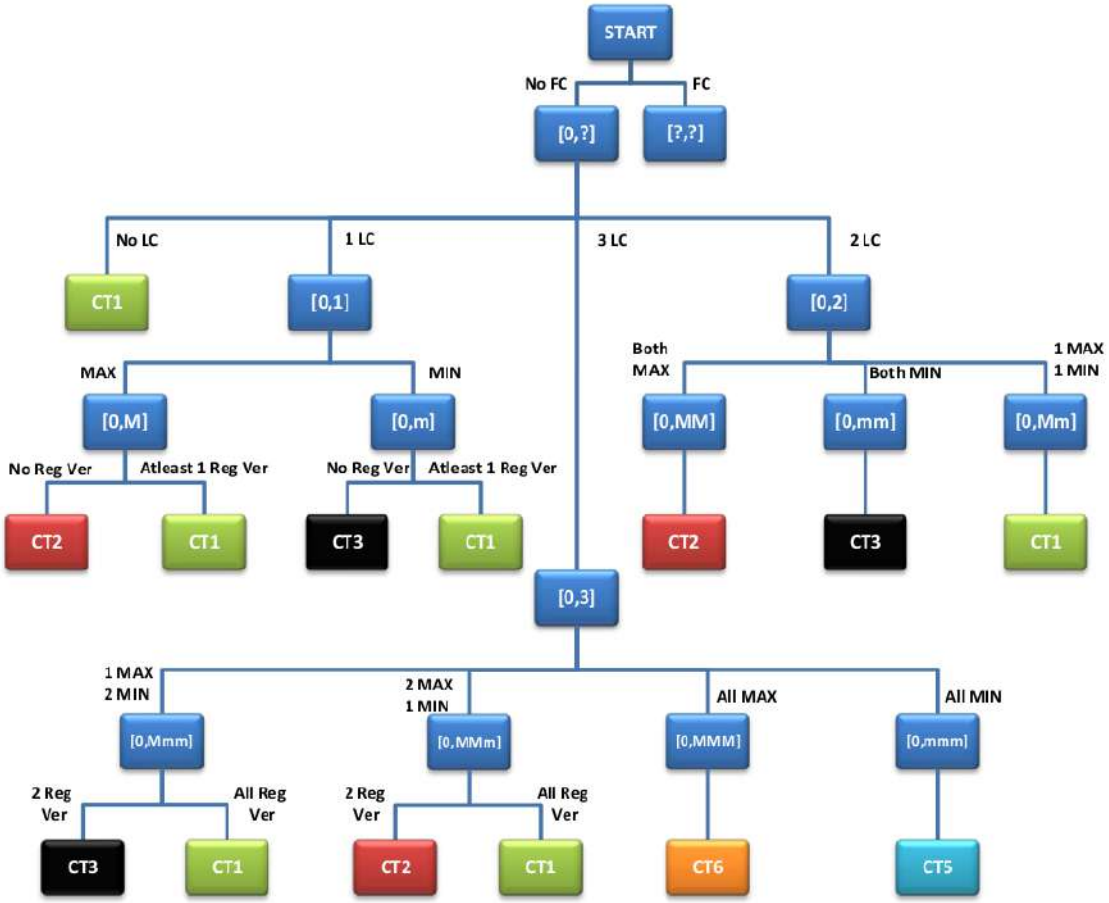
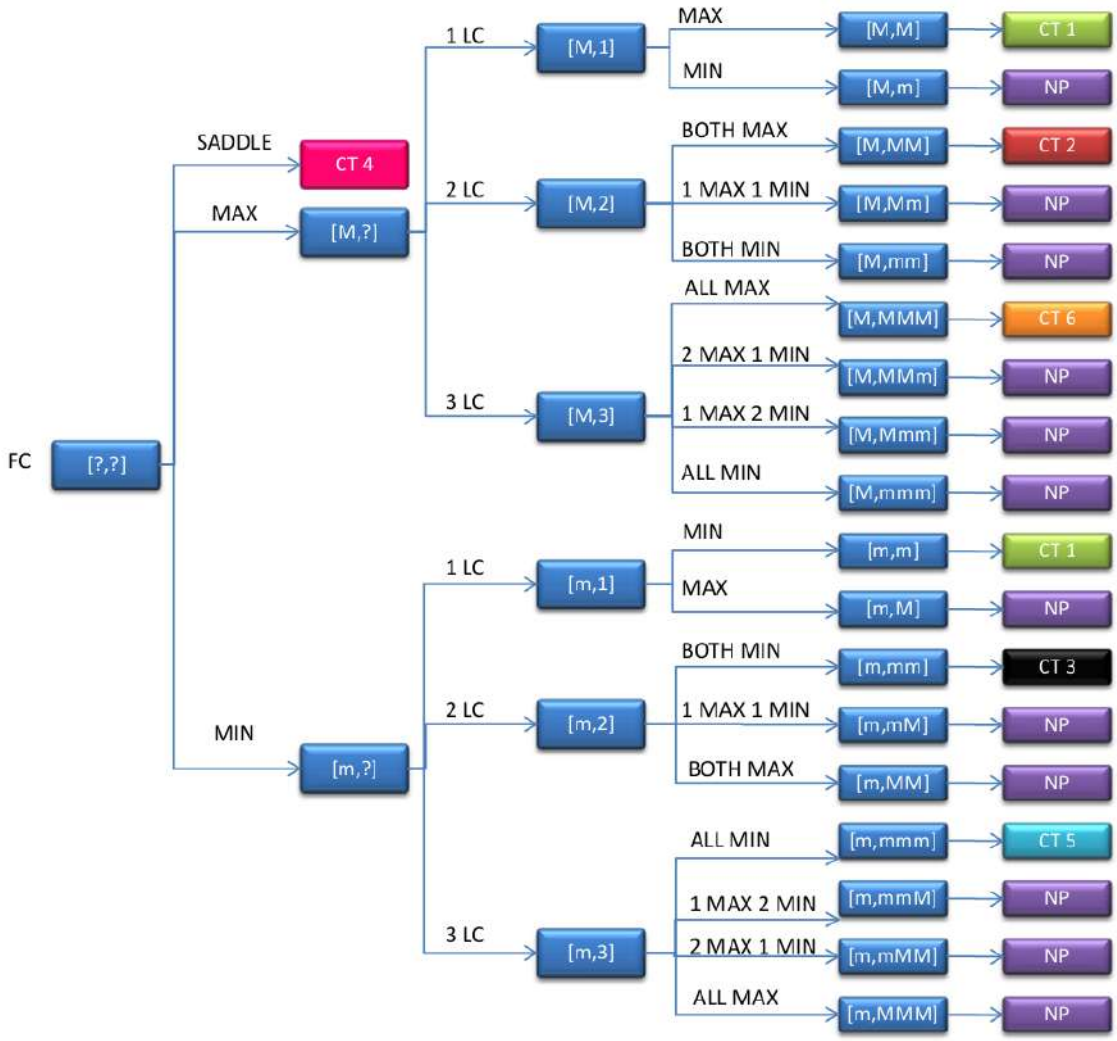


