Direct Feature Visualization Using Morse-Smale Complexes

Attila Gyulassy, Natallia Kotava, Mark Kim, Charles Hansen, *Senior Member*, *IEEE*, Hans Hagen, *Member*, *IEEE*, and Valerio Pascucci, *Member*, *IEEE*

Abstract—In this paper, we characterize the range of features that can be extracted from an Morse-Smale complex and describe a unified query language to extract them. We provide a visual dictionary to guide users when defining features in terms of these queries. We demonstrate our topology-rich visualization pipeline in a tool that interactively queries the MS complex to extract features at multiple resolutions, assigns rendering attributes, and combines traditional volume visualization with the extracted features. The flexibility and power of this approach is illustrated with examples showing novel features.

Index Terms—Volume visualization, applications, feature detection, topology.

1 Introduction

Visualization of scalar valued volumetric data is a well-studied domain. Traditionally, the underlying technology of these techniques is a mapping from a combination of the scalar value and associated local gradient information to a color and opacity. Some compositing of samples, as in ray casting or slice-based rendering, results in a final image. Such techniques have been successful in producing high-quality images, however, the growing size and complexity of data sets are encouraging development of techniques that incorporate automated analysis. Indeed in many fields, techniques are custom designed to identify and visualize features specific to data from a single application domain. Topology-based analysis methods are an attractive alternative and are especially well suited in this context, since they robustly describe a general feature space that can be queried combinatorially for reproducible and consistent results. The Morse-Smale (MS) complex is a topological representation of a scalar function with characteristics that make it particularly useful in identifying features: the various cells of the complex form the basis of a large feature space; the complex can be simplified to represent the function at multiple scales, for example, to be used in noise removal; and simple and combinatorial algorithms for its computation ensure robustness in the analysis. In particular, recent advances in algorithms for the computation of the MS complex utilizing discrete Morse theory enable, in practice, the analysis of increasingly large and complex data. However, visualization techniques have not yet explored the full potential of this technology. Topological feature identification has traditionally been done by experts on a per-application basis, with custom code being developed to visualize the results of each.

Several challenges, until now, have restricted the use of the MS complex for analysis and visualization. First, there has been no consistent framework for querying the MS complex to extract the different features required in different application areas. We present a model of the MS complex as a general graph structure and describe a query language designed to enable generic feature extraction. Second, the field of topology-based visualization is perceived to be complex, theoretical, and nonintuitive. We address this by presenting a visual guide to feature extraction using queries on the MS complex. Third, to the best of our knowledge, the 2-manifold and 3-manifold components of the MS complex have never been used for visualization. We show how they can be used to create compelling images. We present an algorithm and efficient data structures to compute these geometries in a simplification hierarchy, in a manner that allows interactive random access to different levels in the hierarchy. We implement these elements in a hybrid visualization system combining traditional techniques and direct feature visualization. Our system allows interactive specification and exploration of the MS complex feature space with flexible queries using the familiar look and feel of a scene graph. Using our system, we present novel visualizations, such as in Fig. 1, that illustrate never-before-seen features in well-known data sets.

Contributions. We present the following contributions:

- a generic language for querying MS complexes, designed to identify a broad range of application-specific features;
- a visual guide for helping understand the content of a topology-rich visualization, which aids the user in designing queries;
- algorithms and data structures to maintain the geometry of manifolds of different dimensions in a

- *A. Gyulassy, M. Kim, C. Hansen, and V. Pascucci are with the Scientific Computing and Imaging Institute and School of Computing, Department of Computer Science, University of Utah, 72 South Central Campus Drive, WEB 3750, Salt Lake City, Utah 84112.*
  E-mail: {jediati, mkim, pascucci}@sci.utah.edu, hansen@cs.utah.edu.
- *N. Kotava and H. Hagen are with the Department of Computer Science, Technical University of Kaiserslautern, Postfach 3049, Kaiserslautern 67653, Germany.*
  E-mail: kotava@rhrk.uni-kl.de, hagen@informatik.uni-kl.de.

Manuscript received 1 Sept. 2010; revised 15 July 2011; accepted 17 Oct. 2011; published online 26 Oct. 2011.
Recommended for acceptance by T. Möller.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2010-09-0208.
Digital Object Identifier no. 10.1109/TVCG.2011.272.

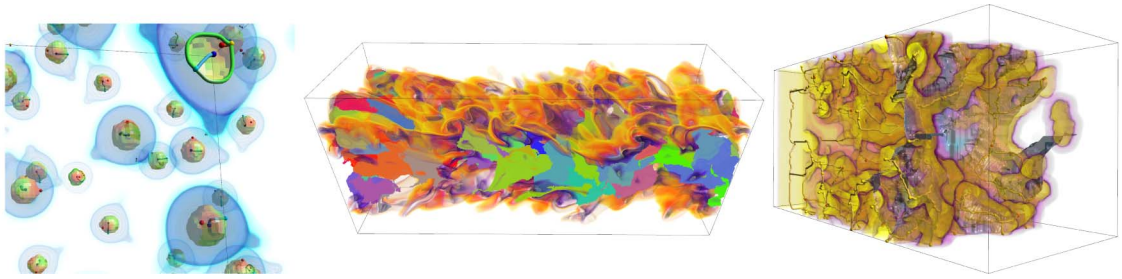


Fig. 1. Topology-based techniques can extract features that are hard to detect with traditional methods. Combinatorial computation of topological invariants results in robust identification of features, even in degenerate cases, such as topological pouches (left). Using the machinery of topological persistence and simplification, we can visualize the 3-manifolds of the MS complex forming flow basins in a manner oblivious to noise (middle). Derived structures, for example, separating surfaces, can be used to represent nonphysical phenomena, such as the "outer surface" of a sponge-like material (right).

simplification hierarchy, supporting interactive random access between levels of the hierarchy;
- specifications for an interactive rendering system that handles concurrent visualization of features of diverse dimension, supporting locally controlled attributes; and
- examples showing the practical impact of our system: for the first time, extracting and visualizing features previously not possible.

2 Related Work

Modern volume rendering techniques have been used for the past 30 years. Kajiya and Von Herzen [1] used ray tracing to render objects represented by densities within a volume grid, based on light scattering equations. Drebin et al. [2] and Levoy [3] laid out the foundation for volume rendering, such as compositing methods, gradient-based shading, and volume classification. Max [4] discussed several different models for light interaction with volume densities, including absorption, emission, reflection, and scattering.

Volume classification provides the means for visually segmenting volumetric data. Kindlmann and Durkin [5] proposed the histogram volume, which captures the relationship between volumetric quantities in a position independent, computationally efficient fashion. They presented semiautomatic methods of generating transfer functions for direct volume rendering. Kniss et al. [6] presented multidimensional transfer functions for interactive volume rendering. Correa and Ma [7], [8] proposed visibility-driven and size-based transfer function design techniques for volume exploration. While transfer function design with multidimensional histograms can discriminate between features (materials) in the data it requires a strong background knowledge about the data and is time consuming. Methods have been proposed to accelerate the transfer function design process. Rezk Salama et al. [9] introduced an additional level of abstraction for parametric models of transfer functions, using semantic models for transfer function design. Tzeng et al. [10] proposed an approach to the volume classification problem that couples machine learning and a painting metaphor to allow more sophisticated classification in an intuitive manner.

Query-driven visualization techniques provide a mechanism for extracting relevant features, defined by the query, from data sets. Such queries can be range queries on scalar or vector fields in the data or can be queries against visualization constructs such as isosurfaces. McCormick et al. [11] described SCOUT which allows a user to define range queries against their data using a data parallel language to form the queries. Stockinger et al. [12] introduced DEX which uses bitmap indexing to efficiently answer multivariate, multidimensional data queries to provide input to a visualization pipeline.

Topology-based techniques are well known in the context of scalar function analysis. In general, a topological representation of the function is computed, then queried. Reeb graphs [13], contour trees, and their variants have been used successfully to guide the removal of topological features [14], [15], [16], [17], [18], [19], [20]. Pascucci et al. [21] showed how the Reeb graph can be constructed in a streaming manner for large data sets. Tierny et al. [22] presented an efficient algorithm to compute Reeb graphs by cutting the domain, and showed how it could be used for isosurface simplification. Weber et al. [23] used contour trees to segment the domain and render each region with a separate transfer function. Chiang and Lu used the augmented contour tree to simplify tetrahedral meshes while preserving isosurface topology [24].

Partitions of surfaces induced by a piecewise-linear (PL) function have been studied in different fields, under different names, motivated by the need for an efficient data structure to store surface features. Cayley [25] and Maxwell [26] proposed a subdivision of surfaces using peaks, pits, and saddles along with curves between them. The development of various data structures for representing topographical features was discussed by Rana [27].

The MS complex is a topological data structure that provides an abstract representation of the gradient flow behavior of a scalar field [28], [29]. Several algorithms have been proposed to compute MS complexes in practice: Edelsbrunner et al. [30] presented the first algorithm for two-dimensional data, and Bremer et al. [31] improved this by following gradients more faithfully and described a multiresolution representation of the scalar field. Although an algorithm was proposed to compute all dimensional manifolds of a three-dimensional complex [32], a practical implementation was never done due to the complexity of the algorithm. The one-skeleton (0- and 1-manifolds) of the MS complex was first computed successfully for volumetric data by Gyulassy et al. [33]. Although the same authors presented a more efficient approach to computing the MS complex by using a sweeping plane [34], data size and computational overhead still proved to be a limiting factor.

Discrete Morse theory, as presented by Forman [35], is an attractive alternative to PL Morse theory, since it simplifies the search for higher dimensional manifolds by discretizing the flow operator. The challenge in using a discrete approach is in generating a discrete gradient field. Lewiner et al. [36] showed how a discrete gradient field can be constructed and used to identify the MS complex. However, this construction required an explicit graph-based representation of gradient paths, thus, was prohibitively expensive for large volumetric data. King et al. [37] presented a method for constructing a discrete gradient field that agrees with the large-scale flow behavior of the data defined at vertices of the input mesh. In our approach, we use the algorithm presented by Gyulassy et at. [38] for its simplicity of implementation and its dynamic simulation of simplicity, that greatly reduces the number of zero-persistence critical points found.

