A Practical Approach to Two-Dimensional Scalar Topology

Peer-Timo Bremer and Valerio Pascucci

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
{pascucci1,ptbremer}@llnl.gov

Summary. Computing and analyzing the topology of scalar fields has proven to be a powerful tool in a wide variety of applications. In recent years the field has evolved from computing contour trees of two-dimensional functions to Reeb graphs of general two-manifolds, analyzing the topology of time-dependent volumes, and finally to creating Morse-Smale complexes of two and three dimensional functions. However, apart from theoretical advances practical applications depend on the development of robust and easy to implement algorithms. The progression from initial to practical algorithms is evident, for example, in the contour tree computation where the latest algorithms consist of no more than a couple of dozens lines of pseudo-code. In this paper we describe a similarly simple approach to compute progressive Morse-Smale complexes of functions over two-manifolds. We discuss compact and transparent data-structures used to compute and store Morse-Smale complexes and demonstrate how they can be used to implement interactive topology based simplification. In particular, we show how special cases arising, for example, from manifolds with boundaries or highly quantized functions are handled effectively. Overall the new algorithm is easier to implement and more efficient both run-time and storage wise than previous approaches by avoiding to refine a given triangulation.

1 Introduction

Scalar field topology has proven to be a powerful tool in a wide variety of applications. In scientific visualization it has been used, for example, for speeding up the extraction of iso-surfaces [25], simplifying them [4], and also for general data analysis [3, 11]. However, topology is also useful as a shape descriptor [22, 14], can help in remeshing surfaces [24, 7], and widely used concepts like watershed and flood-fill algorithms [12] are also topological in nature.

The most complete description of the topology of a scalar field is its *Morse-Smale complex* which segments the field based on its gradient. While the usefulness of the Morse-Smale complex has been widely acknowledged it has rarely been applied in practice. The problem is that intuitive straightforward computations of the Morse-Smale complex which have been described

initially [23, 2] do not work in general. Only recently, generically correct algorithms to compute two- and three-dimensional Morse-Smale complexes have been discovered [9, 8]. However, these algorithms are notoriously difficult to implement which again limits their use in practice. Only the third generation of algorithms [3, 11] approach usability by combining correctness with a reasonable implementation.

In this paper we discuss a further improvement on the algorithms for computing the Morse-Smale complex of two-dimensional functions. In particular, we want to encourage the use of topology in practice. Therefore, the paper focuses not on any particular scientific result using Morse-Smale complexes but rather on providing a simple but complete description of the data structures and algorithms necessary to apply the theory of Morse-Smale complexes in general.

1.1 Related Work

Under various names the Morse-Smale complex has been used in a number of different contexts. Initially, they were used to describe the topology of terrains [5, 17] or other functions that could be connected to geography [26]. Naturally, the early research was limited to theoretical definitions rather than computer based algorithms. This line of research also includes the definitions of early multi-resolution structures [20, 21] as well as an in-depth discussion about potential degeneracies arising from non-smooth functions [19].

The first straight forward implementations are described by Takahashi et al. [23] and Bajaj and Schikore [1]. However, neither paper discusses any of the degeneracies (e.g. areas of zero gradient, merging integral lines) which arise when applying smooth theory to piece-wise linear functions. Therefore, these algorithms are likely to run into difficulties on any scalar field with non-trivial topology. The algorithms discussed in this paper are most closely related to the ones by Edelsbrunner et al. [9] and Bremer et al. [3]. Edelsbrunner et al. describe extensively how to handle merging and splitting steepest paths. However, their algorithm is quite involved and its implementation error prone. Bremer et al. simplify the implementation significantly by showing that only certain steepest lines must be kept separate and suggest to refine the mesh in order to avoid the complicated data structures of Edelsbrunner et al. Unfortunately, this approach, by definition, requires a dynamic mesh data structure able to refine the mesh at run-time. This is a distinct disadvantage when dealing with large data sets since it prevents the use of highly compact but rigid mesh encodings. Here we use a further extension of the algorithm in [3]. In Sect. 3 we show that not only can certain steepest lines be merged but those that must be kept separated can only create a very limited number of local configurations. We show how to encode these configurations efficiently and thus mostly avoid refining the mesh. Furthermore, Sect. 4 describes how a Morse-Smale complex with boundary is simplified and how to locally determine all possible simplifications.

The remainder of the paper is organized as follows: Section 2 introduces all necessary concepts of Morse theory and provides a collection of terms used throughout the paper. Section 3 discusses how to compute a Morse-Smale complex of functions on triangulated two-manifolds. Finally, Sect. 4 deals with simplifying a Morse-Smale complex with boundary and discusses a simple data structure to store a complex progressively.

2 Theory

In this section we introduce the necessary concepts from smooth and piecewise linear Morse theory. We refer to [18] and [16] for further background.