Figure 6.4: Decision tree branch 1 : No face criticality

Lemma 6.3 *Case checking is exhaustive.*

Proof: Above approach is exhaustive because, all possible combinatorial choices of cases are considered and analyzed. [F,0] cases where we have a face criticality and no line criticality are neglected because that combination is not possible [16]. Some of the cases are mentioned as NP (not possible) in decision tree, following proofs address them. In cases which are said to be not possible and the ones where contours are elliptic, if at least one line restriction of f contains a LC point, then all LC points are of same type (if they exist). This explains why cases mentioned below were labeled NP.

- [M , Mm]
- [M , MMm]
- [M , Mmm]



(a)

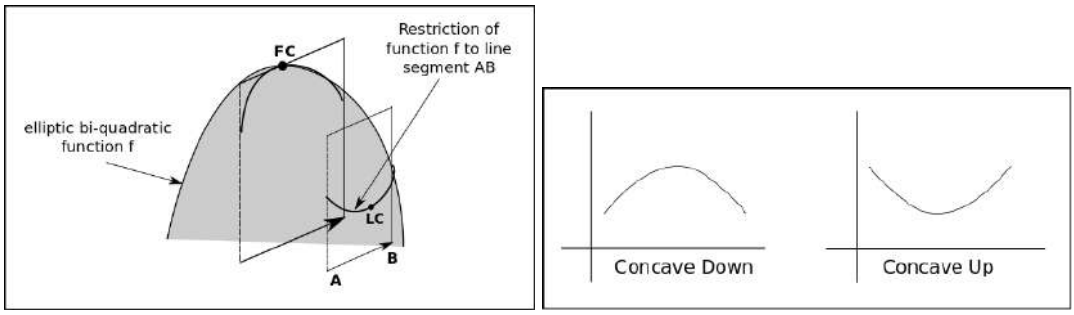
Figure 6.5: Decision tree branch 2 : With face criticality