These techniques have begun to make an impact in analysis of scientific data. Laney et al. [39] used the descending 2-manifolds of a two-dimensional MS complex to segment an interface surface and count bubbles in a simulated Rayleigh-Taylor instability. Bremer et al. [40] used a similar technique to count the number of burning regions in a lean premixed hydrogen flame simulation. Gyulassy et al. [41] used carefully selected arcs from the 1-skeleton of the three-dimensional MS complex to analyze the core structure of a porous solid. In each of these examples, feature definitions were attained by exploring thresholds, levels in a hierarchy, and different components of the MS complex, clearly illustrating the need for interactive feature exploration. Although each result was obtained from different codes, they all use the same underlying mathematical theory, motivating a unified query system for topological structures.

3 Theoretical Background

Scalar valued volumetric data are most often available as discrete samples at the vertices of an underlying mesh. Morse theory has been well studied in the context of smooth scalar functions, and has been adapted for such discrete domains. We first present some basic definitions from smooth Morse theory, and then present the discrete analogue.

3.1 Morse Functions and the MS Complex

Let f be a real-valued smooth map $f : \mathbf{M} \to \mathbf{R}$ defined over a compact d-manifold $\mathbf{M}$. A point $p \in \mathbf{M}$ is critical when $|\nabla f(p)| = 0$, i.e., the gradient is zero, and is nondegenerate when its Hessian (matrix of second partial derivatives) is nonsingular. The function f is a *Morse function* if all its critical points are nondegenerate and no two critical points have the same function value. The *Morse Lemma* states that there exists local coordiantes around p such that f has the following *standard form*: $f_p = \pm x_1^2 \pm x_2^2 \cdots \pm x_d^2$. The number of minus signs in this equation gives the *index* of critical point p. In three-dimensional functions, minima are index-0, 1-saddles are index-1, 2-saddles are index-2, and maxima are index-3.

An integral line in f is a path in $\mathbf{M}$ whose tangent vector agrees with the gradient of f at each point along the path. Each integral line has an origin and destination at critical points of f. *Ascending* and *descending* manifolds are obtained as clusters of integral lines having common origin and destination, respectively. The descending manifolds of f form a cell complex that partitions $\mathbf{M}$; this partition is called the *Morse* complex. Similarly, the ascending manifolds also partition $\mathbf{M}$ in a cell complex. A Morse function f is a *Morse-Smale function* if ascending and descending manifolds of its critical points only intersect transversally. An index-i critical point has an i-dimensional descending manifold and a $(d-i)$-dimensional ascending manifold. The intersection of the ascending and descending manifolds of a Morse-Smale function forms the *Morse-Smale complex*. The critical points are called *nodes* and the one-dimensional cells of the complex connecting them are called *arcs*. We refer the reader to Section 5 for a visual description of these concepts.

3.2 Discrete Morse Theory

Practical algorithms for computing MS complexes for volumetric data rely on discretizing the domain. For example, Gyulassy et al. [38] showed how discrete Morse theory could be applied to sampled data. The following basic definitions from discrete Morse theory are due to Forman [35], and we refer the reader to this work for an intuitive description. A d-cell is a topological space that is homeomorphic to a euclidean d-ball $B^d = \{x \in \mathbb{E}^d : |x| \leq 1\}$. A *chain* is a formal sum of elements. For cells α and β, $\alpha < \beta$ means that α is a *face* of β and β is a *coface* of α, i.e., the vertices of α are a proper subset of the vertices of β. If $dim(\alpha) = dim(\beta) - 1$, we say α is a *facet* of β, and β is a *cofacet* of α. If a cell α has dimension d, we denote it as $\alpha^{(d)}$.

The *boundary operator* ∂ maps a cell to its facets, and induces an orientation on the facets. Formally, $\partial\beta = \sum_{\alpha<\beta}\langle\partial\beta, \alpha\rangle\alpha$. The *inner product* of chains c_i and c_j, denoted $\langle c_i, c_j\rangle$, is equal to $\sum_{\alpha\in c_i}\sum_{\beta\in c_j}\alpha\cdot\beta$, where $\alpha\cdot\beta$ is one if $\alpha = \beta$ and the orientation of α is the same as the orientation of β, minus one if the orientations do not agree, and zero if $\alpha \neq \beta$.

Let K be a regular complex that is a mesh representation of $\mathbf{M}$. A function $f : K \to \mathbf{R}$ that assigns scalar values to every cell of K is a *discrete Morse function* if for every $\alpha^{(d)} \in K$, its number of cofacets $|\{\beta^{(d+1)} > \alpha | f(\beta) \leq f(\alpha)\}| \leq 1$, and its number of facets $|\{\gamma^{(d-1)} < \alpha | f(\gamma) \geq f(\alpha)\}| \leq 1$. A cell $\alpha^{(d)}$ is critical if its number of cofacets $|\{\beta^{(d+1)} > \alpha | f(\beta) \leq f(\alpha)\}| = 0$ and its number of facets $|\{\gamma^{(d-1)} < \alpha | f(\gamma) \geq f(\alpha)\}| = 0$, and has index equal d.

A *vector* in the discrete sense is a pair of cells $\{\alpha^{(d)} < \beta^{(d+1)}\}$, where we say that an arrow points from $\alpha^{(d)}$ to $\beta^{(d+1)}$. Intuitively, this vector simulates a direction of flow. A *discrete vector field* V on K is a collection of pairs $\{\alpha^{(d)} < \beta^{(d+1)}\}$ of cells of K such that each cell is in at most one pair of V. A critical cell is unpaired. Given a discrete vector field V on K, a *V-path* is a sequence of cells

$$\alpha_0^{(d)}, \beta_0^{(d+1)}, \alpha_1^{(d)}, \beta_1^{(d+1)}, \alpha_2^{(d)}, \ldots, \beta_r^{(d+1)}, \alpha_{r+1}^{(d)},$$

such that for each $i = 0, \ldots, r$, the pair $\{\alpha_i^{(d)} < \beta_i^{(d+1)}\} \in V$, and $\{\beta_i^{(d+1)} > \alpha_{i+1}^{(d)} \neq \alpha_i^{(d)}\}$. A V-path is the discrete equivalent of a streamline in a smooth vector field. A discrete vector field in which all V-paths are monotonic and do not contain any loops is a *discrete gradient field* of a discrete Morse function. We use the algorithm presented by Gyulassy et al. [38] to compute a discrete gradient field from sampled data.

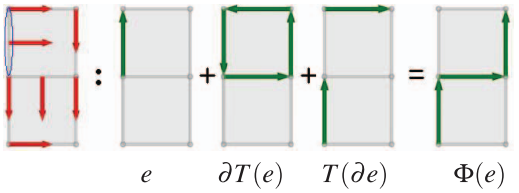


Fig. 2. Discrete gradient arrows (red) pair cell with their facets. We apply the discrete flow operator to the circled edge $e : \Phi(e) = e + \partial T(e) + T(\partial e)$. An orientation (green arrow) is chosen for e, and it induces an opposite orientation on the 2-cell that is the head of the gradient arrow starting at e. It also induces opposite orientations on the edges that are the head of gradient arrows beginning at the facets of e. Cells summed with opposite orientations cancel one another. The result of the flow on e is three edges in a chain.

The *discrete tangent operator*, $T(\alpha)$ maps the cell at the tail of a gradient arrow to the head of the gradient arrow. Formally, $T(\alpha) = -\langle \partial\beta, \alpha \rangle \beta$, where α is the tail of a gradient arrow, and β is the head of a gradient arrow. If α is not the tail of a gradient arrow, then $T(\alpha) = 0$. The discrete flow operator combines the tangent and boundary operators to simulate advecting a cell in the gradient "flow" direction, as is illustrated by Fig. 2. The *flow operator* Φ is defined by $\Phi(\alpha) = \alpha + \partial T(\alpha) + T(\partial\alpha)$. The flow operator, and inverse flow operator are the discrete analogue of an integration step in computing a streamline, allowing us to compute ascending and descending manifolds of critical points. Note that unlike integral lines, paths found using the flow operator can split and merge. Formally, the descending manifold D_α of a critical cell α is the smallest invariant chain under the flow operator containing α: $\Phi(D_\alpha) = D_\alpha$. The ascending manifold A_α of a critical cell α is the smallest invariant chain under the inverse flow operator: $\Phi^{-1}(A_\alpha) = A_\alpha$. Both the ascending and descending manifolds are sets of cells, and in Section 6.2, we discuss how to construct a geometric realization of these structures for visualization. Fig. 4 illustrates the discrete version of critical cells, arcs, and ascending/descending 2- and 3-manifolds.

3.3 Persistence-Based Simplification

A function f is simplified by repeated cancellation of pairs of critical points connected by an arc in the MS complex. The local change in the MS complex indicates a smoothing of the gradient vector field and hence of the function f. Forman [42] showed how a cancellation could be achieved in a discrete gradient field by reversing the gradient path between two critical cells. Gyulassy et al. [38] provided a full characterization of cancellation operations in terms of how they affect the connectivity of the complex and the geometry of the ascending/descending manifolds, operating solely on the combinatorial structure of the complex. Repeated application of cancellations in order of *persistence*, the absolute difference in function value of the canceled critical points, results in a hierarchy of MS complexes and a multiresolution representation of features.

4 Introducing Generic Queries

The MS complex is a representation of the function in terms of its flow properties, and the full complex is the basis for a large feature space; the problem of extracting features is reduced to querying this structure. In our model, the result of querying the MS complex is a set of nodes and a set of arcs. Since higher dimensional manifolds are each associated with one node, extracting them is reduced to the problem of selecting the nodes that generate them. For example, to extract surfaces separating user-selected basins, a query would be designed to select the 1-saddles separating minima, then ascending 2-manifold geometry would be computed to recover the surfaces.