2.1 Morse-Smale Complexes

Throughout this paper, $\mathbb{M}$ denotes a compact 2-manifold without boundary and $f : \mathbb{M} \to \mathbb{R}$ denotes a real-valued smooth function on $\mathbb{M}$. Assuming a local coordinate system (x, y) at a point $a \in \mathbb{M}$, the point is called *critical* if its *gradient* $\bigtriangledown f(a) = (\partial f/\partial x, \partial f/\partial y)$ vanishes and called *regular* otherwise. Examples of critical points are maxima (f decreases in all directions), minima (f increases in all directions), and saddles (f switches between decreasing and increasing more than twice around the point).

Using the local coordinates at a, we compute the *Hessian* of f denoted by $H(a)$, which is the matrix of second order partial derivatives. A critical point is *non-degenerate* if the Hessian is non-singular, which is a property that is independent of the local coordinate system. According to the Morse Lemma, it is possible to construct a local coordinate system such that f has the form $f(x, y) = f(a) \pm x^2 \pm y^2$ in a neighborhood of a non-degenerate critical point a. The number of minus signs is the *index* of a and distinguishes the different types of critical points: minima have index 0, saddles have index 1, and maxima have index 2. The function f is a *Morse function* when all its critical points are non-degenerate and have pairwise different function values.

At any regular point, the gradient (vector) is non-zero, and when we follow the gradient we trace out an *integral line*, which starts at a critical point and ends at a critical point, while technically not containing either of them. Since f is smooth, two integral lines are either disjoint or the same. The *descending manifold* $D(a)$ of a critical point a is the set of points that flow toward a. More formally, it is the union of a and all integral lines that end at a. The collection of descending manifolds is a complex in the sense that the boundary of a cell is the union of lower-dimensional cells. For example, the descending manifolds of maxima are open discs, whose boundary consists of the descending manifolds of saddles (open intervals) which in turn are bounded by (descending manifolds of) minima (points). Symmetrically, we define the *ascending manifold* $A(a)$ of a as the union of a and all integral lines that start at a. If no integral line starts and ends at a saddle, see [9], we can overlay these

two complexes and obtain the *Morse-Smale complex* (MS complex) of f. Its *nodes* are the vertices of the two overlayed complexes, which are the minima, maxima, and saddles of f. Its *arcs* are integral lines starting or ending at saddles, and its *regions* are areas bounded by four arcs. An example is shown in Fig. 8(a). The MS complex provides a complete topological segmentation of $\mathbb{M}$ and thus is the fundamental theoretical structure many popular algorithms like contour-tree extractions or watershed computations are build upon.

2.2 PL Morse Theory

Unfortunately, traditional Morse theory is very dependent on f being smooth; a requirement rarely fulfilled in practice. Usually, one deals with piece-wise linear (pl-) functions defined by function values at the vertices of a triangulation $T = (V, E, F)$ and linearly interpolated on edges and faces of T. For the moment, let us assume that no two neighboring vertices have equal function value. To succinctly describe pl-extensions of Morse theory we first need some handy definitions. Given a vertex $v \in V$ the *star* $St(v)$ is defined as the collection of simplices containing v: $St(v) = \{\sigma \in T | v \in \sigma\}$. The *upper star* $St^+(v)$ consists of all simplices in the star whose vertices all have function values higher than v: $St^+(v) = \{\sigma \in T | u \in \sigma \Rightarrow f(u) > f(v)\}$ and the *lower star* $St_-(v)$ is defined symmetrically. The *link* $Lk(v)$ is defined as the boundary of the star $Lk(v) = \partial St(v)$ and the *lower link* $LL(v)$ as the subset with function values below that of vertex v: $LL(v) = \{\sigma \in Lk(v) | u \in \sigma \Rightarrow f(u) < f(v)\}$.

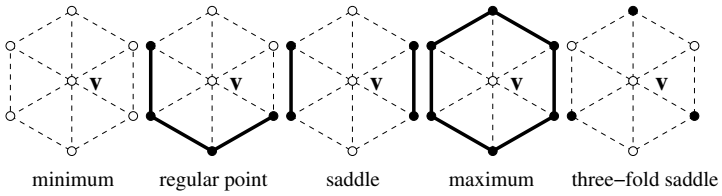


Fig. 1. Classification of a vertex v based on its lower link drawn in bold solid black

Many of the key-concepts necessary to define the Morse-Smale complex do not exist for pl-functions. Most notably, derivatives are not defined. In the following we show how all necessary concepts can be adapted to pl-functions leading to the definition of a *(quasi) Morse-Smale complex* [9]. First, we justify the assumption that no two vertices of T have the same function value by breaking ties using, for example, vertex indices. Now, the topology of the lower link can be used to classify vertices, see Fig. 1. A vertex v is a maximum if its lower link is the entire link and a minimum if its lower link is empty. In all other cases the lower link consists of $k \geq 1$ connected pieces. A vertex is regular if $k = 1$ and a k-fold saddle otherwise. Essentially, v is classified using an arbitrarily small neighborhood around v rather than derivative information

at v. The next step is to ensure that there exist no multiple saddles ($k > 1$) by splitting them into simple ones as described in Sect. 3.2. Finally, integral lines are replaced by *steepest lines*. As the name suggests, steepest lines are defined as greedily following a steepest ascending or descending line starting at a given vertex. In general, steepest lines are not uniquely defined. However, given any set of non-crossing steepest lines one can define ascending and descending manifolds for pl-functions and their intersection returns a quasi MS complex [9]. A quasi MS complex is guaranteed to have the same combinatorial structure as a Morse-Smale complex but is not unique for a given pl-function. Nevertheless, the differences between two equally valid quasi MS complexes are usually minor and mostly due to inadequate sampling. In the following we ignore these differences and simply refer to the MS complex.