- [m , mM]
- [m , mmM]
- [m , mMM]

□

Lemma 6.4 *If a line restriction of elliptic bivariate quadratic function f contains a criticality, then that LC should be of same type as that of the FC.*

Proof: Let say FC is a maximum and lets assume line criticality D between triangle vertices A and B is a minimum. Line restriction of f to a line segment AB can be viewed as a plane starting at AB



(a) (b)

Figure 6.6: (a) Proof image. (b) concave down and concave up curves

and cutting f as shown in the Figure 6.6a. Since we have assumed that the LC is a minimum, line restriction of AB is a concave up quadratic curve. Now move the line restriction plane towards the FC until it passes through it. By doing this we are considering line restrictions restricted to different line segments. Among all such line restrictions consider the one which passes through the FC. This line restriction is a concave down quadratic curve (since FC is a maximum). We notice that as we sweep the restriction plane, curve changes from being concave up to concave down, which only happens at a saddle. This is a contradiction to the fact that f is an elliptic bivariate quadratic function which can only have 1 face critical point. In similar way , it can also be proved when the FC is a minimum and LC is a maximum. Hence the lemma is true.

Lemma 2 addresses which cases listed below are NP.

- [M , m]
- [M , mm]
- [M , mmm]
- [m , M]
- [m , MM]
- [m , MMM]

We have considered all combinatorially possible cases and explained why some of the cases are not possible. Hence case checking is exhaustive. □

Lemma 6.5 *Contour tree is correctly computed for each case.*

Proof: Once the patch is classified as a particular case, All possible global behaviour of the line critical points are considered. Patch vertices can behave as maximum/minimum/regular vertex or saddle (this is decided by doing some comparisons). Vertex behaving as a saddle is handled separately (see appendix for more information). Once we decide on number and type of all criticality, all CT classes which match the obtained specification are considered and refined based on few rules to get the final CT for the patch. Note that obvious NP cases are eliminated and some of the cases mentioned valid may not even be possible because of the behaviour of the bi-quadratic function, we are being cautious and analyzing which CT class to assign in case the control ever visited that leaf. □

6.2 Results

We present some of the results, showing a single triangle with different piecewise-quadratic polynomials defined on them and the corresponding contour tree in Figure 6.7 and Figure 6.8. Images are organized into 6 contour tree classes, for each class two scenarios are presented.