In this model, interactive exploration of features uses the following pipeline: a base MS complex and hierarchy are computed, and then queries on this structure are evaluated by consumer objects. A consumer object can be a renderer, a histogram generator, or any other visualization or analysis tool. One of the key motivations for our functional programming design of queries is the need for interactive exploration, where any parameter can be changed in any part of a visualization pipeline.

4.1 Query Object and Function Types

Table 1 specifies objects and query functions for generic feature extraction. The *objects* describe a generic data structure, in which an MS complex is represented as a set of nodes and a set of arcs. Each node contains a list of arc references that have it as an endpoint, and each arc references the nodes at its upper and lower endpoints. Both arcs and nodes have attribute objects, collections of scalars. In our examples, some of the attributes we query for nodes are index of criticality, function value, and coordinates, although this system could easily be extended to store ascending/descending manifold surface area or volume, count of incident arcs, proximity of other critical points, or other user-specified measures. At the most basic level, an MS complex is represented as an undirected graph where each node and arc has a set of associated values.

A query starts with a precomputed MS complex object. A hierarchy object can be computed by repeated cancellation of critical point pairs of an MS complex. A *persistence selector* operates on a hierarchy and returns the MS complex simplified to an input scalar value. The data structures necessary to extract geometry from the simplified complex are covered in Section 6.3. The set of nodes and arcs for a particular MS complex are returned by *node extractor* and *arc extractor* objects, respectively.

We also specify *function objects* that operate on elements of an MS complex. These objects are used to "navigate" on the complex as well as filter desired features. Features are often defined as objects that satisfy certain conditions, and the *node selector* and *arc selector* function objects enable this functionality. These filter functions take as input a *test*, a Boolean function object that decides whether or not a node or arc is to be included in the output of the selector. The test performs a comparison with the value returned by a *value extractor* object applied to a node or arc. The value extractor returns one of the attributes of the node or arc. For example, to extract the minima of a particular MS complex, a node selector is applied to the node extractor, with a test function comparing the index of the node (returned by a value extractor) to zero.

Navigation on the MS complex is equally important in selecting features. These functions correspond to a "walk" on

TABLE 1
A Functional Description of Objects and Selectors for Querying an MS Complex

Objects		
object	*type*	*name*: *description*
$v ::= (x_0, x_1, \ldots, x_k)$	V	*attribute*: a tuple of values, or functions that evaluate to a value *e.g.*, function value, index, persistence, statistics, arc count, etc.
$n ::= (v, *\ell_a)$	N	node: a pair with an attribute and a reference to a set of incident arcs
$a ::= (v, *n, *n)$	A	arc: a three-tuple of an attribute and two node references
$\ell_n ::= \{*n_0, *n_1, \ldots, *n_{m-1}\}$	L_N	node set: a set of m node references
$\ell_a ::= \{*a_0, *a_1, \ldots, *a_{m-1}\}$	L_A	arc set: a set of m arc references
$msc ::= (*\ell_n, *\ell_a)$	MSC	MS complex: a pair containing an arc set reference and a node set reference
$h ::= (*msc_0, *msc_1, \ldots, *msc_n)$	H	hierarchy: a sequence of MS complexes where msc_{i+1} is obtained by canceling an arc in msc_i

Function Objects		
type	*pseudocode*	*description*
$P : \mathbb{R} \times H \to MSC$	$p(v,h) = h.cancel\_to(v)$	*persistence selector*: returns the MS complex simplified to persistence v
$E_N : MSC \to L_N$	$e_N(m) = m.\ell_n$	*node extractor*: returns the set of nodes in an MS complex
$E_A : MSC \to L_A$	$e_A(m) = m.\ell_a$	*arc extractor*: returns the set of arcs in an MS complex
$F : v \to \mathbb{R}$	$f(v,i) = v.x_i$	*value extractor*: returns a value within an attribute
$T : F \times \mathbb{R} \to \{true, false\}$	$t(f,c) = compare(x,c)$	*test*: compares the result of a value extractor to a constant
$S_a : L_a \times T \to L_a$	$s_a(\ell_a, t) = gather(map(t, \ell_a))$	an *arc selector* returns the subset of its input arcs that pass the test
$S_n : L_n \times T \to L_n$	$s_n(\ell_n, t) = gather(map(t, \ell_n))$	a *node selector* returns the subset of its input nodes that pass the test
$I_n : L_a \to L_n$	$i_n(\ell_a) = \bigcup_{*a_j \in \ell_a} \{a_j.n_0, a_j.n_1\}$	an *incident nodes selector* returns the set of nodes incident to the input arcs
$I_a : L_n \to L_a$	$i_a(\ell_n) = \bigcup_{*n_j \in \ell_n} n_j.\ell_a$	an *incident arcs selector* returns the set of arcs incident to the input nodes

The objects describe generic data structures, and the selector functions extract sets of arcs and nodes.

the graph described by the nodes and arcs. The *incident arcs* function object applied to a set of nodes returns the set of arcs touching each node in the set; similarly, the *incident nodes* returns the set of nodes that are endpoints of the input set of arcs. Special variants of these, for example, an *upper incident arcs* and *upper incident nodes* function, denoted $i_a^+()$ and $i_n^+()$, respectively, combine an index test with the incidence operator to return the set in the "upwards" direction. One can then define an *ascending boundary selector* object to find all the critical points on the boundary of the ascending manifold of a critical point by repeatedly applying the upper incident nodes and arcs function. Therefore, the ascending boundary selector computes the critical points on the boundary of a basin of a minimum by applying the upper incident nodes selector to the upper incident arcs selector repeatedly, to return the set of 1-saddles connected to the minimum, the set of 2-saddles connected to those, and then the set of maxima connected to the 2-saddles.

4.2 Consumers

A consumer object evaluates a query and converts the resulting list of arcs or nodes into another format. Different kinds of consumers are: geometry processors, renderers, and statistics generators. A consumer may rely on the input gradient field and any computed attributes to do its work. For example, a geometry extractor, described in Section 6, may take as input a query that returns a set of saddles, and outputs the set of ascending and descending 2-manifold surfaces. A render object typically is at the end of the visualization pipeline, and can display a set of nodes, a set of arcs, a set of geometry extractors, the scalar function value volume, and an index volume, and render the scene as described in Section 7.3. A statistics generator could output a histogram of an attribute value for a set of arcs and nodes. The consumer objects are not listed in Table 1, since they are highly application dependent. For example, in our implementation, we are interested in generic visualization of extracted features, and therefore have implemented geometry extractors and a renderer. However, one can envision a usage scenario where an application scientist wants to perform custom statistics on a set of features extracted at multiple time steps of a simulation. In this case, the implementation of a statistics tool consumer is left up to the user.

5 A Visual Guide to Features

When performing MS-complex-based analysis and visualization, features are identified as combinations of properly queried nodes and arcs and their corresponding manifold geometry. One of the more challenging tasks in using topology-based analysis is to translate a scientist's intuitive description of an application-specific feature into formal queries on the MS complex. This task is made particularly difficult because it requires the involvement of a user, who is often a domain expert not knowledgeable in topological analysis. When formalizing a feature definition, parameters may be explored in a trial and error process, where interactive evaluation of the output of a given query is key in closing the loop and converging to satisfactory feature definitions.

We provide a simple visual guide to facilitate the interpretation of the topology-rich visualizations that are generated as a result of any given query. Fig. 3 presents as a simple table the key concepts needed for understanding a visual topological model in 3D. Included with each image is an intuitive explanation. The queries used to extract the illustrated features can be found in Table 2 in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.272. The function we use in these examples is a simple sum of two Gaussians centered at two evenly spaced points in a box-shaped domain (Fig. 3 (Morse Function)). Some isosurfaces are shown, with the isovalue of the outer surface being the lowest.

The top row in this guide shows the basic elements of a Morse and Morse-Smale complex, and illustrates the full set of ascending and descending manifolds of this function

