Geometry-Preserving Topological Landscapes

Kenes Beketayev[1,2*] Gunther H. Weber[1] Dmitriy Morozov[1] Aidos Abzhanov[3]
Bernd Hamann[2]
[1] Computational Research Division, Lawrence Berkeley National Laboratory
[2] Institute for Data Analysis and Visualization, Computer Science Department, University of California, Davis
[3] Computer Science Department, Center for Energy Research, Nazarbayev University

Abstract

We propose a novel technique for building *geometry-preserving* topological landscapes. Our technique creates a direct correlation between a scalar function and its topological landscape. This correlation is accomplished by introducing the notion of geometric proximity into the topological landscapes, reflecting the distance of topological features within the function domain. Furthermore, this technique enables direct comparative analysis between scalar functions, as long as they are defined on the same domain.

We describe a construction technique that consists of three stages: contour tree computation, contour tree layout, and landscape construction. We provide a detailed description for the latter two steps. For the contour tree layout stage, we discuss dimension reduction and edge routing techniques that produce a drawing of the contour tree on the plane that preserves the geometric proximity. For the landscape construction stage, we develop a contour construction algorithm that takes the contour tree layout as an input, adds contours at heights that correspond to saddles of the contour tree, and produces a contour map. After an additional triangulation step, this construction method results in the landscape that has the same contour tree as the original function.

CR Categories: I.3.8 [Computing Methodologies]: Computer Graphics—Applications I.3.M [Computer Graphics]: Miscellaneous—Scalar Field Visualization

Keywords: scalar field topology, topological landscapes, visual metaphor, multidimensional scaling

1 Introduction

In light of the recent advances in simulation capabilities and the growth of available computational power, visual exploration of scientific data is becoming an increasingly important task. An integral part of visual exploration is the presentation of helpful two- and three-dimensional abstractions that provide an insight into the structure of the scientific data.

One approach to creating such abstractions is based on the use of topological information. Important insights are known to be obtained using topological analysis in a number of fields [Weber et al. 2007b; Bremer et al. 2009]. In particular, the contour tree was extensively used in scalar data analysis and visualization [Bajaj et al. 1998; van Kreveld et al. 1997; Carr et al. 2003; Mizuta et al. 2004]. However, interpreting the contour tree usually requires at least an intermediate level understanding of Morse theory and related concepts. As a result, topological landscapes [Weber et al. 2007a] have been developed to convey the same information in a more intuitive manner, namely as a two-dimensional terrain with a same level-set topology as the original data set.

A major feature that is missing in the original technique by Weber et al. [2007a] and its modifications [Harvey and Wang 2010; Oesterling et al. 2010a; Oesterling et al. 2011] is a correlation between the geometrical placement of the topological features in the landscape and the original domain space. This property simplifies the process of understanding the landscapes by domain scientists by correlating the proximity of the topological features. It also adds a new capability of comparing functions. However, given the potential complexity and higher dimensionality of the data, it is not simple to develop it.

In this paper, we propose a new technique for building geometry-preserving topological landscape from a contour tree, which addresses the problem of correlating the function and its topological landscape. The proposed technique also makes it possible to compare several complex scalar functions via their topological landscapes, something previous versions of the topological landscapes lack.

The technique consists of three major stages: first, we compute a contour tree of the input function; then, we use the dimension reduction to project the vertices of the contour tree onto the plane and graph drawing algorithm to connect them by non-intersecting edges; finally, we apply a contour construction algorithm, which creates contours of different height and produces the depiction of the terrain similar to the contour map. Finally, after triangulating the terrain, we render the surface that results in the geometry-preserving topological landscape. We demonstrate the use of our technique in case of performance data analysis.

Our contributions are a three-stage technique for creation of geometry-preserving topological landscapes; the capability to compare several functions via their landscapes; and an example application in performance data analysis.