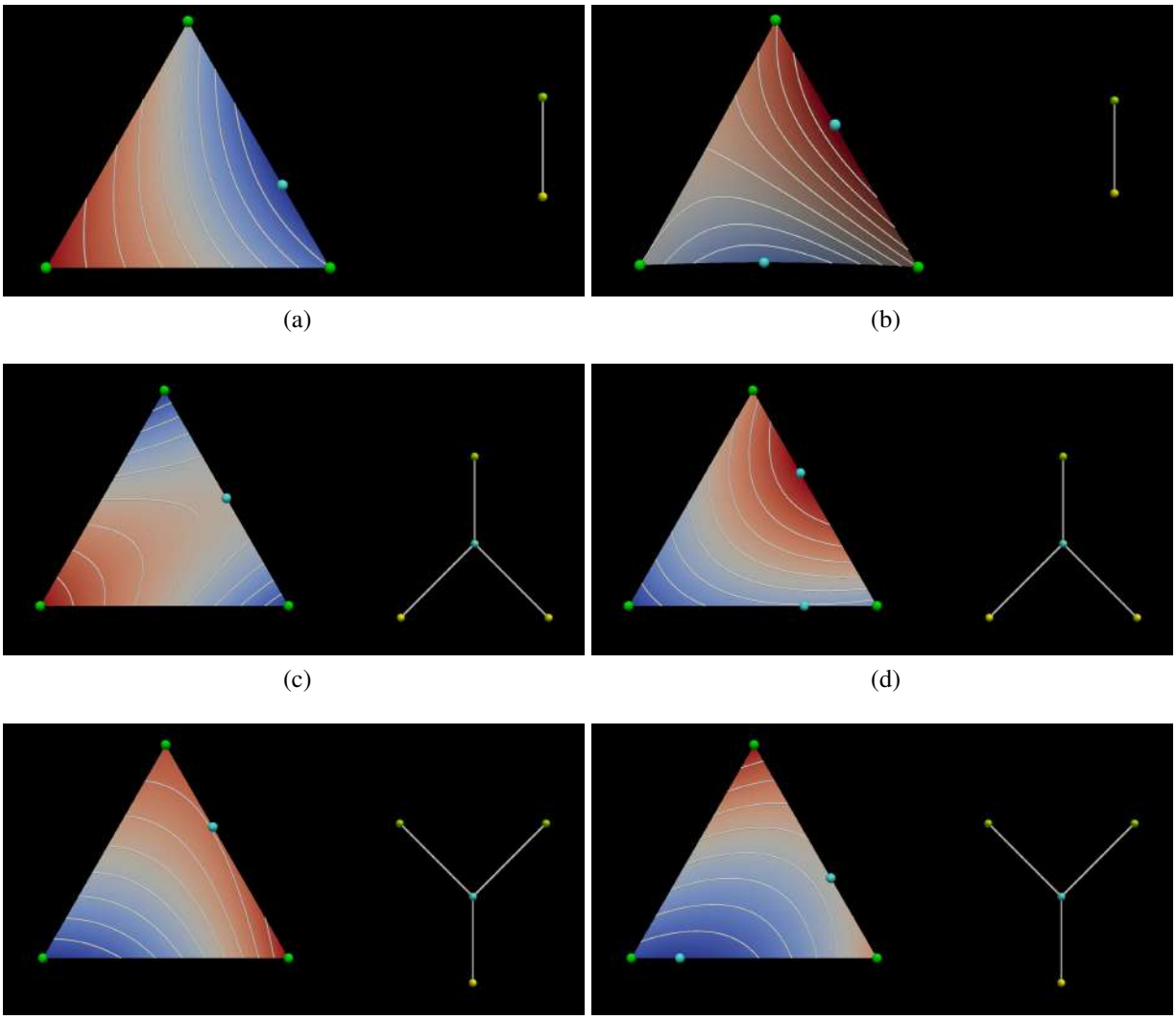


Figure 6.7: (a) Class 1 [0,m] , (b) Class 1 [0,Mm] , (c) Class 2 [0,M] , (d) Class 2 [0,MM] , (e) Class 3 [0,m] , (f) Class 3 [0,mm]