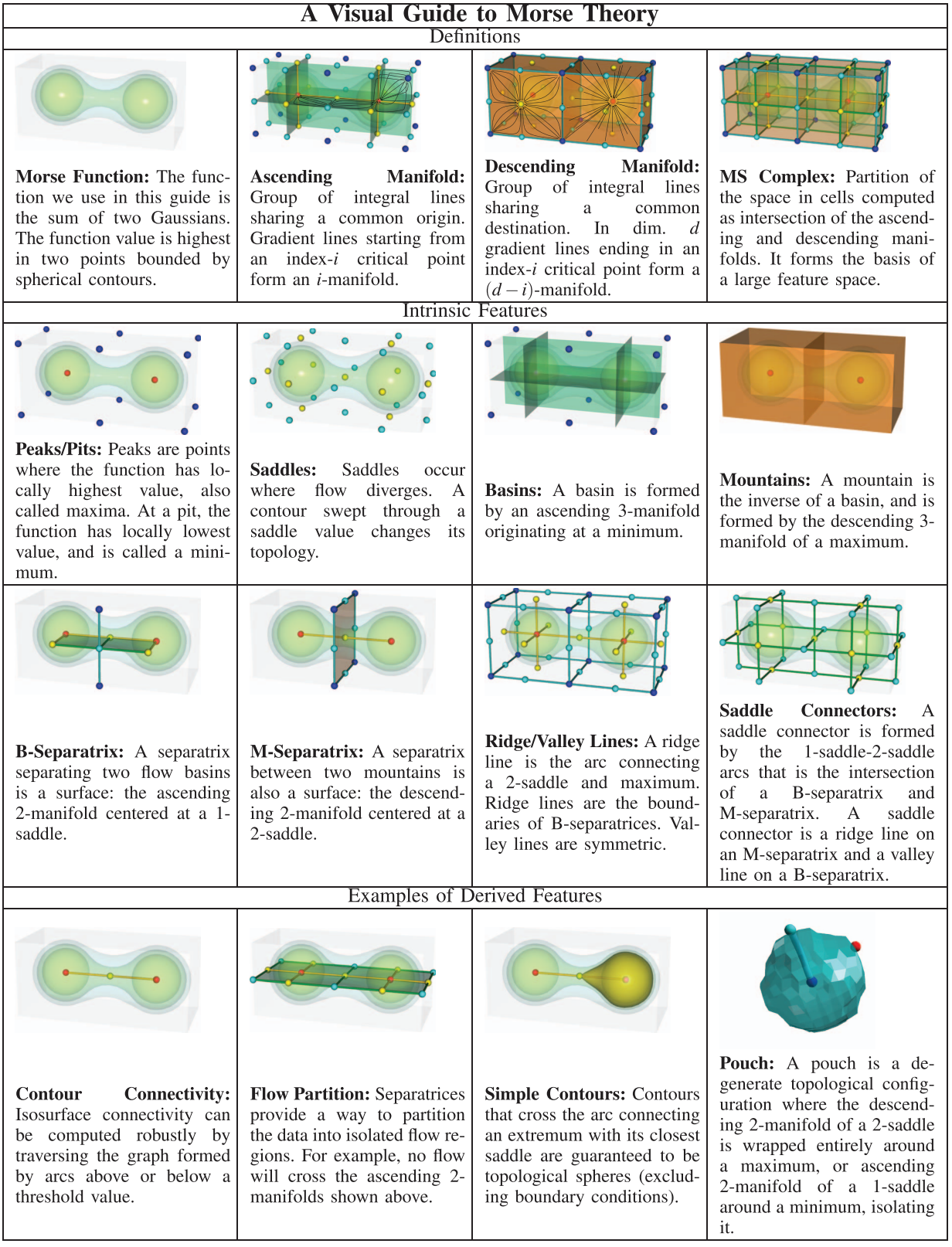


Fig. 3. Key topological definition and intrinsic/derived structures used in data analysis based on Morse theory. The intrinsic features highlighted above are color coded as follows: *blue sphere* = minimum; *cyan sphere* = 1-saddle; *yellow sphere* = 2-saddle; *red sphere* = maxim; *cyan tube* = 1-saddle-minimum arc; *green tube* = 1-saddle-2-saddle arc; *orange tube* = 2-saddle-maximum arc; *orange surface* = descending 2-manifold; *cyan surface* = ascending 2-manifold.

along with the critical points that generate them. We annotate the ascending and descending manifolds with hand-drawn integral lines to indicate gradient flow direction.

The Morse cells partition the domain of a Morse function, and its nodes, arcs, surface, and volumetric cells are the intrinsic topological features that are used as basic

building blocks in topological analysis of scientific data. Isolating some of these elements individually, one can start to extract more interesting features: these are shown in the second and third row in Fig. 3.

The last row of Fig. 3 shows a few important cases of structures that can be derived from the intrinsic topological elements. Here, the filtering and navigation operations are more sophisticated, and in some cases additional value extractors need to be defined.

In the first one of these examples, we show how to count the number of contours at a particular isovalue. We extract exactly those 2-saddle-maximum arcs and nodes that are entirely above this value. Counting the number of connected components in this graph gives the number of contours. Similar to the contour tree, the MS complex enables this computation without needing to compute *any* isosurfaces explicitly.

In the second example, we show a surface that partitions the flow into two regions: integral lines that originate from minima in the upper half and those that originate from minima in the lower half. Intersecting the set of 1-saddles adjacent to each group gives those that have one endpoint in the upper, and one endpoint in the lower halves. The ascending 2-manifolds of these 1-saddles partition the flow.

Next, to compute simple contours (homeomorphic to a 2-sphere), we identify the highest 2-saddle from a maximum, and compute an isosurface between that value and the maximum's value, restricted to the maximum's descending 3-manifold. Finally, we show that our query system is robust enough to identify topological degeneracies, such as a pouch. These structures are present when a 1-saddle has no incident arcs connecting to a 2-saddle, or vice versa. Notice that the number of derived topological concepts is virtually unlimited and the development of a general language for topological queries is essential to allow the user unrestricted exploration of the feature space.

6 Computing Feature Geometry

Visualization of topological elements requires extracting renderable geometry from a discrete gradient field. In addition to the geometric realization of the nodes and arcs, we extract ascending and descending manifolds, and render them as a collection of geometric primitives. Computing ascending and descending manifolds in the context of discrete Morse theory has been well studied: first Cazals et al. [43] computed them from a tree-based representation of the gradient, then Gyulassy et al. [38] gave a description of how to compute them by searching in the discrete gradient field. Neither of these techniques take into account the need for random access into a hierarchy of MS complexes. In previous implementations [43], simplification is achieved by repeated reversal of gradient paths, and therefore random access between levels of a hierarchy would require sequential reversal of paths and expensive recomputation of the manifolds in the modified gradient. The data structures and algorithms we present in this section do not require modification of the input gradient field, and avoid recomputation of manifolds during interactive exploration. First, we review algorithms to compute ascending and descending manifolds. Next, we show how a manifold is translated to renderable primitives. Finally, we present a data structure and technique for maintaining manifold geometry efficiently in a simplification hierarchy.

6.1 Computing Ascending/Descending Manifolds

We restate the algorithm described by Gyulassy et al. [38] for computing ascending and descending manifolds. The following algorithm collects the d-cells in the descending manifold of an index-d critical cell β by repeated application of the flow operator defined in Section 3.2. The following pseudocode implements this operator, isTail() returning true if a cell is the tail of a gradient arrow and false otherwise, isHead() returning true if a cell is the head of a gradient arrow, and discreteTangent() applied to a cell returning the cell it is paired within the discrete gradient.

```
1: AddDescendingCells(cell β):
2: result = {β}
3: for α ∈ ∂β do
4:     if isTail(α) then
5:         β_next = discreteTangent(α)
6:         result = result ⋃ AddDescendingCells(β_next)
7:     end if
8: end for
9: return result
```

The input to the algorithm is the critical cell, and the output is the set of cells of the same dimension that form its descending manifold. The algorithm to compute ascending manifolds is similar, replacing the boundary operator with its inverse.

```
1: AddAscendingCells(cell α):
2: result = {α}
3: for β ∈ ∂^{-1}α do
4:     if isHead(β) then
5:         α_next = discreteTangent(β)
6:         result = result ⋃ AddAscendingCells(α_next)
7:     end if
8: end for
9: return result
```

6.2 Renderable Geometry from Manifolds

The nodes, arcs, and higher dimensional manifolds extracted from the discrete gradient field are initially represented as sets of cells, and we transform them into geometric primitives for rendering. For example, we use spheres of different colors as placeholders at the barycenters of critical cells, since they are a well-understood metaphor for critical points. Fig. 4 shows the geometry conversion for nodes, arcs, ascending and descending 2-manifolds, and ascending and descending 3-manifolds.

6.3 Maintaining Geometry in a Hierarchy

A simplification hierarchy records a sequence of cancellations of pairs of critical points. The geometry associated with the ascending and descending manifolds of any critical points neighboring a cancellation will change. Forman [42] showed that a cancellation can be realized in a discrete gradient field by simply reversing the discrete vectors on a V-path. In this setting, the ascending and descending manifolds can be recovered by the algorithm presented in Section 6.1. However, in a practical system where interactive

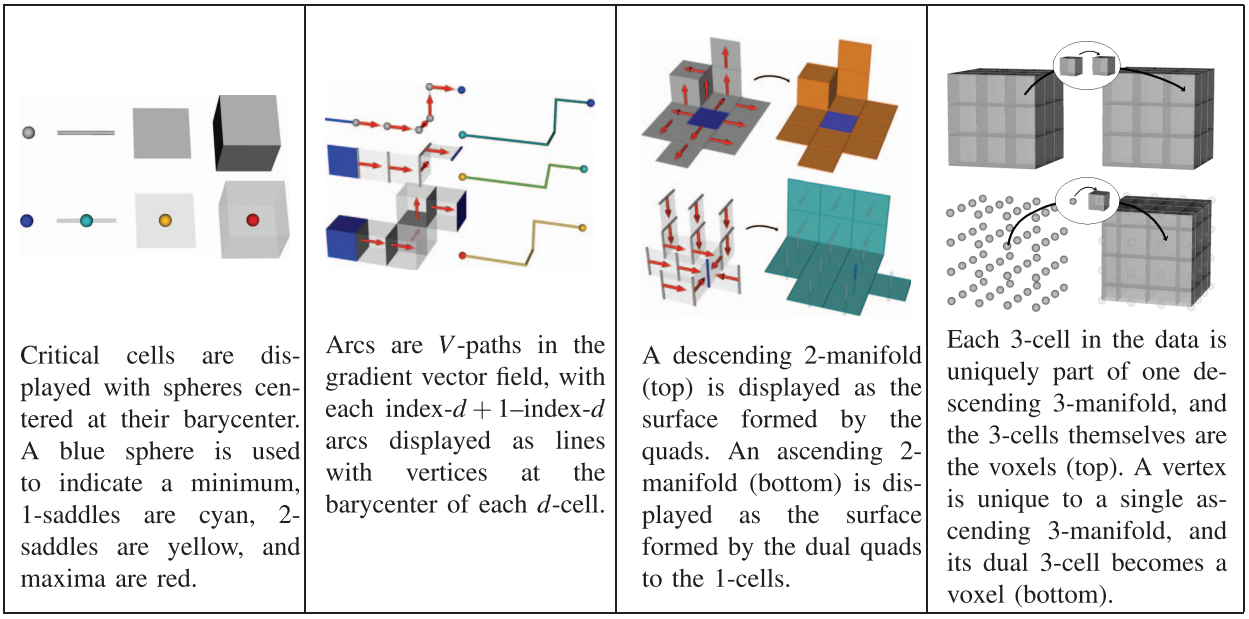


Fig. 4. Once topological structures are computed as sets of cells, they are converted into geometric primitives for rendering.

browsing of the hierarchy is desired, such an operation is very costly. Furthermore, anticancellations become just as expensive. Gyulassy et al. [34] introduced data structures for maintaining the geometry of arcs in a cancellation sequence, however, this structure did not handle the anticancellations necessary to browse a hierarchy. The data structures we introduce for maintaining manifold geometry in this section follow a similar merging structure, and define a similar acyclic graph. In a common usage scenario, a user may wish to skip over tens of thousands of cancellations to view the hierarchy at different persistences. Using our approach, a process that previously took several minutes becomes interactive.

