Computing Contour Trees for 2D Piecewise Polynomial Functions

Girijanandan Nucha, Georges-Pierre Bonneau, Stefanie Hahmann, Vijay Natarajan

▶ **To cite this version:**

Girijanandan Nucha, Georges-Pierre Bonneau, Stefanie Hahmann, Vijay Natarajan. Computing Contour Trees for 2D Piecewise Polynomial Functions. Computer Graphics Forum, 2017, 36 (3), pp.23-33. 10.1111/cgf.13165 . hal-01494431

HAL Id: hal-01494431
https://inria.hal.science/hal-01494431

Submitted on 23 Mar 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Eurographics Conference on Visualization (EuroVis) 2017
J. Heer, T. Ropinski and J. van Wijk
(Guest Editors)

Volume 36 (2017), Number 3

Computing Contour Trees for 2D Piecewise Polynomial Functions

Girijanandan Nucha[1], Georges-Pierre Bonneau[2], Stefanie Hahmann[2], and Vijay Natarajan[1]

[1]Indian Institute of Science, Bangalore, India
[2]Université Grenoble Alpes, CNRS (LJK), Inria, France

Abstract

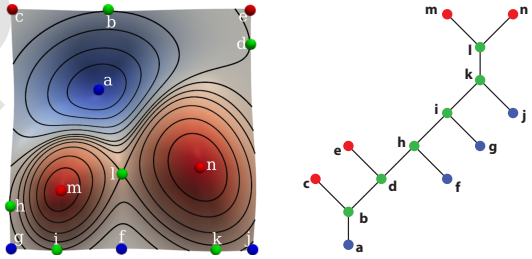
Contour trees are extensively used in scalar field analysis. The contour tree is a data structure that tracks the evolution of level set topology in a scalar field. Scalar fields are typically available as samples at vertices of a mesh and are linearly interpolated within each cell of the mesh. A more 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.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Author preprint

1. Introduction

Scientists and engineers are increasingly using higher order FEM simulations. Consider, as an example, the *h*p-adaptive variant [BG92] of finite element methods for which Nektar++ [CMC*15], Concepts [Sch], and Hermes [Sol] 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. In this paper, we study piecewise higher order real-valued functions defined on planar or curvilinear elements representing 2-manifold geometries.



(a) (b)

Figure 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.

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 nonlinear 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 [KT03,RCMG07] 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 [SBM*06]. Further, the use of high sampling density increases memory usage and may still introduce additional error.

Many visualization techniques for higher order scalar fields have begun to emerge in recent years [CSP01,WCG*03,NK06,SBM*06, MNKW07, POS*11, NLKH12]. 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

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

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. Contour trees are widely applied in the context of volume visualization – for efficient computation of isosurfaces [vKvOB*97], transfer function design [FTAT00, TTF04, WDC*07, ZT09, DN12], and for effective and flexible exploration of isosurfaces [CSvdP10]. The application of contour trees to volume data analysis such as feature extraction and tracking [BWT*11, WBD*11, DNN13], symmetry and similarity detection [TN11, SSW14] is also clearly demonstrated.

Many efficient algorithms have been proposed for computing the contour tree for two- and three-dimensional scalar fields [dBvK97, vKvOB*97, TV98, CSA03, CLLR05] and considerable efforts have been undertaken to develop parallel implementations [PCM04, MDN12, LPG*14, AN15, CWSA16, GFJT16]. 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. [DNW*09] 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 [PCM04] and Acharya and Natarajan [AN15] 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 [CS09] 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 [PCM04, AN15] and for piecewise linear interpolants [MDN12, LPG*14]. 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.

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. [CLLR05], 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 subproblems 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.

This algorithm may be considered as a hybrid approach between the monotone path tracing algorithm of Chiang et al. [CLLR05] and the two-pass union-find based algorithm of Carr et al. [CSA03]. Both algorithms have to be suitably extended to be made applicable to higher order polynomials. Our algorithm 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 quadratic and piecewise cubic polynomials. We also performed experiments on real-world scientific data obtained using COMSOL [Com11].