3 Computation

This section contains an in-depth description of how to compute MS complexes on triangulated two-manifolds with boundary. First, we introduce the data structure and list all flags needed during the algorithm. Second, we discuss the necessary algorithms in detail and provide corresponding pseudo-code to allow easy reimplementation.

3.1 Setup

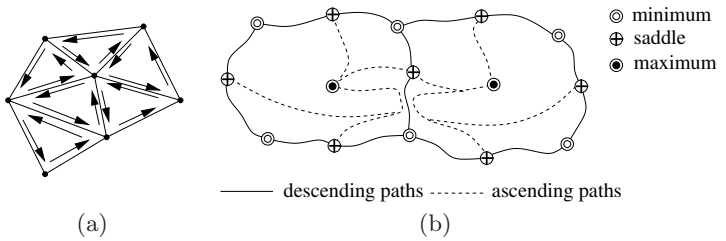


Fig. 2. (**a**) The mesh is stored as a standard half-edge data structure. (**b**) If descending paths are computed first an edge can be shared by at most three paths resulting in an ascending-descending-ascending classification. Any additional path merges with one of the existing ones

We store the triangulation in a typical half-edge data structure consisting of vertices, (half-)edges, and faces, see Fig. 2(a). Each vertex stores a function value and the function is continued to edges and faces by piece-wise linear interpolation. To each element we add some flags necessary for the algorithm:

- vertex: each vertex stores a type and a classification
- half-edge: each half-edge stores a classification and direction.

A vertex type can be either minimum, maximum, saddle, or regular. Vertices and half-edges are classified as either free or part of an ascending and/or descending path. The classifications of two neighboring half-edges provide a directed classification of edges. Given a certain orientation an edge can be, for example, part of an ascending path on its left but part of a descending paths on its right. As will be discussed below we compute all descending paths before computing ascending paths and paths in the same direction merge. As a result the most complicated edge classification possible is ascending-descending-ascending as shown in Fig. 2(b). This makes it possible to store all possible edge-classifications using two bits in each half-edge. Another bit is used to indicate the direction (upwards or downwards) of a half-edge. The vertex type and classification also each take two bits. If necessary, the memory foot-print can be optimized further by encoding the edge classification in three bits per edge (not half-edge) leaving one bit for the direction. The resulting two bits per half-edge can be stored in one byte per triangle.

3.2 Algorithm

The algorithm proceeds in four steps:

1. Preparing the mesh;
2. Computing descending paths;
3. Computing ascending paths;
4. Extracting connectivity information between Morse-Smale regions.

Mesh preparation. Section 2.2 discussed how areas of zero gradient are simulated as non-degenerate by breaking ties arbitrarily via the direction flags. There exist other degeneracies that for various reasons we also resolve symbolically before computing an MS complex. In particular, we split multiple saddles into a collection of simple saddles and remove boundary saddles.

Figure 3(a) shows how a saddle s of arbitrary multiplicity can be recursively split into a collection of simple saddles. While the multiplicity of s is larger than two we compute a single component C of its lower link and create a new vertex u with $f(u) = f(s)$. We delete all edges incident to C from s and connect them to u. Furthermore, we create three new edges connecting u to s and to the two vertices a and b neighboring C in the link of s (creating two new faces in the process). Finally, we mark the (directed) edge from u to s as descending and the edges from u to a and b as ascending. As a result, u is a simple saddle and the multiplicity of s has been reduced by one. In practice, one often wants the new vertices created in the process of splitting multiple saddles to have a reasonable embedding. One choice of embedding for u that works well in practice is to pick u as the mid-point of one of the edges between s and a vertex in C, as shown in Fig. 3(a).