2 Related Work

2.1 Scalar Field Topology

Scalar field topology characterizes the data by topology changes of level sets. Given a real-valued function without degenerate critical points, level set topology changes only at isolated critical points [Milnor 1963]. The contour tree tracks the change in topology of level sets as they appear at minimum, split and merge at saddles, and disappear at maximum [Carr et al. 2003].

The *persistence* of the level set is its "lifespan," computed as an absolute function value difference between the critical points that constitute the given level set. The *branch decomposition* [Pascucci et al. 2005] is a multi-resolution representation of the contour tree that decomposes it into the branches that correspond to extremum-saddle pairs. This representation creates a hierarchical structure,

*e-mail:kbeketayev@lbl.gov

which allows a traversal of the contour tree efficiently up to a desired level of detail.

2.2 Contour Tree Visualization

A number of visual models for the contour tree exists. For example, planar and volume graph representations of the contour tree are widely used [Gansner and North 1999; Heine et al. 2011]. In fact, our technique produces the planar graph representation of the contour tree as an intermediary result (detailed discussion is in Section 2.4).

While a graph representation is a good starting point, it is often hard to visually derive the required information from it, especially in case of large contour trees. To address this issue, more intuitive visualizations were proposed. The contour nest [Mizuta et al. 2004; Mizuta et al. 2006] focuses on the nesting properties of isosurfaces and represents the contour tree as a set of nested rectangles, where rectangle size corresponds to feature size. We consider another metaphor that provides the requested insight called *topological landscape*, proposed by Weber et al. [Weber et al. 2007a]. It proved to be useful in several application fields [Harvey and Wang 2010; Oesterling et al. 2010a; Oesterling et al. 2011; Oesterling et al. 2010b].

In the original work [Weber et al. 2007a], a topological landscape was constructed from the persistence-based branch decomposition [Pascucci et al. 2005] of the contour tree [Carr et al. 2003], such that each branch corresponds to some box element of the landscape. Box elements were arranged using a spiral layout. Later, several modified versions of the original topological landscapes were proposed [Oesterling et al. 2010a; Oesterling et al. 2011], including the one that uses tree map layout scheme for box elements [Harvey and Wang 2010]. However, both layout schemes (spiral and tree map) ignore feature proximity in the domain. Therefore, we propose a new layout scheme that consists of three steps: projecting vertices of the contour tree onto the plane; projecting edges of the contour tree onto the plane; drawing contours around the projected contour tree. These steps produce the terrain representation similar to the topographic map, which after additional triangulation step results in the desired landscape.

2.3 Dimension Reduction

In the first stage of the proposed layout method, vertices of the contour tree are projected onto the plane. This projection can be achieved by number of dimension reduction techniques (we refer to the work by Fodor [2002] for an overview). In particular, we choose classical multidimensional scaling [Torgerson 1952]. It is a well-accepted method [Garth et al. 2004; Engel et al. 2011] that preserves relative distances between points, as they are projected from the original domain onto the plane.

2.4 Graph Drawing

Once the vertices of the contour tree are fixed, we need to layout the edges onto the plane. A number of algorithms for drawing a tree on the plane exist, see the book by Tamassia [2012] for an overview. Level-based layouts arrange the tree in a hierarchical fashion, starting from the root vertex. Radial layouts draw the vertices of the tree on concentric circles with different radii. However, these methods assume that vertex locations are flexible, which is not true in our case. Pach and Wenger [2001] address the problem of drawing a tree with fixed vertex locations. However, the authors mainly discuss theoretical issues and limitations, such as an upper bound on number of bends per edge. While they describe a high-level scheme for drawing an actual tree, it appears to be theoretical and mostly for illustration purposes. In particular, they allow the spacing between the edges to be infinitely small, leading to potentially cluttered layouts.