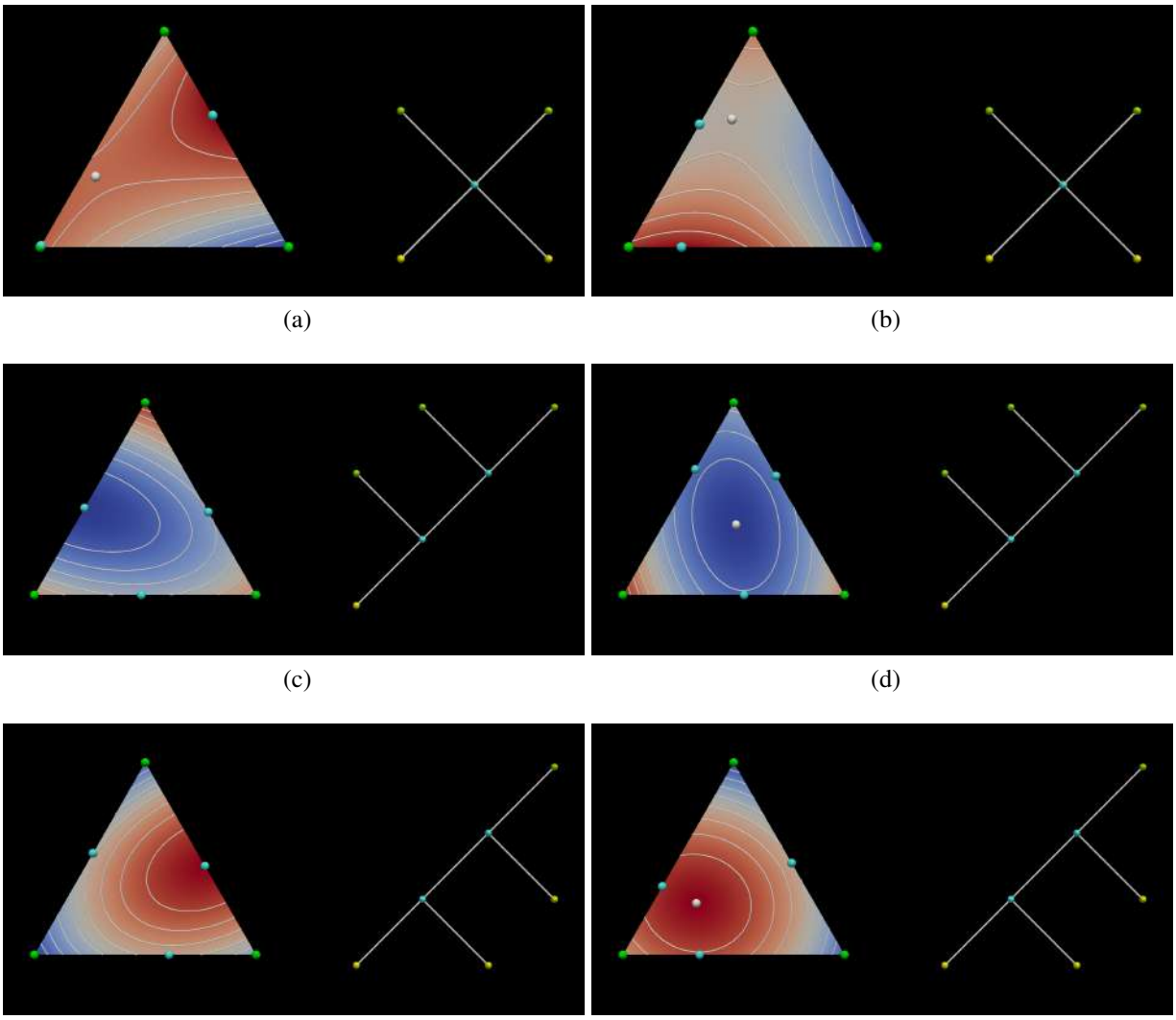


Figure 6.8: (a) Class 4 [S,M] , (b) Class 4 [S,Mm] , (c) Class 5 [0,mmm] , (d) Class 5 [m,mmm] , (e) Class 6 [0,MMM] , (f) Class 6 [M,MMM]

Chapter 7

Conclusions

We have described a novel algorithm for computing the contour tree of a 2D piecewise polynomial scalar field that is defined over a triangle mesh. The algorithm employs efficient numerical computations to trace monotone paths within each triangle. All other steps are combinatorial in nature. Experimental results show how the algorithm can efficiently capture topological features that are not easily identified using a linear approximation. With the increasing use of higher-order element data in simulations, it is essential that the analysis and visualization techniques are also directly applied on the higher-order elements. This is particularly true for topology-based visualization techniques, which aim to capture and represent key features in the data.

Bibliography

[1] Aditya Acharya and Vijay Natarajan. A parallel and memory efficient algorithm for constructing the contour tree. In 2015 IEEE Pacific Visualization Symposium (PacificVis), pages 271–278, 2015. 2, 3, 13, 15, 16

[2] Utkarsh Ayachit. The paraview guide: a parallel visualization application. Kitware, Inc., 2015. 20

[3] I. Babuška and B. Q. Guo. The h, p and h-p version of the finite element method: Basis theory and applications. Adv. Eng. Softw., 15(3-4):159–174, November 1992. 1

[4] Peer-Timo Bremer, Gunther Weber, Julien Tierny, Valerio Pascucci, Marc Day, and John Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. IEEE Transactions on Visualization and Computer Graphics, 17(9):1307–1324, September 2011. ISSN 1077-2626. 2

[5] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. Nektar++: An open-source spectral/element framework. Computer Physics Communications, 192:205 – 219, 2015. 1

[6] Hamish Carr and Jack Snoeyink. Representing interpolant topology for contour tree computation. In Hans-Christian Hege, Konrad Polthier, and Gerik Scheuermann, editors, Topology-Based Methods in Visualization II, pages 59–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 3