Other than areas of zero gradient and multiple saddles, boundary saddles are in principle not degenerate. Even smooth Morse functions on manifolds with boundary can have saddles on the boundary. However, Morse theory in

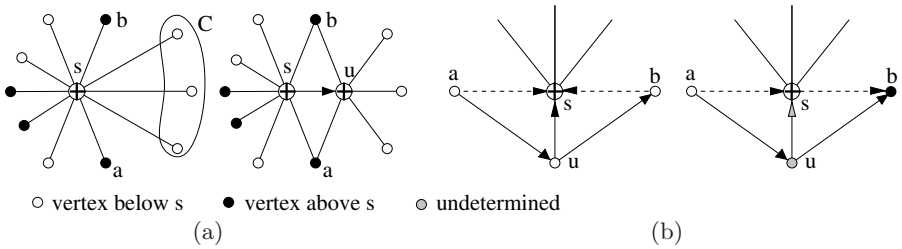


Fig. 3. Preparing the mesh: Filled/empty circles indicate vertices above/below s and for selected edges the arrow indicates ascending direction. (**a**) Splitting a multiple saddle by "removing" one component of its lower link. (**b**) Padding boundary saddles: Restricted to the boundary the saddle can be an extremum (left) or a regular vertex (right). It is padded by two triangles moving it into the interior. The grey color indicates that the choice of direction for u, s on the far right depends on the actual function values

general does not apply to manifolds with boundary. Instead, one must look to stratified Morse theory [10] for theoretically applicable results. A central assumption in stratified Morse theory is that there exist no boundary saddles. Apart from the theoretical justification for removing boundary saddles we found that they significantly complicate the computation as well as simplification of an MS complex.

We remove boundary saddles by padding them with two new triangles effectively moving them to the interior of the mesh. Figure 3(b) illustrates this process. Let s be the boundary saddle and a and b its neighboring vertices along the boundary. We create a new vertex u and two new triangles (u, s, a) and (u, b, s) and wlg. assume $f(a) < f(s)$ and $f(a) < f(b)$. We set $f(u) = 0.5*$ $(f(a) + f(b))$. We mark the edges (a, u) and (u, b) upward and depending on $f(u) < f(s)$ or $f(u) > f(s)$ we mark (s, u) downward or upward. As a result, u is a regular boundary vertex and s has become an interior saddle which, if necessary, can be split into simple saddles as described above. Splitting saddles as well as moving boundary saddles is a local operation that is performed during the initial read-in of the mesh and therefore does not require a dynamic mesh structure.

Descending Path Computation. For the remainder of the paper we assume that the mesh has been prepared according to the previous section and there exist no multiple or boundary saddles. In the next step we compute all descending paths. When computing integral lines in general there exist two choices. One can restrict steepest lines to follow edges in the triangulation [9] or trace steepest lines through triangles [3]. This paper focuses on the first option as it requires more complicated algorithms and potentially creates a number of challenging degenerate cases. Furthermore, tracing along existing edges is faster and requires no additional mesh refinement. Nevertheless, we

indicate how all algorithms can be adapted to tracing steepest lines through triangles and what the advantages and disadvantages are in either case.

Paths are traced starting at saddles and ending at extrema. We store paths by marking edges and vertices incident to them. As discussed above, edges are marked in a directed way such that an ascending-descending classification can be distinguished from a descending-ascending one. Initially, only critical points are classified non-free: Maxima/minima are classified as part of an ascending/descending path and saddles as both.

Descending paths are computed following three rules:

R1: Two paths of the same type can merge;
R2: Paths are not allowed to split;
R3: Paths that reach the boundary stay on the boundary;

R1 is a direct consequence of f being piece-wise linear. One can easily imagine a valley floor consisting of only a single edge. All descending lines flowing into such a valley must merge on this edge. R2 is fulfilled implicitly, since each path is computed following the same (deterministic) algorithm. R3 potentially forces a path from its "correct" location since boundary edges are not necessarily the steepest edges. However, by enforcing rule R3 we effectively avoid

```
Let T = (V, E, F) be a triangulation
ComputeAllDescendingPaths(Vertices V,Edges E)
  forall s ∈ V with MorseIndex(s) == 1        //for all saddles
    forall C component of LL(s)        //for each component of the lower link
      //find the steepest edge
      u = EndPointOfSteepestDescendingEdge(C, V, E);
      ClassifyAsDescending(Edge(s, u),left&right);
      if IsDescending(u) == false        //if we have not hit an existing path
        ClassifyAsDescending(u);
        TraceSteepestDescendingLine(u);        //continue tracing
      endif
    endfor
  endfor

TraceSteepestDescendingLine(Vertex v)
  if IsBoundaryVertex(v) == true        //If v lies on the boundary
    u = GetLowerIncidentBoundaryVertex(v);        //stay on the boundary
  else
    u = EndPointOfSteepestDescendingEdge(LL(v), V, E);
  endif
  ClassifyAsDescending(Edge(s, u),left&right);
  if IsDescending(u) == false        //if we have not hit an existing path
    ClassifyAsDescending(u);
    TraceSteepestDescendingLine(u);        //continue tracing
  endif
```

Fig. 4. Pseudo-code to compute all descending paths in a triangulation

detachment points [13] and simplify constructing the final MS complex significantly. If necessary, one can check whether rule R3 has significantly altered the final complex by re-computing an unconstrained steepest line and comparing the end-points. In such a case the complex can be corrected accordingly [9].

Following these rules, computing descending paths is straightforward: In each component of the lower link of all saddles we search for a steepest descending edge and classify it descending on both sides. If its end-point a is not yet classified descending we mark it as such and recursively look for a steepest descending edge out of a. The detailed algorithm is shown in Fig. 4.