Alternatively, it is possible to draw the edges iteratively, using edge routing algorithms. Inspired by the fields of robotics and circuit design, these methods are based on notion of finding the route between two fixed locations with given obstacles. Usually, these obstacles are the vertex locations and previously drawn edges, given that layout should be non- intersecting. Drawing an edge as a single straight line is not always possible due to potential obstacles. Instead, an edge can be drawn via polyline, curve, etc. The majority of the edge routing algorithms tries to optimize certain predefined requirements for the route [Dwyer and Nachmanson 2009], such as minimizing a number of bends or maximizing the smallest bending angle. However, we need the graph layout as an input for constructing contours. This imposes a requirement that edges should be spread out. Currently, there is no method satisfying this requirement. Therefore, we propose an edge routing algorithm, which for each edge uses the Voronoi diagram of already drawn parts of the contour tree, hence satisfying the stated requirement.

Once we have computed the layout of the contour tree on the plane, we apply the contour construction algorithm, described in Section 3.3. Finally, we compute the constrained Delaunay triangulation and render the surface to produce the resulting landscape.

3 Algorithm

In this section we present an algorithm for building the geometry-preserving landscape. The general outline of the algorithm is as follows. First, we compute the contour tree from the data. Then, we use multidimensional scaling to project its vertices onto the plane. Subsequently, we project the edges of the contour tree onto the plane. Once the contour tree layout is finalized, we construct contours based on the contour tree, and triangulate the resulting contour map to produce the required landscape.

3.1 Contour Tree Computation

Consider a scalar function $f : R^d \rightarrow R$. We compute a contour tree of the function f using the algorithm by Carr et al. [2003], and construct persistence-based branch decomposition [Pascucci et al. 2005] from it. Additionally, we record the spatial locations of contour tree vertices in the original domain space.

3.2 Contour Tree Layout

Let's denote a point set $P_d = \{p_1, p_2, ..., p_N\}$ to be the spatial locations of the contour tree vertices $V = \{v_1, v_2, .., v_N\}$ in d-dimensional space. We apply the classical multidimensional scaling method to the point set to project it to two dimensions $P_d \xrightarrow{MDS} P_2$.

Once the locations of the vertices are calculated, we start drawing the edges of the contour tree iteratively. For this purpose we design an algorithm that for a given edge $e = (v_i, v_j)$ constructs a polyline that connects corresponding points p_i, p_j on the plane. The algorithm computes the Voronoi diagram for all existing elements (already drawn vertices and edges), finds an intersection-free path along the edges of the diagram from p_i to p_j, and adds it as a polyline to the layout. If more than one path exists, Dijkstra's algorithm can be used optionally to find the shortest (in terms of line segments or the distance travelled along those line segments) path. To guarantee the existence of at least one path between any two points, we

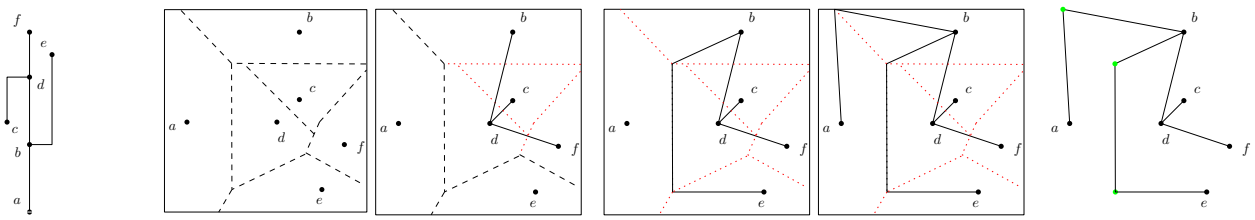


Figure 1: The contour tree layout process. *First we project the vertices. Then, we start iteratively connecting edges. The initial three edges* $(f, d), (c, d)$, *and* (d, b), *connect the points directly. Then, the edge* (e, b) *is routed via Voronoi diagram. Finally, the edge* (b, a) *is connected through intersections of Voronoi diagram and bounding box, since no path through Voronoi diagram is available.*

allow routing along the bounding box of drawn parts of the contour tree, slightly extended in all directions.

We found that ordering the edges by their increasing L_2-norm length on the plane (i.e., direct Euclidean distance between the end points of an edge) leads to fewer and shorter polylines in the final layout. We note that often a few swaps in the ordering can lead to the significant improvements in the layout, thus it is recommended to try several slightly different orderings. One example criterion can be swapping edges with close L_2-norm lengths.