We present an efficient technique for representing the geometry of ascending and descending manifolds at any time in the hierarchy by storing their merging in an acyclic graph. Gyulassy et al. [38], characterized how a cancellation changes both the structure of the complex and the geometry of the manifolds. When n_u, an index-$i+1$ critical point, and n_l, an index-i critical point, are canceled, the following changes occur to the manifolds:

1. For every node of index $i+1$ in the neighborhood of n_l, merge its descending manifold with the descending manifold of n_u.
2. For every node of index i in the neighborhood of n_u, merge its ascending manifold with the ascending manifold of n_l.

Using this foundation, our data structure creates a "merge" element every time the manifold of a node changes. A hierarchy is a simplification sequence, therefore we can assign an integer time to each creation/destruction event. In particular, the creation time of a merge element is equal to the number of cancellations so far in the sequence. We define the merge elements as follows:

$$merge ::= (base\_merge,\ other\_merge,\ time) \mid leaf\_geometry.$$

A merge object is either a 3-tuple of two merge objects and a merge time, or a pointer to the leaf geometry. Leaf geometry associated with a critical cell α is evaluated as the result of AddDescendingCells(α) or AddAscendingCells(α), and has an implicit merge time of 0. Every node n in the MS complex has one merge object for its ascending manifold geometry, denoted $n.asc\_man\_geom$, and one for its descending manifold geometry, $n.dsc\_man\_geom$. Initially, before any cancellation has occurred, every merge object is leaf geometry. During a cancellation, we create new merge nodes and update the graph structure. In the following algorithm, neighborhood() of a node returns the set of nodes that share an arc in the complex.

```
ManifoldCancellationUpdate(node n_l, node n_u, int cancel_time):
for n_i ∈ neighborhood(n_l), n_i != n_u do
    new_dsc_geom = (n_i.dsc_man_geom, n_u.dsc_man_geom, cancel_time)
    n_i.dsc_man_geom = new_dsc_geom
end for
for n_i ∈ neighborhood(n_u), n_i != n_l do
    new_asc_geom = (n_i.asc_man_geom, n_l.asc_man_geom, cancel_time)
    n_i.asc_man_geom = new_asc_geom
end for
```

The history of the changes to a node's geometry is stored in the $base\_merge$ field of the merge element: it is a list with the creation times of each merge object. Therefore, to extract the state of the merge graph at time t, we start at a node's merge element pointer, the newest merge node, and follow $base\_merge$ pointers until $cancel\_time$ is at most t. The merge element returned was the merge element pointed to by the node at time t in the creation of the hierarchy. Since directed edges in the merge graph always point from higher cancellation-time merge objects to lower cancellation-time objects, any path in this graph is monotonic, and hence the graph is acyclic. To recover the full ascending or descending

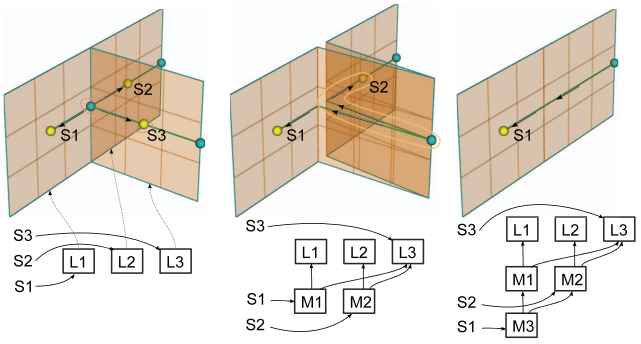


Fig. 5. S1, S2, and S3 are 2-saddles, with initial leaf geometry merge elements L1, L2, and L3, respectively (left). The first cancellation of S3 and the circled 1-saddle creates new merge elements M1 and M2, both having $time = 1$, for the nodes S1 and S2 (middle). The second cancellation, S2 with the circled 1-saddle, creates a new merge element M3 for S1 with $time = 2$ (right). The surface of S1 at $time = 1$ is L1 and L3. At $time = 2$, the surface from S1 is L1 and L2; L3 is counted an even number of times in a depth-first search.

manifold from a merge element, we gather the leaf geometries that are alive at the current simplification level with a depth-first search. The following algorithm returns the set of merge leaves that have been merged in the cancellation process to create the input merge element. In this algorithm, isLeaf() returns true if a merge element is a pointer to leaf geometry, false otherwise.

```
1: RecGatherOddLeaves(merge m, set& s):
2: if isLeaf(m) then
3:     if s ∩ {m}! = {} then
4:         s = s − {m}
5:         return
6:     else
7:         s = s ∪ {m}
8:         return
9:     end if
10: end if
11: RecGatherOddLeaves(m.base_merge, s)
12: RecGatherOddLeaves(m.other_merge, s)
```

Note that we only count leaves that have been visited an odd number of times: this simulates symbolically reversing the path along an arc during cancellation. Fig. 5 illustrates this: a single cancellation reverses the flow along an arc and extends the manifold for neighboring nodes; a second cancellation reverts those cells to their original direction. The following algorithm shows how to traverse a hierarchy to extract descending manifolds for a particular time in the simplification sequence.

```
1: DescendingManifoldAtTime(node n, int time):
2: m = n.dsc_man_geom
3: while m.time > time do
4:     m = m.base_merge
5: end while
6: s = {}
7: RecGatherOddLeaves(m, s)
8: return s
```

Fig. 5 illustrates these algorithms and data structures, and shows how they are maintained through some cancellation operations.

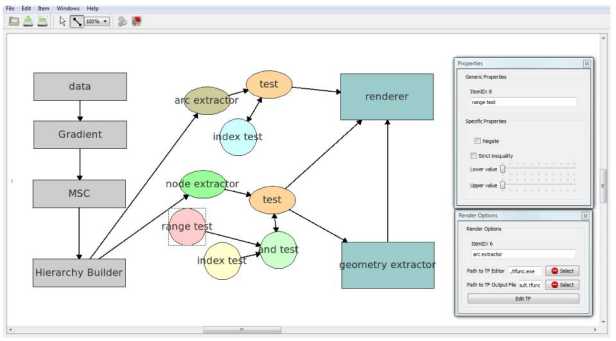


Fig. 6. A user specified work flow diagram is translated into queries on the MS complex. Updates to the diagram are reflected in real time in the output of the renderer. Here, a simple scene is illustrated. A gradient, MS complex, and hierarchy are precomputed from the input data. The result of node and arc extractors are filtered by selectors. Each selector is also linked with a Boolean test. The range test is selected, which brings up a window to manipulate its properties.

7 Topology-Rich Rendering

As a practical example, we describe our interactive feature extraction and visualization systems. These consist of two main components: an interface for constructing queries, modifying parameters and assigning rendering attributes, and a renderer that is specially designed to handle the kinds of features that can be identified using the MS complex.

7.1 Interactive Exploration

The data objects of an MS complex and function objects in a query have a natural branching and dependency structure, and our interface utilizes this by representing a scene as a work flow diagram. Our prototype implementation supports specifying queries in a graphical user interface, where the input and output to selectors are represented by arrows. The user chooses different types of objects (tests, selectors, etc.) from a tool bar and connects them together in a work flow graph. Each object and selector function has properties that can be manipulated. For example, the constant factors in a test function that implements range comparison can be changed with sliders to interactively adjust queries, and any changes force recomputation of downstream queries. The number of objects and depth of the work flow diagram are limited in practice, and each downstream object can be recomputed interactively. Our interface provides for saving and loading an interaction session in an XML file. This allows a scientist to use the GUI to design queries, and then use the resulting XML description to extract features in a batch job, for example, to dump surfaces or statistics to disk for several time steps of a simulation. Fig. 6 shows an interaction session in our interface prototype.

7.2 Rendering Attributes

A work flow diagram is also the scene graph and gives the user control of rendering attributes. Each query object of the MS complex can be assigned valuator functions that determine how its result is rendered by a render object. For example, a set of nodes can be assigned a valuator object that scales the radius of its rendered sphere by some transfer function. As another example, the surface output from a geometry extractor can be rendered with its own transfer function. In this way, we can assign valuators to

sets of nodes, arcs, surface, or volumes that are locally evaluated by each element in that set. Initially, the nodes and arcs extracted from an MS complex have default values for rendering attributes, such as coloring nodes by index. In our model, a new selector or node inherits its valuators from its parent.

7.3 Interactive Rendering

Our rendering system combines traditional volume rendering with direct feature visualization. Our depth-peeling GPU ray-caster allows inlays of semitransparent solid geometry (spheres, lines, tubes, surfaces) into a locally controlled volume rendering. The overriding design principle to our rendering system is to enable rendering of any feature using its own rendering attributes combined with the underlying scalar field. To achieve this, our implementation uses

1. the input volume of scalar values,
2. a volume of ascending or descending 3-manifold identifiers storing the segmentation of the domain,
3. a set of surfaces representing ascending and descending 2-manifolds,
4. a set of lines or tubes representing arcs, and
5. a set of spheres representing nodes.

The following describes how each of these is handled.

Volume rendering. In our volume ray caster, the four-channel colors (red, green, blue, and alpha) of sample points are found along a ray and composited front-to-back. Our system allows the user to select the transfer function, color, and blending mode for each topological feature individually. Furthermore, a user can rescale the function values within ascending or descending 3-manifolds, for example, to display ones with different ranges uniformly with the same transfer function.