Note, that in the boundary case of TraceSteepestDescendingLine there exists a unique lower boundary vertex neighboring v since otherwise v would be a saddle. Furthermore, the test whether a vertex u is already classified as descending combines three possible cases. We stop tracing a path when we have reached: another path (when we merge), a minimum (when we are done), or another saddle. In the last case the path merges with one of the descending path starting at u. However, there exist two equally valid choices. By convention, we assume that a descending path hitting another saddle verges to the right relative to its downward direction. Figure 5(a) shows a stable manifold whose boundary is computed by the algorithm described above.

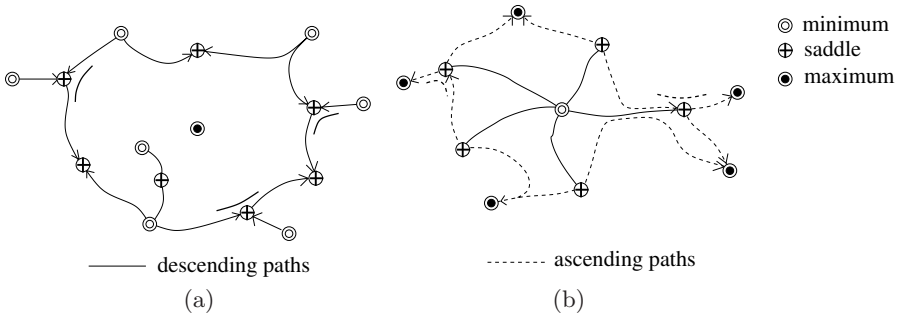


Fig. 5. (**a**) Stable manifold as computed by ComputeAllDescendingPaths. The arrows indicate the ascending direction and the bold line elements indicate how paths hitting saddles are routed. (**b**) Unstable manifold as computed by ComputeAllAscendingPaths

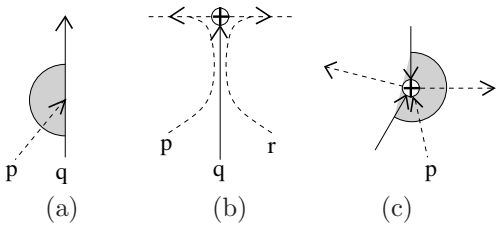


Fig. 6. Degeneracies encountered when computing ascending paths. The shaded disk slices indicate possible choices for the steepest edge.

When one is interested in tracing paths through triangles the simplest method is to change EndPointOfSteepestDescendingEdge. Rather than only examining all existing edges one computes the true steepest path. If this path runs through a triangle one splits the triangle to create a corresponding edge and returns the newly created vertex. However, refining extremely large meshes is often undesirable and in the case of out-of-core meshes can be prohibitively expensive.

Ascending path computation. In principle, ascending paths are computed symmetrically to the descending ones. The same three rules R1-R3 hold but another one must be added:

R4: Two paths are not allowed to cross.

Whenever one looks to extend a steepest ascending path the choice of "steepest" edge must be modified to observe rule R4. This becomes relevant when descending and ascending paths share a vertex. Figure 6 shows some examples in which special care must be taken to not violate R4. In Fig. 6(a) an ascending path p has joined a descending path q from the left (relative to p's direction). Since p must not cross q only those ascending edges to the left of q or part of q can be used to extend p. In particular, there can exist another ascending path r joining q from the right as shown in Fig. 6(b). Even though p and r share edges they are separated by q and cannot merge. If they would be allowed to merge they could never split (rule R2) and therefore one of them would cross q at some point. The same principles apply if an ascending paths p hits a saddle s, see Fig. 6(c). There can exist only one ascending path starting at s that p can join without crossing descending paths. Given these conventions, the ascending paths can be computed as shown in Fig. 7.

There are two significant differences between computing the ascending compared to the descending paths. First, an ascending path does not necessarily stop once it hits another vertex classified as ascending or even a saddle. Only reaching a maximum or sharing an edge already marked (on the correct side) is a sufficient condition to stop tracing. Second, the search for steepest edges is constrained by rule *R4* and therefore only considers parts of the upper star of a vertex, see Fig. 6. As before, the function SteepestAscendingEdge can be adapted to compute the true steepest ascent by splitting triangles. Furthermore, it can be modified to refine the mesh to keep ascending and descending paths completely separate. The disadvantage is again the potential cost of refining the mesh, the advantage is that extracting the connectivity between MS regions, see below, becomes trivial.