3.3 Landscape Construction

At this point we assume that we have obtained an intersection-free contour tree layout, given by the set of points P and polylines on the plane. Now we construct a landscape from the contour tree layout.

The main idea is to take a branch decomposition of the contour tree, and for each branch (which is an extremum–saddle pair) construct a contour that goes through its saddle and encloses its extremum. If the branch has no children, i.e., it is a leaf branch, we construct a *triangle contour*, see Figure 3. Otherwise, i.e., if it is a parent branch, we sort its children in an ascending or descending order, depending on the kind of extremum of the parent branch. Then, for each child branch we perform two operations. First, we construct contours for it as needed (recursively, if it is also a parent branch). Second, we construct an *offset contour* for the previously processed part of the parent, see Figure 4. One exception is the first child, for which the previously processed part of the parent is an edge (e.g., processed part (f, d) of the parent branch (f, a) in Figure 2), hence we use the triangle contour.

The sequence of processing the branches is based on the hierarchical traversal of the given branch decomposition. We start with a root branch. If it is a simple branch, we construct the triangle contour and return. Otherwise, we apply the parent branch handling described above. For all child branches we call this procedure recursively.

```
b_cur(e_cur, s_cur)
b_cur ← root branch
function DRAWCONTOURS(b_cur)
    S ← all child saddles
    if S = ∅ then
        TRICONTOUR(b_cur)
        return
    if val(e_cur) > val(s_cur) then
        sortDecreasing(S)
    else
        sortIncreasing(S)
    for s ∈ S do
        if s is first then
            TRICONTOUR((e_cur, s))
        else
            OFFSETCONTOUR((e_cur, s))
    DRAWCONTOURS(b_s)
```

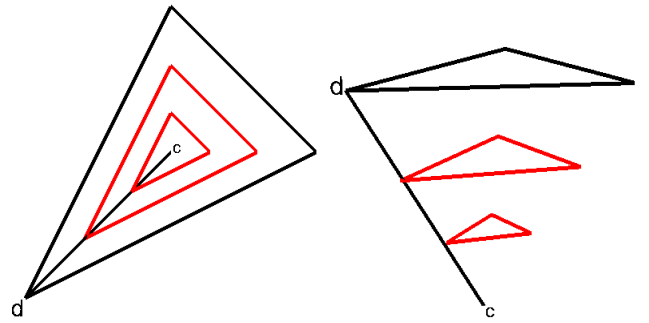


Figure 3: Triangle contour. *Constructing a triangle contour (black) for the branch* (c, d) *creates a valley with inner contours (red) corresponding to the given branch.*

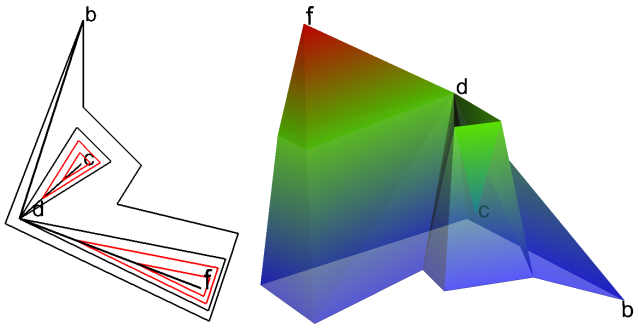


Figure 4: Offset contour. *Constructing an offset contour (black) around the upper section of the saddle* b *creates a continuation of the peak* f *with the valley* c *on it.*

Now we provide geometric details of drawing contours. We represent each contour as a simply-connected polygon and maintain the list of all polygons, associated with each saddle.

The function TRICONTOUR draws a simple triangle around the given edge, see Figure 3. If the edge is polyline, we still compute a single triangle for an edge of the same length as the given polyline edge. Then, we recompute the triangle coordinates at the bends of the polyline edge (see Figure 2, contour of the edge (e, b)).