In the following pseudocode that returns the color of a sample in the volume, let $f : \mathbf{R^3} \rightarrow \mathbf{R}$ be a function that returns the value of the input scalar field at the point, let $i : \mathbf{R^3} \rightarrow \mathbf{Z}$ be a function that maps the point to the unique ID of the ascending or descending 3-manifold containing it, let $rmin : \mathbf{Z} \rightarrow \mathbf{R}$ and $rmax : \mathbf{Z} \rightarrow \mathbf{R}$ map an ID to local rescale values, let $color : \mathbf{Z} \rightarrow \mathbf{R^3}$ map an ID to a constant color, let $transfer : \mathbf{Z} \rightarrow (\mathbf{R} \rightarrow \mathbf{R^3})$ map an ID to a transfer function, and finally let $blend : \mathbf{Z} \rightarrow [\mathbf{0}, \mathbf{1}]$ map an ID to a constant controlling the compositing of $color$ with $transfer$.

RGBA(point p):
1: $ID = i(p)$
2: $VAL = (f(p) - rmin(ID))/(rmax(ID) - rmin(ID))$
3: $RES.rgb = blend(ID) * transfer(ID)(VAL).rgb +$
$(1 - blend(ID)) * color(ID)$
4: $RES.a = transfer(ID)(VAL).a$
5: return RES

The result of $f(p)$ is rescaled by the feature rescale values found at ID $i(p)$ of the $rmax$ and $rmin$ tables, and this value is sent to the transfer function found for ID, returning a color. This color is blended with the assigned color of the feature, using the blend factor and color found for the ID in the $blend$ and $color$ tables, respectively.

Surface rendering. Surface colors are computed similarly to colors in a volume, except that every point on a surface has the same ID, and therefore the functions i, $rmin$, $rmax$, $color$, and $blend$ are all replaced by constants, and $transfer$ returns the same transfer function for all points on the surface.

Tubes and lines. Arcs are rendered as tubes or lines. A tube or line is represented as a list of vertices, each having a color and radius. Tubes and lines are fully opaque objects. The values for the color and radius along a tube or line can be set using a transfer function.

Spheres. Nodes are rendered as spheres. A sphere may have arbitrary color and radius. Spheres are fully opaque objects.

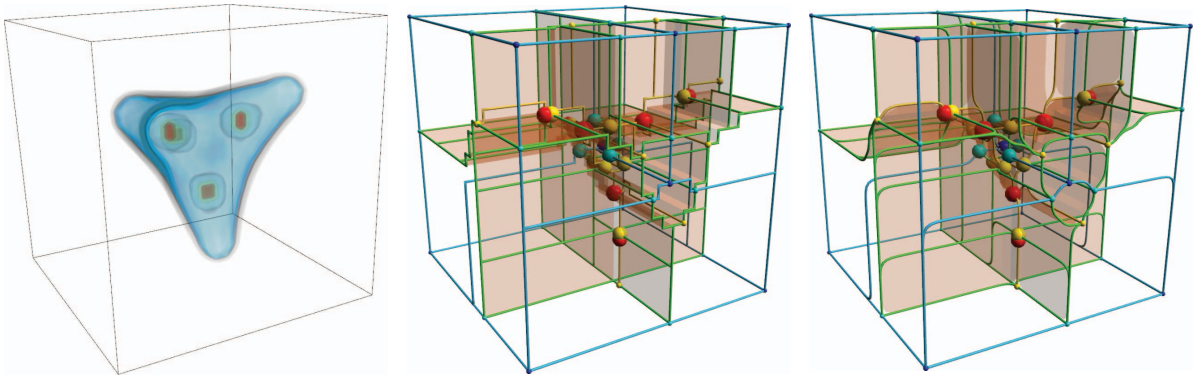
We utilize a GPU-based ray caster and depth peeling to resolve transparency. Samples in the volume and on surfaces and lines are shaded with fragment programs written in GLSL. The functions f and i are three-dimensional samplers operating on volumetric textures storing the scalar values and indices, respectively. The index volume i is sampled with no interpolation. A single transfer function is implemented as a one-dimensional texture lookup, and $transfer$ is implemented as a two-dimensional lookup. An additional mapping $texID : \mathbf{Z} \rightarrow \mathbf{Z}$ is used to map volume indices to transfer function ID, and the result of this lookup is the first coordinate in the two-dimensional transfer function texture. The other coordinate is simply the rescaled function value. Note that the maximum index number is the number of extrema generating the segmentation of the volume. GLSL restricts the maximum size of a texture dimension to 4,096, therefore $texID$, $rmin$, $rmax$, $color$, and $blend$ are implemented as linear lookups into two-dimensional textures. Note that we can apply Laplacian smoothing to lines and surfaces for aesthetics.

As the MS complex is simplified, several of the regions in the index texture will merge. Instead of updating the full volumetric texture, we simulate the merging by copying the parameters to the two-dimensional textures. This avoids having to modify the volume value and volume index textures.

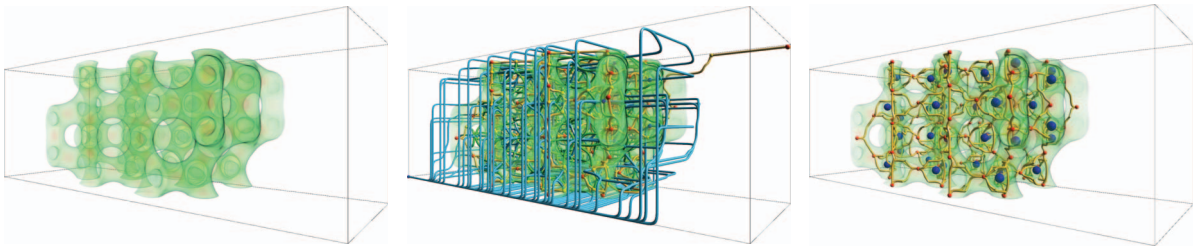
8 Results/Examples

Our visualization results were performed on an off-the-shelf 2.26 GHz laptop with 4 GB main memory and GeForce GT 130M graphics card with 1 GB physical memory. In each case, the 1-skeleton of the MS complex was precomputed using a single-block version of the algorithm presented by Gyulassy et al. [38], with preprocessing runtimes of the same order, taking between 1 second (Tetrahedrane) and 4 minutes (Cosmology). Subsequent exploration, in terms of query evaluation and parameter editing, is performed interactively, with frame rates varying between 30 frames per second for smaller, less complex scenes, to 0.5 seconds per frame for large scenes with high level of transparent overlap. We wish to emphasize that our renderer is unoptimized research code, and we expect that frame rates can be improved.

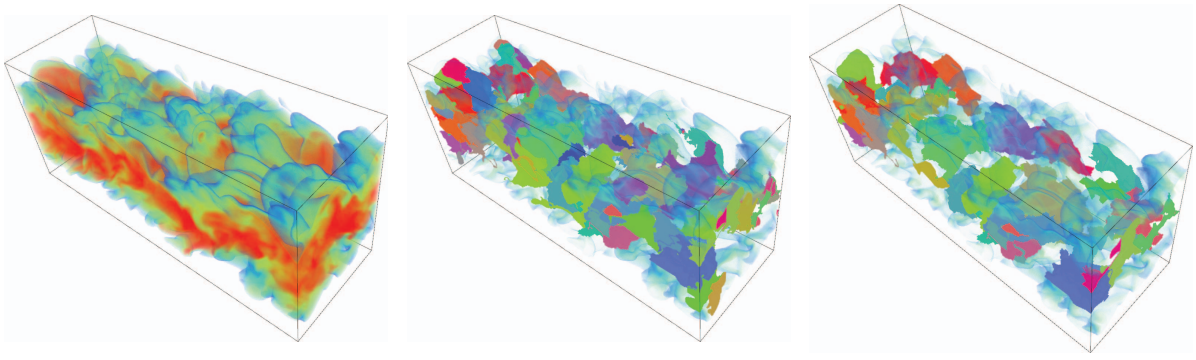
The overall memory requirements of this approach are output sensitive, however, every analysis pipeline begins with computation and storage of the discrete gradient field, taking exactly 8 bytes per sample location. Furthermore, we load the entire data set into main memory and GPU texture memory. If segmentation volumes are needed in the



Tetrahedrane (left), $24 \times 24 \times 24$ floats: The scalar value represents the probability distribution electrons in a tetrahedrane (C_4H_4) molecule. High peaks are centered around the atoms. We show selected elements from the computed MS complex, first without geometric smoothing (middle), and after some iterations of Laplacian smoothing (right). *Structure: (middle)* Add all nodes and arcs from the complex to be rendered. Set radius of the nodes to be a scale factor of the function value. Select all 1-saddles and extract their ascending 2-manifold geometry, render orange with transparency. *Geometry: (right)* Add all nodes and arcs from the complex to be rendered. Set radius of the nodes to be a scale factor of the function value. Select all 1-saddles and extract their ascending 2-manifold geometry, render orange with transparency. Lines and surfaces with three iterations of smoothing.



Silicium (left), $34 \times 34 \times 98$, byte: In the simulation of a silicon lattice, values correspond to electric potential. We compute the full MS complex (middle) and show atom locations (blue) and the tetrahedral arrangement of covalent bonds (red) around them (right). *Full MS complex: (middle)* Create a hierarchy simplifying to 10% total persistence. Select all arcs and nodes from the hierarchy and render them. *Peaks and ridges: (right)* Create a hierarchy simplifying to 10% total persistence. Select all 2-saddle-maximum arcs and add them to be rendered with smoothing. Select maxima incident to these arcs and minima between 15.0 and 35.0, add to rendering.