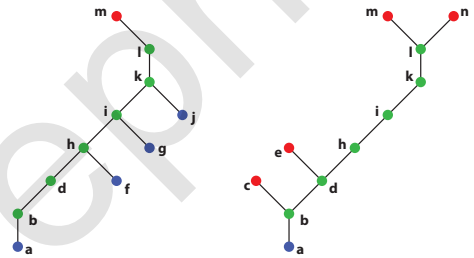


Figure 2: Join tree (a) and split tree (b) of the function described in Figure 1. The contour tree is the union of the join and split trees.

2. Background

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 $\nabla f(p) = 0$. 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* [Mil63] if and only if

- 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 both 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, 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 [Ree46]. 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 saddles are interior nodes of degree 3. 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

Author preprint

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

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 shows a 2D function and the corresponding contour tree. Nodes colored red, blue, and green correspond to maxima, minima, and saddles, 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* are defined in a manner analogous to the contour tree, by tracking connected components of the sub-level sets (preimage of $f^{-1}(-\infty, c]$) and the super-level sets (preimage of $f^{-1}[c, +\infty)$), respectively. Figure 2 shows the join and split tree of the scalar function shown in Figure 1. They are useful intermediate structures for computing the contour tree as explained in Section 3.2 and Section 3.6.

3. Algorithm

We now 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. 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 3b 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 3a. Figure 3d 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.

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

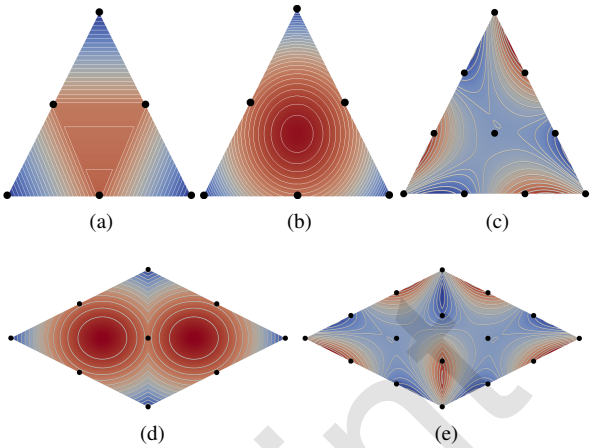


Figure 3: Linear and higher order interpolation over a triangle for a function sampled at multiple points (black). (a) Linear interpolation within each of the four triangles obtained by subdividing the input triangle. (b) Quadratic polynomial function interpolating the sample points. (c) Cubic polynomial 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.

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 consists of four steps:

1. Compute critical points for each patch.
2. Construct 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 [Ver11] 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 critical points lie on the XY-plane. The method reports points lying in the interior of the triangle. *Line-critical points*, defined as critical points

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Author preprint

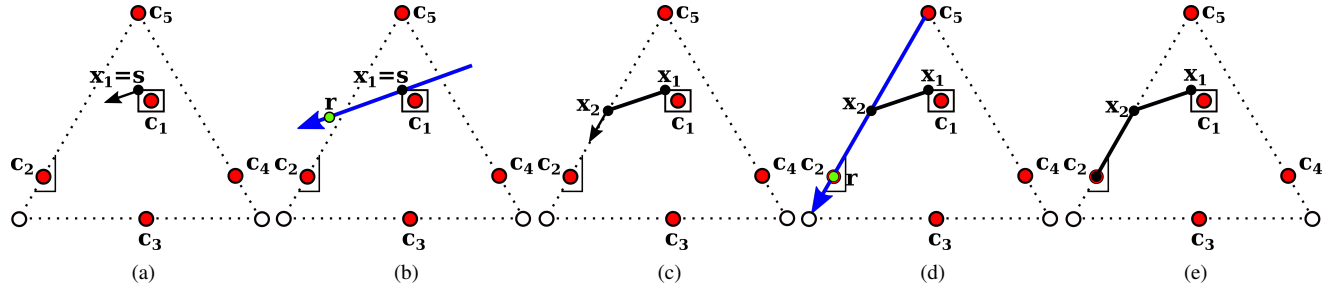


Figure 4: Tracing a monotone descending path from c_1. The steepest descent direction (blue) helps locate monotonic segments. If a path reaches the boundary, it is restricted to the boundary.

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 [CSA03, DNN13]. 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 [CLRS09]. 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 [Gal09]. To facilitate the identification of the number of MDPs originating at a critical point or vertex, we first compute disjoint axis-aligned bounding boxes for all of them. Chattopadhyay et al. [CVY17] 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

Author preprint

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

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.