[7] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. Computational Geometry, 24(2):75 – 94, 2003. 2, 3, 11, 15, 16

[8] Hamish Carr, Jack Snoeyink, and Michiel van de Panne. Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. Comput. Geom. Theory Appl., 43(1):42–58, January 2010. ISSN 0925-7721. 2, 23

[9] Hamish A Carr, Gunther H Weber, Christopher M Sewell, and James P Ahrens. Parallel peak pruning for scalable smp contour tree computation. In 6th IEEE Symposium on Large Data Analysis and Visualization, 2016. 2

[10] Amit Chattopadhyay, Gert Vegter, and Chee K. Yap. Certified computation of planar morse-smale complexes. Journal of Symbolic Computation, 78:3 – 40, 2017. Algorithms and Software for Computational Topology. 12

[11] Yi-Jen Chiang, Tobias Lenz, Xiang Lu, and Günter Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. Computational Geometry, 30(2):165 – 195, 2005. ISSN 0925-7721. 2, 3, 18

[12] Comsol. Multiphysics Reference Guide for COMSOL 4.2, 2011. 3, 22

[13] G. Coppola, S.J. Sherwin, and J. Peiró. Nonlinear particle tracking for high-order elements. Journal of Computational Physics, 172(1):356 – 386, 2001. 2

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848. 11

[15] M. de Berg and M. van Kreveld. Trekking in the alps without freezing or getting tired. Algorithmica, 18(3):306–323, 1997. 2

[16] Scott E. Dillard, Vijay Natarajan, Gunther H. Weber, Valerio Pascucci, and Bernd Hamann. Topology-guided tessellation of quadratic elements. International Journal of Computational Geometry & Applications (IJCGA), 19(2):195–211, April 2009. A preliminary version of this paper appeared in Proceedings of the 17th International Symposium on Algorithms and Computation, Kolkata, India, Lecture Notes in Computer Science (LNCS) 4288, 2006, 722–731. LBNL-63771. 3, 38

[17] Harish Doraiswamy. Reeb Graphs: Computation, Visualization and Applications. PhD thesis, Indian Institute of Science Bangalore, 2012. 20

[18] Harish Doraiswamy and Vijay Natarajan. Output-sensitive construction of reeb graphs. IEEE Transactions on Visualization and Computer Graphics, 18(1):146–159, January 2012. 2

[19] Harish Doraiswamy, Vijay Natarajan, and Ravi S. Nanjundiah. An exploration framework to identify and track movement of cloud systems. IEEE Transactions on Visualization and Computer Graphics, 19(12):2896–2905, December 2013. 2, 11

[20] Herbert Edelsbrunner and John L. Harer. Computational Topology : An Introduction. American Mathematical Society, Providence (R.I.), 2010. ISBN 978-0-8218-4925-5. URL http://opac.inria.fr/record=b1133235. 22

[21] Issei Fujishiro, Yuriko Takeshima, Taeko Azuma, and Shigeo Takahashi. Volume data mining using 3d field topology analysis. IEEE Comput. Graph. Appl., 20(5):46–51, September 2000. 2

[22] Mark Galassi. GNU scientific library : reference manual. Network Theory, Bristol, 2009. ISBN 0954612078. 12, 20

[23] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 20

[24] Charles Gueunet, Pierre Fortin, Julien Jomier, and Julien Tierny. Contour forests: Fast multi-threaded augmented contour trees. In 6th IEEE Symposium on Large Data Analysis and Visualization, 2016. 2

[25] Rahul Khardekar and David Thompson. Rendering higher order finite element surfaces in hardware. In Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE'03, pages 211–ff, 2003. 1

[26] Aaditya G Landge, Valerio Pascucci, Attila Gyulassy, Janine C Bennett, Hemanth Kolla, Jacqueline Chen, and Peer-Timo Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1020–1031, 2014. 2, 3

[27] S. Maadasamy, H. Doraiswamy, and V. Natarajan. A hybrid parallel algorithm for computing and tracking level set topology. In High Performance Computing (HiPC), 2012 19th International Conference on, pages 1–10, Dec 2012. 2, 3

[28] M. Meyer, B. Nelson, R. Kirby, and R. Whitaker. Particle systems for efficient and accurate high-order finite element visualization. IEEE Transactions on Visualization and Computer Graphics, 13(5):1015–1026, 2007. 2