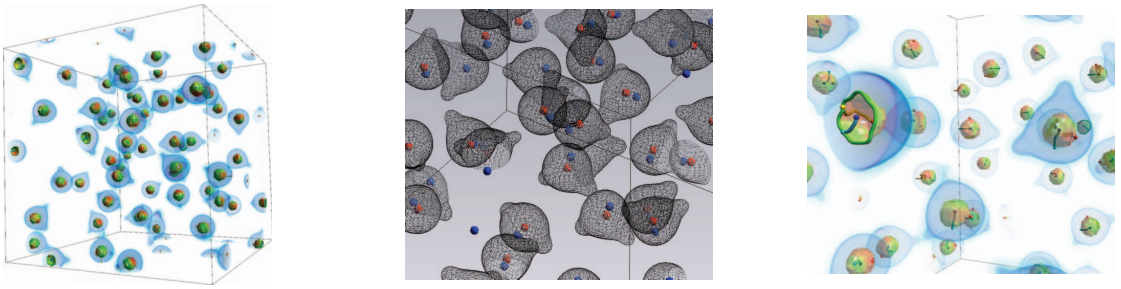


Combustion Jet (left), $384 \times 168 \times 124$, float: In the simulation of a combustion jet, fuel mixture fraction basins have been linked to dissipation elements. We render these volumetric elements at two topological scales. The basins at the finer scale (middle) merge into larger basins at the coarser scale (right). High values in the transfer function are red, low are blue. *Persistence 2.5%: (middle)* Create a hierarchy simplifying to 2.5% total persistence. Select all minima in the value range $[0.4, 0.6]$, and assign a random color. For each minimum, compute its ascending 3-manifold. Render each selected 3-manifold as fully opaque, with the associated minimum's color. Unselected 3-manifolds are rendered with a background transfer function. *Persistence 5%: (right)* Create a hierarchy simplifying to 5.0% total persistence. The selection and rendering is subsequently the same as (middle). The background transfer function for both is a ramp from blue (low value) to green to red (high values).

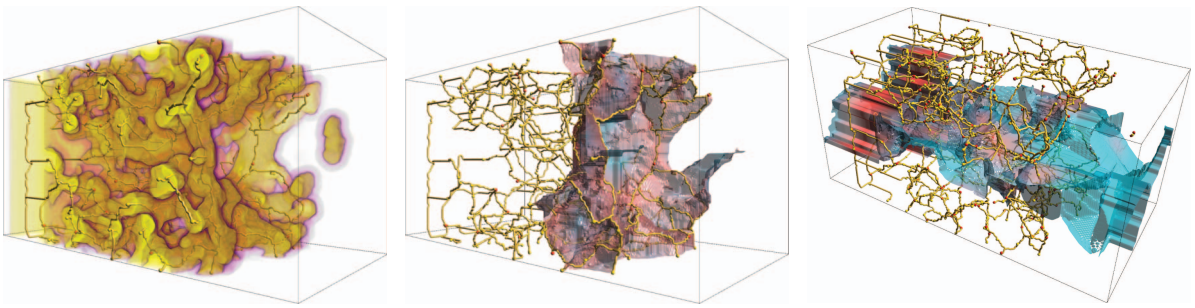
Fig. 7. In each of these visualizations, the caption provides a brief description of the data as well as the queries and parameters used to generate the images.

pipeline, they are also computed and stored in texture memory, taking 4 bytes per sample location. Nodes, arcs, and geometry objects are computed on-the-fly; for example, the *impact crater* surface of the porous solid required storing 30K triangles and function values, replicated on the GPU.

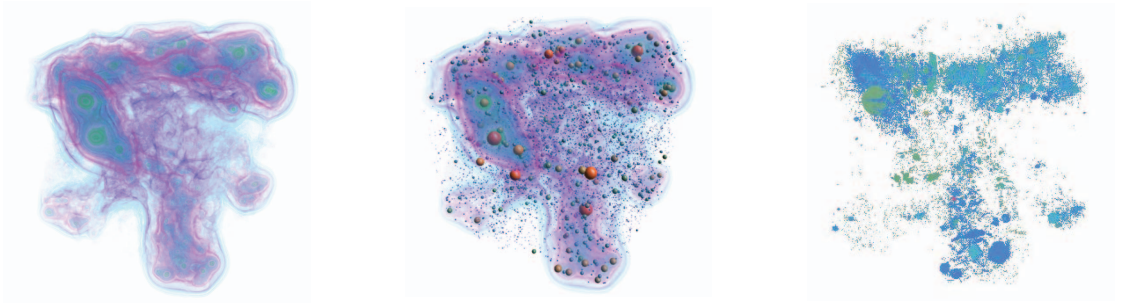
The examples in Figs. 7 and 8 show scenes highlighting the illustrative power and generality of this technique in order of increasingly complex topology-rich visualizations. The tetrahedrane has simple topology and is used to illustrate two rendering modes, showing the structure and



High-Pressure H_2O (left), $128 \times 128 \times 128$, float: The scalar value is electron density distribution in simulated high-pressure water. The high values forming a shell around the location of the oxygen atom. We extract this topological pouch, and color the surface with a separate transfer function. *Contour tree based features: (middle)* Applying contour-tree based analysis [44] to the same data, a persistent minimum/maximum pair without the separating surface suggests a dipole, not a pouch. *MS complex pouches: (right)* Create a hierarchy simplifying to 5.0% total persistence, select all extrema and arcs entirely above 0.2 for rendering. Extract ascending 2-manifolds for any 1-saddles selected, smooth, and apply a green (low) to red (high) transfer function with transparency.



Porous Solid (left), $115 \times 115 \times 237$, float: A signed distance field from the material boundary represents a porous copper foam. This time-step ends a sequence where a micro-meteoroid impacts the foam from the right. We identify core structure, and additionally we extract a surface representing the impact crater (middle), and shear surface through the weakest points connecting the top half of the material to the bottom (right). *Impact Crater: (middle)* Simplify to 5.0% total persistence using the threshold values found by Gyulassy et al. [41]. Select 2-saddle-maximum arcs in the range $[-2.0, 30]$ and incident nodes for rendering. Select the lowest minimum (located outside the crater, and select neighboring 1-saddles (using two incidence selectors). Extract ascending 2-manifolds from these saddles, smooth, and apply a blue (low) to red (high) ramp transfer function with transparency. *Fracture Surface: (right)* Use the same hierarchy and arcs/nodes for rendering as in (Middle). Select all 2-saddle-maximum arcs that cross y-coordinate = 57.5. Select 2-saddles from their incident nodes. Extract descending 2-manifolds from these, smooth, and apply a blue (low) to red (high) transfer function with transparency.



Cosmology Simulation (left), $256 \times 256 \times 256$, float: In this simulation of a 90 MParsec box [45], the scalar value represents particle density in space. We display peaks in this data in two ways: (middle) direct rendering of peaks; (right) locally-scaled transfer functions show the region of influence of a maximum, more correctly displaying the actual volume of a feature. *Peaks: (middle)* Simplify to 5.0% total persistence after a log scaling. Select all maxima in the hierarchy, and add to render. Set the color of the sphere to be based on a transfer function with a blue (low) to red (high) ramp, and the radius also to be scaled by the value. Render the volume with a background transfer function. *Mountain tops: (right)* Simplify to 5.0% total persistence after a log scaling. Select all maxima in the hierarchy and extract descending 3-manifolds. Set the local scaling factor on each 3-manifold to: *rmax* = value of maximum, *rmin* = value of highest neighboring 2-saddle. A step function is used to map values to alphas, and the color is set to the color of the maximum.

Fig. 8. Additional examples showing never-before-seen features. Each image combines traditional visualization techniques with various components of the Morse-Smale complex.

the geometry of the MS complex. The silicium example illustrates how simple queries can be composed to construct an abstract representation of the molecule. In the combustion simulation, we extract 3-manifold basins around minima hypothesized to be involved in flame extinction events, and show how persistence simplification allows exploration of these features at multiple topological scales. With the high-pressure H_2O model, we show how the MS

complex provides a more accurate interpretation of features: we show how topological pouches can be extracted, something not possible with previous approaches. In the porous medium data set, we illustrate how to build nonphysical features, such as the crater surface derived from the impact with a micrometeoroid, or the most likely shear fracture dividing the top and bottom of the material. In the last example, we show how the same analysis of a cosmology simulation can be visualized in drastically different ways: in one, the maxima of density are rendered as spheres whose size and color depend on function value; in the other the largest simple contours around a maximum are displayed with color according to function value. Overall, this set of examples illustrates the power of our topology-rich visualization technique, showing a variety of visualizations that were not available within a single integrated system.

9 Conclusions/Future Work

The queries and data structures we introduced allow our system to integrate interactive feature definition and exploration with data analysis and visualization. One current limitation to our tool is data size: the current implementation relies on GPU memory to store both the data and an ID volume. On the visualization side, this problem can be addressed by level-of-detail techniques, or techniques that compress relevant regions in the segmentation. Furthermore, our lazy evaluation of manifold geometry requires that the discrete gradient be stored in memory. We plan on addressing this with a distributed model, where a restricted space of features is specified and precomputed in parallel. The combinatorial nature of our algorithms ensures that the same work flow diagram results in identical analysis results on two different machines, and we wish to leverage this in building collaborative analysis tools. Finally, intermediate feedback, such as provided by statistics viewers, can aid in selecting parameters while designing a work-flow diagram, and we will augment our tool with such information.

References

[1] J.T. Kajiya and B.P. Von Herzen, "Ray Tracing Volume Densities," *Proc. 11th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 165-174, 1984.

[2] R.A. Drebin, L. Carpenter, and P. Hanrahan, "Volume Rendering," *Proc. 15th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 65-74, 1988.

[3] M.S. Levoy, "Display of Surfaces from Volume Data," PhD dissertation, Chapel Hill, NC, 1989.

[4] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 1, no. 2, pp. 99-108, June 1995.

[5] G. Kindlmann and J.W. Durkin, "Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering," *Proc. IEEE Symp. Vol. Visualization (VVS '98),* pp. 79-86, 1998.

[6] J. Kniss, G. Kindlmann, and C. Hansen, "Multidimensional Transfer Functions for Interactive Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 8, no. 3, pp. 270-285, July-Sept. 2002.