Extracting connectivity. Once all paths are constructed one also knows all arcs (endpoints and geometric embedding) of the MS complex. Then we identify the starting edge of each path with the corresponding arc. (These are the edges used for the first step in ComputeAll(A-)DescendingPaths.) Trivially, the order of arcs around saddles is given by the order of these edges. An additional breadth-first traversal of only descending paths starting from the

```
Let T = (V, E, F) be a triangulation
ComputeAllAscendingPaths(Vertices V,Edges E)
  forall s ∈ V with MorseIndex(s) == 1        //for all saddles
    forall C component of St+(s)        //for each component of the upper star
      e = SteepestAscendingEdge(C, V, E);        //find the steepest edge
      //now we make an arbitrary choice to start all
      //ascending paths on the right side of an edge
      ClassifyAsAscending(e,right);
      if MorseIndex(v) ≠ 2  AND  IsAscending(u,right) = false
        TraceSteepestAscendingLine(e,right);        //start tracing
      endif
    endfor
  endfor

TraceSteepestAscendingLine(Edge e, Orientation side)
  v = Endpoint(e);
  if IsBoundaryVertex(v) = true        //If v lies on the boundary
    u = AscendingBoundaryEdge(v);
  else if IsDescending(v) = true  AND  side = right
    u = SteepestAscendingEdge(RightHalf(St+(v)),V, E);
  else if IsDescending(v) = true  AND  side = left
    u = SteepestAscendingEdge(LeftHalf(St+(v)),V, E);
  else
    u = SteepestAscendingEdge(St+(v), V, E);
  endif
  v = Endpoint(u);
  if MorseIndex(v) ≠ 2  AND  IsAscending(u,side) = false
    ClassifyAsAscending(u,side);
    if IsDescending(u,side)
      TraceSteepestAscendingLine(u,side);
    else if IsDescending(v) = true  AND  u ∈ LeftHalf(St−(v))
      TraceSteepestAscendingLine(u,left);
    else
      TraceSteepestAscendingLine(u,right);
    endif
  endif
```

Fig. 7. Pseudo-code to compute all ascending paths in a triangulation

minima determines the order of arcs around minima. Note, that care must be taken for descending paths flowing though saddles. Here, observing the earlier convention that paths always turn right at saddles is crucial. Knowing the order of arcs around minima and saddles is enough to deduce the order of arcs around maxima which completes the connectivity information. If necessary one can perform a flood-fill to determine which faces belong to which Morse-Smale region. When ascending and descending paths are kept disjunct the connectivity can also be computed using a simple flood-fill algorithm. In this case each Morse-Smale region corresponds to a single cluster of faces which

form a topological disc. The clusters must touch all four arcs of the region and the connectivity is induced directly by the connectivity of the mesh.

4 Simplification

In most applications computing the topology of a data set is only the first step. To use the topology for analysis one must often simplify it if only to remove noise. This section describes how to store an MS complex as a progressive mesh [15]. A progressive mesh is a data structure that for a given error threshold allows fast access to a simplified MS complex in which all features below the error threshold have been removed.

The generic operation to simplify an MS complex is called a *cancellation*. An example is shown in Fig. 8(a) and (b). The maximum-saddle pair u, v

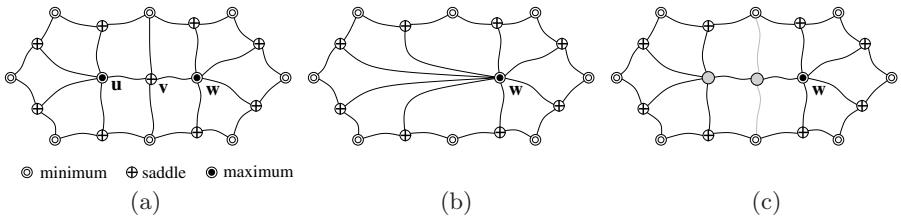


Fig. 8. Situation before (**a**) and after (**b**) canceling the critical point pair u, v. (**c**) Implementing a cancellation by deactivating nodes and arcs

is removed by merging four regions into two and extending all arcs ending at u. The reverse operation which introduces u, v and splits the regions is called an *anti-cancellation*. To rank cancellations each one is assigned an error indicating the importance of the corresponding node pair. Some examples are the difference in function value between the critical points involved (also called persistence), geometric distance, or absolute function value.

Valid cancellations. Not all connected critical point pairs can be canceled. For example, if a saddle is connected to the same extremum twice, see Fig. 9(a), it cannot be canceled with this extremum. Such a cancellation would require a change in genus of the underlying manifold which is usually not desirable. For MS complexes without boundary such double-connected saddles are the only configuration which cannot be canceled. For manifolds with boundary, however, there exist additional rules. Figure 9(b) shows why a boundary extremum can only be canceled if its "opposite" extremum across the saddle also lies on the boundary. Canceling a boundary extremum to the inside would create a boundary saddle which is disallowed. Let us assume a saddle is connected to two boundary maxima and should be canceled with either of them. The boundary minimum between the two maxima is implicitly