Example. Figure 4 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 4(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 4(c). Next, the univariate polynomial along the blue ray is computed followed by locating the closest minimum r, see Figure 4(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 [AN15]. 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.

3.6. Contour tree

The global join and split trees are merged resulting in the global contour tree [CSA03]. This merge step is also described in previous work. For completeness, the merging procedure is described in Algorithm MergeJoinAndSplitTree. 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.

3.7. Degeneracies

A patch that contains at least one non-isolated critical point is said to be degenerate. Figure 5(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 5(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 [CSA03]. Inserting the line-critical points ensures that the subsequent stitching step applies to all patches, degenerate or otherwise, without modification.

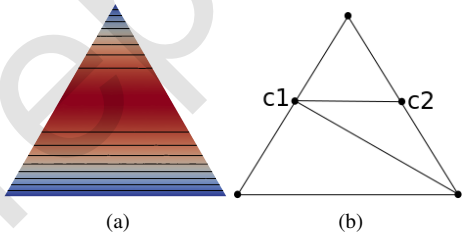


Figure 5: (a) A degenerate patch. (b) Subdividing the patch into triangles after inserting all line-critical points. The local join tree and local split tree is computed by assuming piecewise linear interpolation within each smaller triangle.

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. This follows from the result of Chiang et al. [CLLR05]. The algorithm traces all required MDPs from a critical point / triangle vertex as shown in Lemma 3 in the appendix. 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 1 and 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.

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

Author preprint

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Algorithm 1: BUILDLOCALJOINTREE

Input: Polynomial function f defined on a triangle that lies on the XY plane
Input: Set C of critical points and triangle vertices. Minimum distance, d_{min}, between points in C.
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 $x_i \in C$. */
/* Let R_c denote the bounding box of a point $c \in C$. */
/* Let S denote the set of starting points of monotone descending paths. These are points x_1 that lie on the bounding box of a point in C. */
/* Let UF denote a union-find data structure to store collection of points in C. Each set in UF is represented by the critical point with highest function value. */

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

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 requires$O(n_t n_c)$ 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 $O(n_t d^2 n_p^2)$.

Author preprint

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Algorithm 2: StitchJoinTrees [AN15]

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

Algorithm 3: MergeJoinAndSplitTree [AN15]

Input: Global join tree JT and split tree ST
Output: Contour tree CT

$L \longleftarrow$ Set of leaves in JT and ST
while $L \neq \phi$ **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 $l \neq T.root$ 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

4. Experimental Results

We now present results of computational experiments on 2D quadratic and cubic polynomials. The algorithm is implemented in C++. The open source C++ library Eigen [GJ*10] is used for finding the solution of a system of linear equations, the GNU Scientific Library [Gal09] for finding the roots of a univariate polynomial, and PHCpack [Ver11] for finding roots of a polynomial system. 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.

Piecewise quadratic input. The thermal conductor dataset represents the temperature distribution at the surface of an electronic component computed by solving a multi-physics simulation using COMSOL [Com11]. The temperature field is a polynomial of maximal degree 2 within each triangle. The six samples are located at the triangle vertices and at the mid-point of the edges, as shown in Figure 3b. For this dataset, the geometry of the elements is also modeled as a quadratic polynomial that accurately fits the curved surface. Figure 6a-6c shows the temperature field, its critical points together with a set of contours, and 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 [EH10], which is equal to the difference in function value at the pair of critical points.

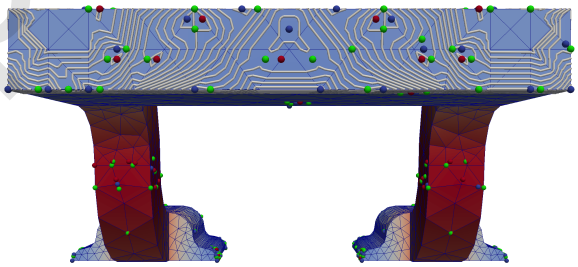


Figure 7: Top of the conductor contains multiple degeneracies. The algorithm identifies all the critical points and handles the degenerate patches gracefully.