The function OFFSETCONTOUR takes as an input an upper/lower

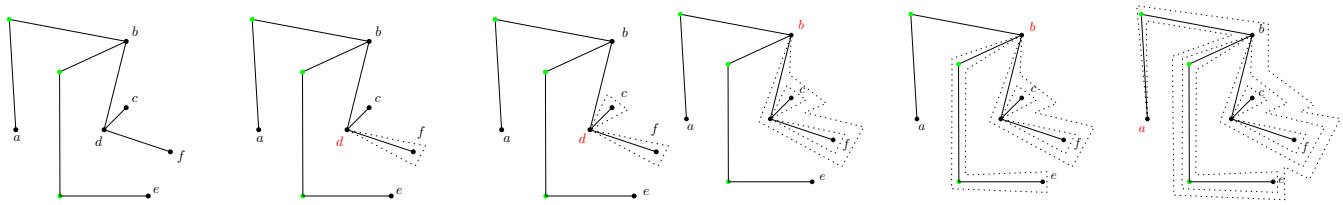


Figure 2: Contour map construction process. *We start from a root branch (f, a). It has children, so we sort them in descending manner, and start with the child branch (c, d) that has highest saddle d. First, the upper (processed) section of it is an edge (f, d), so we construct the triangle contour. Second, its child branch (c, d) is a leaf, so we also use the triangle contour. However, the upper section for the next saddle b is complex, so we use the offset contour. Again, the corresponding child branch (e, b) is a leaf, so we use the triangle contour. Finally, for the minimum a, we construct the offset contour for the branch (f, a), which concludes the contour drawing.*

(already processed) section of some saddle s. It consists of contours that belongs to the previous saddle s_{prev} and an edge to it from the current saddle, see Figure 4. Our aim is to enclose these into a single contour. First, we note that each contour of saddle s_{prev} has a corresponding polygon. We combine all these polygons into a single, non-convex, strictly-simple polygon $P_{s_{prev}}$, where strictly-simple means that the area, bounded by the polygon, is simply-connected. We apply a 2D polygon offsetting method, provided by *Computational Geometry Algorithms Library* (CGAL) [cga], to polygon $P_{s_{prev}}$. This offsetting produces new, non-convex, strictly-simple polygon $P^{offset}_{s_{prev}}$. Second, we construct the triangle for the edge (s, s_{prev}). Finally, we apply 2D polygon join operation (also provided by CGAL) to the computed offset polygon and the triangle, and produce the output polygon $P_s = P^{offset}_{s_{prev}} \cup T_{(s,s_{prev})}$. This polygon is saved to the list of contours for the current saddle s.

Once all the contours are drawn, we apply the constrained Delaunay algorithm (provided in CGAL) to create a triangulated surface, resulting in a landscape with the given contour tree.

3.4 Comparable Landscapes

To produce comparable landscapes from several contour trees $T_1, ..., T_k$, we combine their vertices into one set $V = \{V_1, \ldots, V_k\}$ and apply the projection step of the algorithm to corresponding point set $P_d \xrightarrow{MDS} P_2$. After projection, we again separate each point set and continue with each separately. As a result, we obtain landscapes that are comparable in terms of the locations of corresponding elements.

4 Results

We have applied our technique to the problem of tuning a ray casting algorithm on a multicore shared-memory system. We follow the study, conducted by Bethel and Howison [2012] that explores wide range of potential tuning parameters for this algorithm. For the purposes of demonstrating our technique, we select the parameters mainly targeted by the study, namely the work block width {1,...,512} and height {1,...,512}, and levels of concurrency {1,2,4,8}. We also consider an additional parameter, the ray sampling method option, which can be either *nearest-neighbor* or *trilinear*. This parameter produces two data sets (one for each option) that we will use later to demonstrate the comparative capability of the geometry-preserving landscapes.