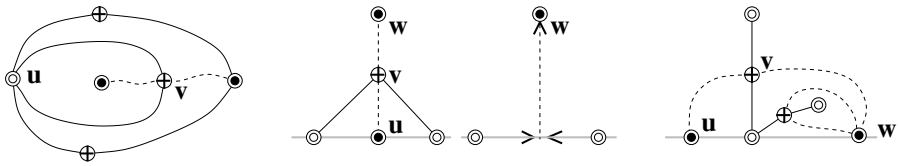


Fig. 9. Invalid cancellations: (**a**) Canceling u, v would change the genus of the underlying manifold. (**b**) Canceling u, v creates a boundary saddle at u. (**c**) Canceling u, v would remove all critical points between $u - v - w$ and the boundary

removed as well since the restriction of f to the boundary is a one-dimensional function of which minima and maxima can only be canceled in pairs. However, if the minimum is connected to additional saddles, as shown in Fig. 9(c), these structures must be removed as well since without the minimum they cannot form part of a valid MS complex. To avoid such implicit multi-cancellations we disallow the cancellation of boundary extrema unless they remove exactly three critical points. This condition is equivalent to requiring the minimum between the two maxima in our example to have valence one. Subsequently, we call a cancellation *valid* if it does not violate one of the three rules described above.

Construction. Given an error metric it is easy to progressively simplify an MS complex. Each arc of the MS complex corresponds to a possible (but not necessarily valid) cancellation. All valid cancellations are added into a priority queue which returns the arc/cancellation with the smallest error. As long as the queue is not empty we greedily cancel the critical point pair with smallest error. Note, that performing one cancellation can invalidate other cancellations. For example, a saddle can usually be canceled either with a minimum or with a maximum and clearly either choice will invalidate the other. For most metrics the error of a cancellation can be changed by neighboring cancellations as the connectivity of the complex changes. Typically, however, the error is only increased by earlier cancellations and therefore it can be updated in a lazy fashion. The complete simplification algorithm is shown in Fig. 10.

4.1 Progressive MS Complexes

The algorithm of Fig. 10 simplifies an MS complex until no valid cancellations remain. By introducing a break-off point it can easily be adapted to simplify a complex up to a certain error threshold. In practice, however, one usually wants to interactively change this threshold and simplifying the complex in a bottom-up fashion each time is slow and cumbersome. Instead, one wants to maximally simplify the complex once and keep a record of all intermediate configurations.

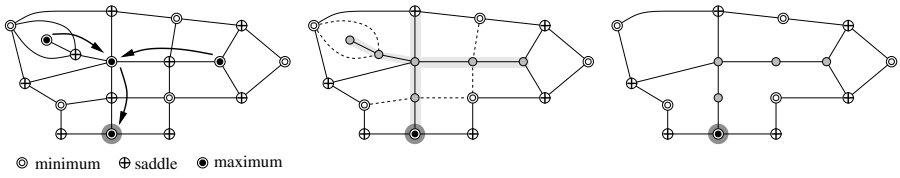
We store an MS complex as a graph consisting of nodes and arcs. Arcs are assumed to be ordered around nodes and regions are not represented explicitly.

```
Let MS = (N, A, R) be an MS complex
Let Q be a priority queue of arcs ordered by increasing error
SIMPLIFY(MS complex MS)
  forall a ∈ A     //for all arcs in the complex
    if ISVALIDCANCELLATION(a)
      PUSH(Q, a, COMPUTEERROR(a));
    endif
  endfor
  while ISEMPTY(Q) == false
    top = POP(Q);
    if ISVALIDCANCELLATION(top)     // if this cancellation is still valid
      err = COMPUTEERROR(top);
      if (err ≤ NEXTPRIORITY(Q);
        CANCELCRITICALPOINTS(a);
      else
        PUSH(Q, a, err);
      endif
    endif
  endwhile
```

Fig. 10. Pseudo-code to maximally simplify an MS complex

Using this simple data structure we implement a cancellation as shown in Fig. 8(c). Rather than deleting the two arcs and two critical points, we only "deactivate" them. deactivating a node creates "super-arcs" by concatenating the remaining active arcs and deactivated arcs are treated as being removed. A progressive MS complex is stored by assigning each node and each arc an error. Prior to simplification, the errors are initialized to a value larger than the largest possible error. During a cancellation elements are deactivated by setting their error to the error of the current cancellation. For any given error threshold the corresponding simplified MS complex is given by the collection of active nodes and super-arcs, where an element is considered active if its error is larger than the current threshold.