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 [CSvdP10]. Figure 6d 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 7. 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

Author preprint

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

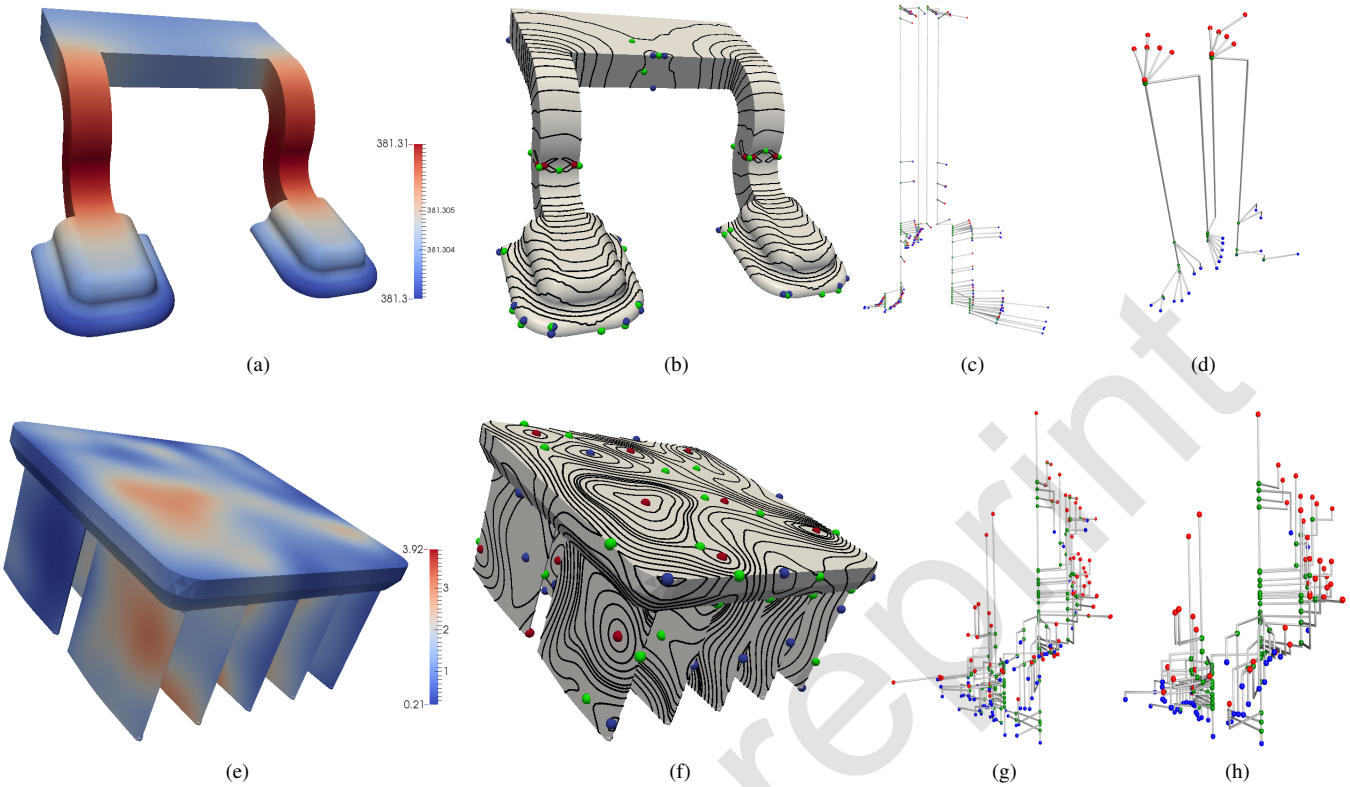


Figure 6: Contour trees for piecewise quadratic and cubic polynomials (a) Temperature distribution on the surface of a thermal conductor as a piecewise quadratic function. (b) Critical points and contour lines. (c) Contour tree consisting of 530 nodes. (d) Contour tree after simplification using a persistence threshold of 0.1% contains 67 nodes. (e) A cubic function defined on a heater geometry. (f) Critical points and contour lines. (g) Contour tree consisting of 268 nodes. (h) Contour tree with 178 nodes obtained after simplifying using a persistence threshold of 0.5%.

contains two similar subtrees as expected. The subtrees correspond to the symmetric legs of the conductor. It took approximately 15 minutes to compute the contour tree for this data set (8598 vertices, 4298 triangles). Algorithmic and code optimizations may result in significant reduction in running times. For example, several steps of the algorithm can be parallelized. We plan to do this in future.