[7] C. Correa and K.-L. Ma, "Size-based Transfer Functions: A New Volume Exploration Technique," *IEEE Trans. Visualization and Computer Graphics,* vol. 14, no. 6, pp. 1380-1387, Nov./Dec. 2008.

[8] C.D. Correa and K.-L. Ma, "Visibility-Driven Transfer Functions," *Proc. IEEE Pacific Visualization Symp. (PACIFICVIS '09),* pp. 177-184, 2009.

[9] C. Rezk Salama, M. Keller, and P. Kohlmann, "High-Level User Interfaces for Transfer Function Design with Semantics," *IEEE Trans. Visualization and Computer Graphics,* vol. 12, no. 5, pp. 1021-1028, Sept./Oct. 2006.

[10] F.-Y. Tzeng, E.B. Lum, and K.-L. Ma, "An Intelligent System Approach to Higher-Dimensional Classification of Volume Data," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 3, pp. 273-284, May/June 2005.

[11] P. McCormick, J. Inman, J. Ahrens, J. Mohd-Yusof, G. Roth, and S. Cummins, "Scout: A Data-Parallel Programming Language for Graphics Processors," *Parallel Computing,* vol. 33, nos. 10/11, pp. 648-662, 2007.

[12] K. Stockinger, J. Shalf, K. Wu, and E.W. Bethel, "Query-Driven Visualization of Large Data Sets," *Proc. IEEE Conf. Visualization (VIS '05),* pp. 167-174, 2005.

[13] G. Reeb, "Sur Les Points Singuliers d'une Forme De Pfaff Complètement intégrable ou d'une Fonction Numérique," *Comptes Rendus de L'Académie ses Séances,* vol. 222, pp. 847-849, 1946.

[14] P. Cignoni, D. Constanza, C. Montani, C. Rocchini, and R. Scopigno, "Simplification of Tetrahedral Meshes with Accurate Error Evaluation," *Proc. Conf. Visualization '00,* pp. 85-92, 2000.

[15] H. Carr, J. Snoeyink, and M. van de Panne, "Simplifying Flexible Isosurfaces Using Local Geometric Measures," *Proc. IEEE Visualization '04,* pp. 497-504, Oct. 2004.

[16] I. Guskov and Z.J. Wood, "Topological Noise Removal," *Proc. Graphics Interface '01,* pp. 19-26, 2001.

[17] S. Takahashi, G.M. Nielson, Y. Takeshima, and I. Fujishiro, "Topological Volume Skeletonization Using Adaptive Tetrahedralization," *Proc. Geometric Modeling and Processing '04,* pp. 227-236, 2004.

[18] Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder, "Removing Excess Topology from Isosurfaces," *ACM Trans. Graphics,* vol. 23, no. 2, pp. 190-208, 2004.

[19] H. Carr, J. Snoeyink, and U. Axen, "Computing Contour Trees in All Dimensions," *Computational Geometry: Theory and Applications,* vol. 24, no. 22, pp. 75-94, 2003.

[20] Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote, "Simple and Optimal Output-Sensitive Construction of Contour Trees Using Monotone Paths," *Computational Geometry,* vol. 30, no. 2, pp. 165-195, 2005.

[21] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas, "Robust On-line Computation of Reeb Graphs: Simplicity and Speed," *ACM Trans. Graphics,* vol. 26, pp. 58.1-58.9, July 2007.

[22] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci, "Loop Surgery for Volumetric Meshes: Reeb Graphs Reduced to Contour Trees," *IEEE Trans. Visualization and Computer Graphics,* vol. 15, no. 6, pp. 1177-1184, Nov./Dec. 2009.

[23] G.H. Weber, S.E. Dillard, H. Carr, V. Pascucci, and B. Hamann, "Topology-Controlled Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 13, pp. 330-341, Mar. 2007.

[24] Y. Chiang and X. Lu, "Progressive Simplification of Tetrahedral Meshes Preserving All Isosurface Topologies," *Computer Graphics Forum,* vol. 22, no. 3, pp. 493-504, 2003.

[25] A. Cayley, "On Contour and Slope Lines," *The London, Edinburgh and Dublin Philosophical Magazine and J. Science,* vol. 17, pp. 264-268, 1859.

[26] J.C. Maxwell, "On Hills and Dales," *The London, Edinburgh and Dublin Philosophical Magazine and J. Science,* vol. 40, pp. 421-427, 1870.

[27] S. Rana, *Topological Data Structures for Surfaces: An Introduction to Geographical Information Science.* Wiley, 2004.

[28] S. Smale, "On Gradient Dynamical Systems," *Annals of Math.,* vol. 74, pp. 199-206, 1961.

[29] S. Smale, "Generalized Poincaré's Conjecture in Dimensions Greater than Four," *Annals of Math.,* vol. 74, pp. 391-406, 1961.

[30] H. Edelsbrunner, J. Harer, and A. Zomorodian, "Hierarchical Morse-Smale Complexes for Piecewise Linear 2-manifolds," *Discrete and Computational Geometry,* vol. 30, no. 1, pp. 87-107, 2003.

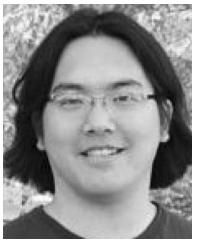
[31] P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci, "A Topological Hierarchy for Functions on Triangulated Surfaces," *IEEE Trans. Visualization and Computer Graphics,* vol. 10, no. 4, pp. 385-396, July/Aug. 2004.

[32] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Morse-Smale Complexes for Piecewise Linear 3-Manifolds," *Proc. 19th Ann. Symp. Computational Geometry,* pp. 361-370, 2003.

[33] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann, "Topology-Based Simplification for Feature Extraction from 3D Scalar Fields," *Proc. IEEE Conf. Visualization (VIS '05)*, pp. 535-542, 2005.
[34] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann, "A Topological Approach to Simplification of Three-Dimensional Scalar Functions," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 4, pp. 474-484, July/Aug. 2006.
[35] R. Forman, "A User's Guide to Discrete Morse Theory," *Séminaire Lotharingien de Combinatoire*, vol. B48c, pp. 1-35, 2002.
[36] T. Lewiner, H. Lopes, and G. Tavares, "Applications of Forman's Discrete Morse Theory to Topology Visualization and Mesh Compression," *IEEE Trans. Visualization and Computer Graphics*, vol. 10, no. 5, pp. 499-508, Sept./Oct. 2004.
[37] H. King, K. Knudson, and N. Mramor, "Generating Discrete Morse Functions from Point Data," *Experimental Math.*, vol. 14, no. 4, pp. 435-444, 2005.
[38] A. Gyulassy, P.-T. Bremer, B. Hamann, and V. Pascucci, "A Practical Approach to Morse-Smale Complex Computation: Scalability and Generality," *IEEE Trans. Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1619-1626, Nov./Dec. 2008.
[39] D. Laney, P.T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci, "Understanding the Structure of the Turbulent Mixing Layer in Hydrodynamic Instabilities," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1053-1060, Sept./Oct. 2006.
[40] P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell, "Analyzing and Tracking Burning Structures in lean Premixed Hydrogen Flames," *IEEE Trans. Visualization and Computer Graphics*, vol. 16, no. 2, pp. 248-260, Mar./Apr. 2010.
[41] A. Gyulassy, M. Duchaineau, V. Natarajan, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann, "Topologically Clean Distance Fields," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1432-1439, Nov./Dec. 2007.
[42] R. Forman, "Morse Theory for Cell Complexes," *Advances in Math.*, vol. 134, no. 1, pp. 90-145, 1998.
[43] F. Cazals, F. Chazal, and T. Lewiner, "Molecular Shape Analysis Based Upon the Morse-Smale Complex and the Connolly Function," *Proc. 19th Ann. Symp. Computational Geometry*, pp. 351-360, 2003.
[44] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli, "The TOPORRERY: Computation and Presentation of Multi-Resolution Topology," *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pp. 19-40, Springer, 2009.
[45] K. Heitmann, P.M. Ricker, M.S. Warren, and S. Habib, "Robustness of Cosmological Simulations, I. Large-Scale Structure," *The Astrophysical J. Supplement Series*, vol. 160, no. 1, p. 28, 2005.

Attila Gyulassy received the bachelor's of Arts in computer science and applied mathematics from the University of California, Berkeley in 2003 and the PhD degree in computer science from the University of California, Davis in 2009. His research interests as a postdoc at the Scientific Computing and Imaging (SCI) Institute, University of Utah, include topology-based data analysis and visualization.

Natallia Kotava received the MSc degree in computer science from the University of Kaiserslautern, Germany. Currently she is working toward the PhD degree in the computer graphics research group at the University of Kaiserslautern. Her research interests include graphics, scientific visualization, and computational geometry.



Mark Kim received the MS degree in computer science from the University of Denver in 2005 and is working toward the PhD degree under Professor Charles Hansen. His research interests include scientific visualization and interactive computer graphics.

Charles (Chuck) Hansen is a professor of computer science and an associate director of the Scientific Computing and Imaging Institute at the University of Utah. He was awarded the IEEE Technical Committee on Visualization and Graphics "Technical Achievement Award" in 2005 in recognition of seminal work on tools for understanding large-scale scientific data sets. He is a senior member of the IEEE.

Hans Hagen received the PhD degree in mathematics from the University of Dortmund, Germany, in 1982. He is currently a professor of computer science at the University of Kaiserslautern and the head of DFG's International Research Training Group on Visualization of Large and Unstructured Data Sets, Applications in Geospatial Planning, Modeling and Engineering. In 2009, he received the IEEE career award. He is a member of the IEEE.

Valerio Pascucci received the EE laurea (master's) degree from the University La Sapienza in Rome, Italy, in December 1993, as a member of the Geometric Computing Group, and the PhD degree in computer science from Purdue University in May 2000. He is an associate professor at the Scientific Computing and Imaging (SCI) Institute at the University of Utah. Before joining SCI, he served as a group leader and project leader at the Lawrence Livermore National Laboratory, Center for Applied Scientific Computing. He is a member of the IEEE.