Each data set is obtained by running a ray casting algorithm based on a selected ray sampling strategy and all combinations of other input parameters, and recording the resulting normalized running time, given in milliseconds. High running time corresponds to poor performance, depicted in the landscapes as peaks. In Figure 5 we show the volume rendering of the performance data set (produced by selecting trilinear ray sampling option), together with corresponding geometry-preserving topological landscape.

First we note that it is visually easier to see the correlation between features in the landscape than a direct visualization. For example, from the landscape we can derive the closeness of the peaks, the fact that needs some effort to check otherwise (e.g., by sweeping different isovalues and constructing corresponding isosurfaces, see Figure 6). This is an important observation, which suggests to the domain scientist that there exists a subspace enclosing poor configurations to be avoided.

Further we produce the comparable landscapes by combining the projection step for two data sets with trilinear and nearest-neighbors ray sampling options (as discussed in Section 3.2). We observe a correlation between two data sets (see Figure 6), where very high peaks, i.e., particularly poor performing configurations, appear to be close in both data sets.

An interesting observation can be made about moderate peaks (see Figure 6). In the case of the trilinear ray sampling option, they are distributed close to high peaks (circle A in Figure 6(a)). However, in the case of the nearest-neighbors ray sampling option, they are distant from the main cluster (distinct circles A and B in Figure 6(b)). This observation provides an important insight into the underlying behavior of the algorithm and the ray sampling option selection. It can be suggested that if the parameter ranges are limited, the trilinear ray sampling option is preferable, since unlike nearest-neighbors, in the lower parameter ranges it has no moderate peaks, i.e., moderately poor configurations. This fact can be easily missed in the direct visual exploration or other versions of topological landscapes.

5 Conclusion

We presented a novel technique for building geometry-preserving topological landscapes. In particular, we described three stages of building such landscapes, the contour tree computation, the contour tree layout, and the landscape construction. We provide all necessary implementation details, and show an example application of our technique to the real-world data.

We note that our technique can be applied only to moderate size data, due to the potential explosion of the number of points/edges during the iterative edge routing process. We plan to address this issue in the future work by excluding the merge saddles from the projection stage, thus adding flexibility in their placement.

Another issue that can we plan to address is calculation of the gap distance during the offsetting. Currently, the algorithm roughly calculates it as the shortest distance between any two points, divided

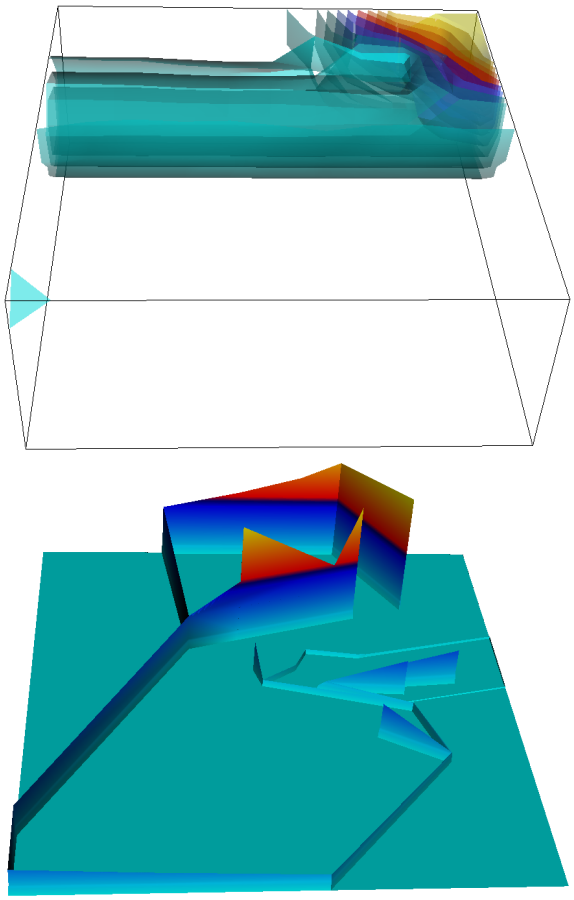
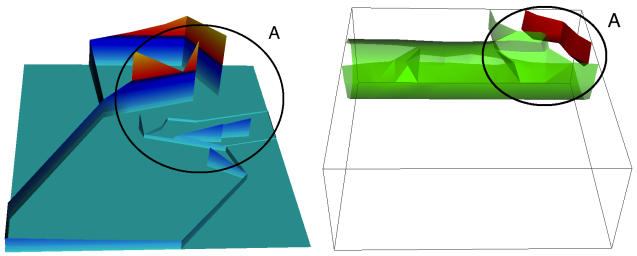
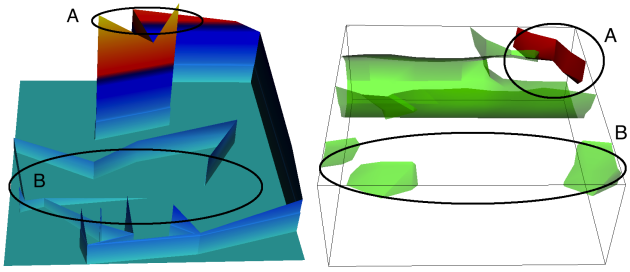