Figure 8 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. The section of the contour tree shown in Figure 8(i) contains more number of maxima than Figure 8(j). However, these additional maxima are introduced due to flat regions and have zero persistence.

Piecewise cubic input. 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 6e-6h 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.

Figure 9 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

Author preprint

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

on an equal number of triangles. A 10-fold increase in number of triangles seems to be necessary to obtain similar contours.

5. Conclusions

We have described a simple 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.

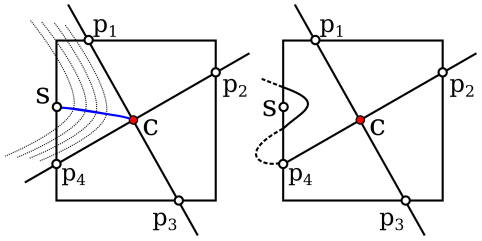


Figure 10: (a) Bounding box R_c for a critical point c showing its neighborhood N_c. (b) A box containing c, which is larger than the required bounding box.

Appendix

We prove three lemmas that are required to show the correctness of the algorithm.

Lemma 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 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 $l(I)$. □

Lemma 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 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 10(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 10(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. □

Acknowledgements

This research was partially funded by a grant from the Indo-French Centre for Applied Mathematics. The first author thanks Talha Bin Masood for active discussions and help with dataset preparation.

References

[AN15] ACHARYA A., NATARAJAN V.: A parallel and memory efficient algorithm for constructing the contour tree. In 2015 IEEE Pacific Visualization Symposium (PacificVis) (2015), pp. 271–278. 2, 5, 7

[BG92] BABUŠKA I., GUO B. Q.: The h, p and h-p version of the finite element method: Basis theory and applications. Adv. Eng. Softw. 15, 3-4 (Nov. 1992), 159–174. 1

[BWT*11] BREMER P.-T., WEBER G., TIERNY J., PASCUCCI V., DAY M., BELL J.: Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. IEEE Transactions on Visualization and Computer Graphics 17, 9 (Sept. 2011), 1307–1324. 2

[CLLR05] CHIANG Y.-J., LENZ T., LU X., ROTE G.: Simple and optimal output-sensitive construction of contour trees using monotone paths. Computational Geometry 30, 2 (2005), 165 – 195. 2, 5

[CLRS09] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: Introduction to Algorithms, Third Edition, 3rd ed. The MIT Press, 2009. 4

[CMC*15] CANTWELL C., MOXEY D., COMERFORD A., BOLIS A., ROCCO G., MENGALDO G., GRAZIA D. D., YAKOVLEV S., LOMBARD J.-E., EKELSCHOT D., JORDI B., XU H., MOHAMIED Y., ESKILSSON C., NELSON B., VOS P., BIOTTO C., KIRBY R., SHERWIN S.: Nektar++: An open-source spectral/ element framework. Computer Physics Communications 192 (2015), 205 – 219. 1

[Com11] COMSOL: Multiphysics Reference Guide for COMSOL 4.2, 2011. 2, 7

[CS09] CARR H., SNOEYINK J.: Representing interpolant topology for contour tree computation. In Topology-Based Methods in Visualization II, Hege H.-C., Polthier K., Scheuermann G., (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 59–73. 2

[CSA03] CARR H., SNOEYINK J., AXEN U.: Computing contour trees in all dimensions. Computational Geometry 24, 2 (2003), 75 – 94. 2, 4, 5

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Author preprint

[CSP01] COPPOLA G., SHERWIN S., PEIRÃŞ J.: Nonlinear particle tracking for high-order elements. Journal of Computational Physics 172, 1 (2001), 356 – 386. 1

[CSvdP10] CARR H., SNOEYINK J., VAN DE PANNE M.: Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. Comput. Geom. Theory Appl. 43, 1 (Jan. 2010), 42–58. 2, 7

[CVY17] CHATTOPADHYAY A., VEGTER G., YAP C. K.: Certified computation of planar morse-smale complexes. Journal of Symbolic Computation 78 (2017), 3 – 40. Algorithms and Software for Computational Topology. 4