[29] J.W. Milnor. Morse Theory. Annals of mathematics studies. Princeton University Press, 1963. ISBN 9780691080086. 6

[30] B. Nelson and R. M. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. IEEE Transactions on Visualization and Computer Graphics, 12(1):114–125, 2006. 2

[31] Blake Nelson, Eric Liu, Robert M. Kirby, and Robert Haimes. Elvis: A system for the accurate and interactive visualization of high-order finite element solutions. IEEE Transactions on Visualization and Computer Graphics, 18(12):2325–2334, 2012. 2

[32] Christian Pagot, D. Osmari, F. Sadlo, Daniel Weiskopf, Thomas Ertl, and J. Comba. Efficient Parallel Vectors Feature Extraction from Higher-Order Data. Computer Graphics Forum, 2011. 2

[33] Valerio Pascucci and Kree Cole-McLaughlin. Parallel computation of the topology of level sets. Algorithmica, 38(1):249–268, 2004. 2, 3

[34] Georges Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. Comptes Rendus Acad. Sciences, 222:847–849, 1946. 7

[35] Jean-François Remacle, Nicolas Chevaugeon, Émilie Marchandise, and Christophe Geuzaine. Efficient visualization of high-order finite elements. International Journal for Numerical Methods in Engineering, 69(4):750–771, 2007. ISSN 1097-0207. 1

[36] Himangshu Saikia, Hans-Peter Seidel, and Tino Weinkauf. Extended Branch Decomposition Graphs: Structural Comparison of Scalar Data. Computer Graphics Forum, 2014. 2

[37] Kersten Schmidt. Concepts - Numerical C++ Library for partial differential equations. TU Berlin, ETH Zürich, Matheon Research Center. 1

[38] W. J. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. P. Pebay, R. O'Bara, and S. Tendulkar. Methods and framework for visualizing higher-order finite elements. IEEE Transactions on Visualization and Computer Graphics, 12(4):446–460, 2006. 1, 2

[39] Pavel Solin. Hermes - Higher-Order Modular Finite Element System (User's Guide). University of Reno, Nevada, University of West Bohemia, Pilsen, Institute of Thermomechanics, Prague, Czech Republic, http://hpfem.org. URL http://hpfem.org/. 1

[40] Shigeo Takahashi, Yuriko Takeshima, and Issei Fujishiro. Topological volume skeletonization and its application to transfer function design. Graphical Models, 66(1):24 – 49, 2004. 2

[41] Sergey P. Tarasov and Michael N. Vyalyi. Construction of contour trees in 3d in o(n log n) steps. In Proceedings of the Fourteenth Annual Symposium on Computational Geometry, SCG '98, pages 68–75, New York, NY, USA, 1998. ACM. 2

[42] Dilip Mathew Thomas and Vijay Natarajan. Symmetry in scalar field topology. IEEE Transactions on Visualization and Computer Graphics, 17(12):2035–2044, December 2011. 2

[43] Marc van Kreveld, René van Oostrum, Chandrajit Bajaj, Valerio Pascucci, and Dan Schikore. Contour trees and small seed sets for isosurface traversal. In Proceedings of the Thirteenth Annual Symposium on Computational Geometry, SCG '97, pages 212–220, New York, NY, USA, 1997. ACM. 2

[44] Jan Verschelde. Polynomial homotopy continuation with phcpack. ACM Commun. Comput. Algebra, 44(3/4):217–220, January 2011. 10, 20

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

[46] Gunther H. Weber, Peer-Timo Bremer, Marcus S. Day, John B. Bell, and Valerio Pascucci. Feature tracking using reeb graphs. In Valerio Pascucci, Xavier Tricoche, Hans Hagen, and Julien Tierny, editors, Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications, pages 241–253. Springer Verlag, 2011. LBNL-4226E. 2

[47] David F Wiley, Henry R Childs, Benjamin F Gregorski, Bernd Hamann, and Kenneth I Joy. Contouring curved quadratic elements. In VisSym, 2003. 2

[48] Jianlong Zhou and Masahiro Takatsuka. Automatic transfer function generation using contour tree controlled residue flow model and color harmonics. IEEE Transactions on Visualization and Computer Graphics, 15(6):1481–1488, November 2009. 2