Figure 5: *Trilinear ray sampling option data set. Volume rendering (top) and the corresponding geometry-preserving topological landscape (bottom).*



(a) *Poor performing configurations are closely located (circle A).*



(b) *Poor performing configurations are spread around (circles A, B).*

Figure 6: *Geometry-preserving landscapes of (a) trilinear ray sampling option data set, (b) nearest-neighbor ray sampling option data set. In landscapes, we can easily see how close/far the poor performing configurations are from each other. To derive the same observation from the direct view, we have to search for specific, often different, isosurfaces that would encompass such configurations. Furthermore, this search might be impossible, if the dimensionality of the function exceeds three.*

by the number of saddles (i.e., potential contours). A more precise approximation is possible, given that routing via the Voronoi diagram maintains the half-distance to other elements.

Finally, we consider several interesting directions for the future work. One direction is to expand the current landscape by embedding more information. For example, we can associate the area, bounded by contours, with an alternative importance metric (e.g., branch volumes, similar to the original method), while adjusting the gap to increase/decrease the area as necessary. The landscape colors can also be used for similar purposes. Another direction is to use the information about the cancellation of critical points from the Morse-Smale complex in the landscape construction, resulting in a landscape that would have a correct Morse-Smale complex, a feature that other landscape techniques lack.

Acknowledgements

The authors would like to thank Tim Dwyer from Microsoft Research for valuable input. The authors would like to thank the reviewers for helping us to improve the paper, as well as extremely interesting ideas for future work. The authors are grateful to Kanat Baigarin and Olzhas Makhambetov for their continuous support. Performance data is a courtesy of E. Wes Bethel, LBNL. This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. DOE under Contract No. DE-AC02-05CH11231 (Lawrence Berkeley National Laboratory) through the grants "Towards Exascale: High Performance Visualization and Analytics" and "Topology-based Visualization and Analysis of High-dimensional Data and Time-varying Data at the Extreme Scale", and the Program 055 "Fundamental and applied scientific research" of the Ministry of Education and Science of the Republic of Kazakhstan under the contract with the Center for Energy Research, Nazarbayev University.

References

BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. 1998. Visualization of scalar topology for structural enhancement. In *Proc. IEEE Visualization*, IEEE Computer Society Press, D. Ebert, H. Hagen, and H. Rushmeier, Eds., IEEE, 51–58.

BETHEL, E. W., AND HOWISON, M. 2012. Multi-core and many-core shared-memory parallel raycasting volume rendering optimization and tuning. *International Journal of High Performance Computing Applications*. `http://dx.doi.org/10.1177/1094342012440466`.

BREMER, P.-T., WEBER, G. H., TIERNY, J., PASCUCCI, V., DAY, M. S., AND BELL, J. B. 2009. A topological framework for the interactive exploration of large scale turbulent combustion. In *Proc. 5th IEEE International Conference on e-Science*, IEEE, 247–254.