[CWSA16] CARR H. A., WEBER G. H., SEWELL C. M., AHRENS J. P.: Parallel peak pruning for scalable smp contour tree computation. In 6th IEEE Symposium on Large Data Analysis and Visualization (2016). 2

[dBvK97] DE BERG M., VAN KREVELD M.: Trekking in the alps without freezing or getting tired. Algorithmica 18, 3 (1997), 306–323. 2

[DN12] DORAISWAMY H., NATARAJAN V.: Output-sensitive construction of reeb graphs. IEEE Transactions on Visualization and Computer Graphics 18, 1 (Jan. 2012), 146–159. 2

[DNN13] DORAISWAMY H., NATARAJAN V., NANJUNDIAH R. S.: An exploration framework to identify and track movement of cloud systems. IEEE Transactions on Visualization and Computer Graphics 19, 12 (Dec. 2013), 2896–2905. 2, 4

[DNW*09] DILLARD S. E., NATARAJAN V., WEBER G. H., PASCUCCI V., HAMANN B.: Topology-guided tessellation of quadratic elements. International Journal of Computational Geometry & Applications (IJCGA) 19, 2 (Apr. 2009), 195–211. 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. 2

[EH10] EDELSBRUNNER H., HARER J. L.: Computational Topology : An Introduction. American Mathematical Society, Providence (R.I.), 2010. 7

[FTAT00] FUJISHIRO I., TAKESHIMA Y., AZUMA T., TAKAHASHI S.: Volume data mining using 3d field topology analysis. IEEE Comput. Graph. Appl. 20, 5 (Sept. 2000), 46–51. 2

[Gal09] GALASSI M.: GNU scientific library : reference manual. Network Theory, Bristol, 2009. 4, 7

[GFJT16] GUEUNET C., FORTIN P., JOMIER J., TIERNY J.: Contour forests: Fast multi-threaded augmented contour trees. In 6th IEEE Symposium on Large Data Analysis and Visualization (2016). 2

[GJ*10] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3. http://eigen.tuxfamily.org, 2010. 7

[KT03] KHARDEKAR R., THOMPSON D.: 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 (2003), GRAPHITE'03, pp. 211–ff. 1

[LPG*14] LANDGE A. G., PASCUCCI V., GYULASSY A., BENNETT J. C., KOLLA H., CHEN J., BREMER P.-T.: 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 (2014), pp. 1020–1031. 2

[MDN12] MAADASAMY S., DORAISWAMY H., NATARAJAN V.: A hybrid parallel algorithm for computing and tracking level set topology. In High Performance Computing (HiPC), 2012 19th International Conference on (Dec 2012), pp. 1–10. 2

[Mil63] MILNOR J.: Morse Theory. Annals of mathematics studies. Princeton University Press, 1963. 2

[MNKW07] MEYER M., NELSON B., KIRBY R., WHITAKER R.: Particle systems for efficient and accurate high-order finite element visualization. IEEE Transactions on Visualization and Computer Graphics 13, 5 (2007), 1015–1026. 1

[NK06] NELSON B., KIRBY R. M.: Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. IEEE Transactions on Visualization and Computer Graphics 12, 1 (2006), 114–125. 1

[NLKH12] NELSON B., LIU E., KIRBY R. M., HAIMES R.: Elvis: A system for the accurate and interactive visualization of high-order finite element solutions. IEEE Transactions on Visualization and Computer Graphics 18, 12 (2012), 2325–2334. 1

[PCM04] PASCUCCI V., COLE-MCLAUGHLIN K.: Parallel computation of the topology of level sets. Algorithmica 38, 1 (2004), 249–268. 2

[POS*11] PAGOT C., OSMARI D., SADLO F., WEISKOPF D., ERTL T., COMBA J.: Efficient Parallel Vectors Feature Extraction from Higher-Order Data. Computer Graphics Forum (2011). 1

[RCMG07] REMACLE J.-F., CHEVAUGEON N., MARCHANDISE Ã., GEUZAINE C.: Efficient visualization of high-order finite elements. International Journal for Numerical Methods in Engineering 69, 4 (2007), 750–771. 1

[Ree46] REEB G.: Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. Comptes Rendus Acad. Sciences 222 (1946), 847–849. 2

[SBM*06] SCHROEDER W. J., BERTEL F., MALATERRE M., THOMPSON D., PEBAY P. P., O'BARA R., TENDULKAR S.: Methods and framework for visualizing higher-order finite elements. IEEE Transactions on Visualization and Computer Graphics 12, 4 (2006), 446–460. 1

[Sch] SCHMIDT K.: Concepts - Numerical C++ Library for partial differential equations. TU Berlin, ETH Zürich, Matheon Research Center. 1

[Sol] SOLIN P.: 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

[SSW14] SAIKIA H., SEIDEL H.-P., WEINKAUF T.: Extended Branch Decomposition Graphs: Structural Comparison of Scalar Data. Computer Graphics Forum (2014). 2

[TN11] THOMAS D. M., NATARAJAN V.: Symmetry in scalar field topology. IEEE Transactions on Visualization and Computer Graphics 17, 12 (Dec. 2011), 2035–2044. 2

[TTF04] TAKAHASHI S., TAKESHIMA Y., FUJISHIRO I.: Topological volume skeletonization and its application to transfer function design. Graphical Models 66, 1 (2004), 24 – 49. 2

[TV98] TARASOV S. P., VYALYI M. N.: Construction of contour trees in 3d in o(n log n) steps. In Proceedings of the Fourteenth Annual Symposium on Computational Geometry (New York, NY, USA, 1998), SCG '98, ACM, pp. 68–75. 2

[Ver11] VERSCHELDE J.: Polynomial homotopy continuation with phcpack. ACM Commun. Comput. Algebra 44, 3/4 (Jan. 2011), 217–220. 3, 7

[vKvOB*97] VAN KREVELD M., VAN OOSTRUM R., BAJAJ C., PASCUCCI V., SCHIKORE D.: Contour trees and small seed sets for isosurface traversal. In Proceedings of the Thirteenth Annual Symposium on Computational Geometry (New York, NY, USA, 1997), SCG '97, ACM, pp. 212–220. 2

[WBD*11] WEBER G. H., BREMER P.-T., DAY M. S., BELL J. B., PASCUCCI V.: Feature tracking using reeb graphs. In Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications (2011), Pascucci V., Tricoche X., Hagen H., Tierny J., (Eds.), Springer Verlag, pp. 241–253. LBNL-4226E. 2

[WCG*03] WILEY D. F., CHILDS H. R., GREGORSKI B. F., HAMANN B., JOY K. I.: Contouring curved quadratic elements. In VisSym (2003). 1

[WDC*07] WEBER G. H., DILLARD S. E., CARR H., PASCUCCI V., HAMANN B.: Topology-controlled volume rendering. IEEE Transactions on Visualization and Computer Graphics 13, 2 (Mar. 2007), 330–341. 2

[ZT09] ZHOU J., TAKATSUKA M.: Automatic transfer function generation using contour tree controlled residue flow model and color harmonics. IEEE Transactions on Visualization and Computer Graphics 15, 6 (Nov. 2009), 1481–1488. 2