CARR, H., SNOEYINK, J., AND AXEN, U. 2003. Computing contour trees in all dimensions. *Comput. Geom.—Theory and Apps 24*, 2, 75–94.

CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

DWYER, T., AND NACHMANSON, L. 2009. Fast edge-routing for large graphs. In *Proc. of 17th International Conference on Graph Drawing*, 147–158.

ENGEL, D., ROSENBAUM, R., HAMANN, B., AND HAGEN, H. 2011. Structural decomposition tree. *Computer Graphics Forum 30*, 3, 921–930.

FODOR, I. 2002. A survey of dimension reduction techniques. Tech. rep., LLNL.

GANSNER, E. R., AND NORTH, S. C. 1999. An open graph visualization system and its applications to software engineering. *Software—Practice and Experience 30*, 1203–1233.

GARTH, C., TRICOCHE, X., AND SCHEUERMANN, G. 2004. Tracking of vector field singularities in unstructured 3d time-dependent data sets. In *Proc. of IEEE Visualization*, 329–336.

HARVEY, W., AND WANG, Y. 2010. Topological landscape ensembles for visualization of scalar-valued functions. *Computer Graphics Forum 29*, 3, 993—1002.

HEINE, C., SCHNEIDER, D., CARR, H., AND SCHEUERMANN, G. 2011. Drawing contour trees in the plane. *IEEE Trans. Vis. Comput. Graph. 17*, 11, 1599–1611.

MILNOR, J. W. 1963. *Morse Theory*. Princeton University Press, Princeton, New Jersey, May.

MIZUTA, S., SUWA, Y., ONO, T., AND MATSUDA, T. 2004. Description of the topological structure of digital images by contour tree and its application. Tech. rep., Institute of Electronics, Information and Communication Engineers.

MIZUTA, S., ONO, T., AND MATSUDA, T. 2006. Contour nest: A two-dimensional representation for three-dimensional isosurfaces. In *Proc. Volume Graphics*, 67–70.

OESTERLING, P., HEINE, C., JÄNICKE, H., AND SCHEUERMANN, G. 2010. Visual analysis of high dimensional point clouds using topological landscapes. In *Proc. IEEE Pacific Visualization 2010 Symposium*, 113–120.

OESTERLING, P., SCHEUERMANN, G., TERESNIAK, S., HEYER, G., KOCH, S., ERTL, T., AND WEBER, G. 2010. Two-stage framework for a topology-based projection and visualization of classified document collections. In *Proc. IEEE Symposium on Visual Analytics Science and Technology*.

OESTERLING, P., HEINE, C., JÄNICKE, H., SCHEUERMANN, G., AND HEYER, G. 2011. Visualization of high-dimensional point clouds using their density distribution's topology. *IEEE Trans. Vis. Comput. Graph. 17*, 11, 1547–1559.

PACH, J., AND WENGER, R. 2001. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics 17*, 717–728.

PASCUCCI, V., COLE-MCLAUGHLIN, K., AND SCORZELLI, G. 2005. Multi-resolution computation and presentation of contour trees. Tech. rep., LLNL.

TAMASSIA, R. 2012. *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, Aug.

TORGERSON, W. S. 1952. Multidimensional Scaling: I. Theory and Method. *Psychometrika 17*, 401–419.

VAN KREVELD, M. J., VAN OOSTRUM, R., BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. 1997. Contour trees and small seed sets for isosurface traversal. In *Symposium on Computational Geometry*, 212–220.

WEBER, G. H., BREMER, P.-T., AND PASCUCCI, V. 2007. Topological landscapes: A terrain metaphor for scientific data. *IEEE Trans. Vis. Comput. Graph. 13*, 6, 1416–1423.

WEBER, G. H., DILLARD, S. E., CARR, H., PASCUCCI, V., AND HAMANN, B. 2007. Topology-controlled volume rendering. *IEEE Trans. Vis. Comput. Graph. 13*, 2, 330–341.