Author Copy

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

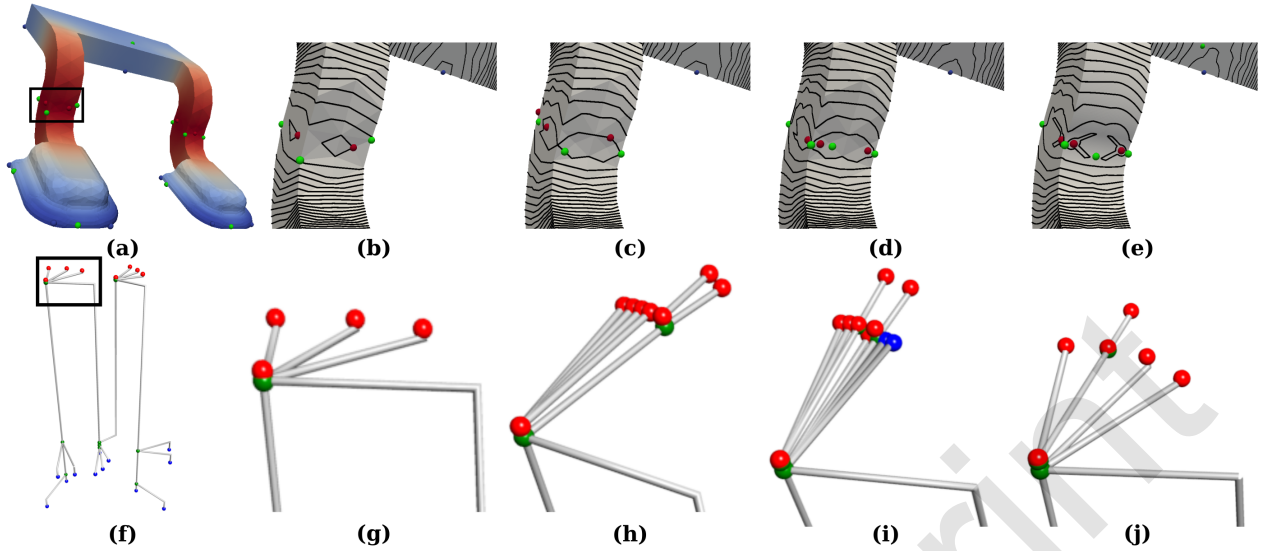


Figure 8: Visual comparison with piecewise linear (PL) approximations. (a) A PL approximation of the temperature field over the conductor. (f) Contour tree for the PL approximation. (b-d) Close up of region of thermal conductor showing critical points and contours for successive PL subdivisions of conductor containing 4298, 17192, and 68768 triangles, respectively. (g-i) Corresponding nodes and arcs in the contour tree. (e) Critical points and contour lines for piecewise quadratic function with 4298 triangles. (j) Nodes and arcs from contour tree of the piecewise quadratic function.

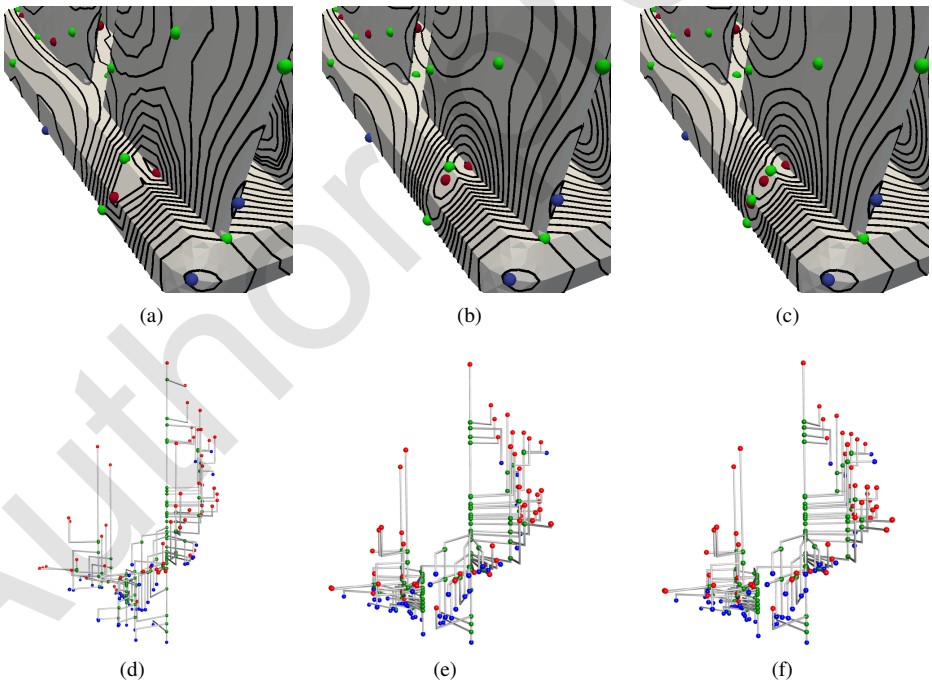


Figure 9: Visual comparison with PL approximations. (a) Close up view of heater dataset. Critical points computed using a PL approximation on a mesh with 12438 triangles.(d) Corresponding contour tree containing 210 nodes. (b) PL approximation with 111942 triangles. (e) Corresponding contour tree with 208 critical points. (c) Close up of heater showing critical points computed using a piecewise cubic function with 12438 triangles. (f) Corresponding contour tree containing 212 critical points.

Author preprint

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.