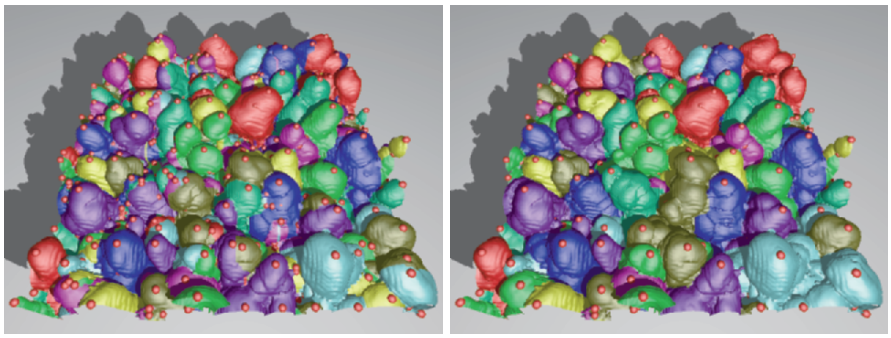
To explicitly construct a simplified MS complex for a particular error budget one performs restricted breadth-first traversals of the graph starting at each active extremum. The restrictions are that only active arcs and deactivated nodes can can be traversed. Each traversal covers a tree of active arcs rooted at the starting extremum with either active saddles or deactivated extrema as leaves. (Note, that an active saddle can be a leaf of two different branches of this tree. However, in this case it makes sense to consider the saddle as two nodes and maintain that the structure has no cycle.) Each path from a leaf to the root represents a super-arc. The simplified MS complex consists of all active nodes and those super-arcs that start at saddles, see Fig. 11. In practice, the tree traversals are helpful in a variety of situations. For example, a complex can be rendered very efficiently by associating each arc with a line strip and creating appropriate display lists during the tree traversals. Furthermore, when one considers a cancellation as the merging of



(a) (b) (c)

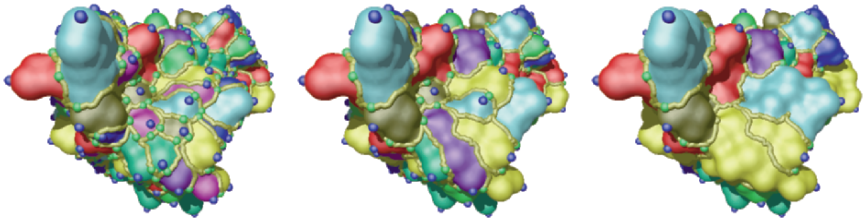
Fig. 11. Accessing a simplified MS complex using tree traversals. (**a**) Original MS complex with three cancellations indicated by the arrows. (**b**) The complex of (a) after the cancellations. The tree traversed from the indicated root is shaded in grey. (**c**) The final MS complex. Note, that the left most branch of the tree ended at an extremum and has been removed

three critical points into one, the deactivated nodes in each tree represent all critical points merged with the root. The main advantage of this data structure is that it is small and very easy to implement. The only disadvantage is that creating a simplification takes time linear in the size of the full complex, rather than linear in the size of the simplification. However, in all applications we have encountered so far the MS complex is several orders of magnitude smaller than the surfaces mesh. For example, the full resolution surfaces of the Mixing Fluids data set (see Sect. 5) contain up to 22 million vertices yet never more than about 20.000 maxima/bubbles. Therefore, the simplification remains fully interactive for all models we have been able to process using this approach.



(a) (b)

Fig. 12. Segmentation of parts of the Mixing Fluids data sets into bubbles. Maxima are shown in red and each bubble is randomly assigned one of nine colors. (**a**) Initial segmentation; (**b**) Segmentation at 0.2% persistence (relative to the maximal range in function value). (Colorplate on p. 218.)



(a) (b) (c)

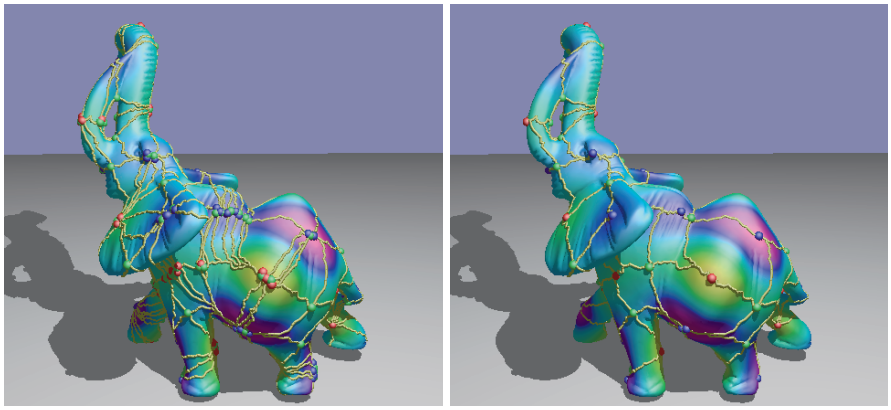
Fig. 13. Segmentation of the atomic density function on a molecule. Minima are shown in blue and ascending paths in gold. Segmentation into 198 (**a**); 100 (**b**); and 50 (**c**) protrusions (colorplate on p. 218).

5 Applications

The algorithms described above provide an efficient and stable method to compute MS complexes of any function on a triangulated manifold. We have found this a very versatile tool helpful in a variety of independent areas. Here, we show three very different applications for which we currently use MS complexes: The first is a physics simulation of the turbulent mixing between two fluids. Figure 12 shows an iso-surface between two mixing fluids extracted from one time-step of a simulation performed at the Lawrence Livermore National Laboratory. The data has been generated by the Miranda code a higher order hydrodynamics code for computing fluid instabilities and turbulent mixing [6]. In particular, scientists are interested in "bubbles" formed during the mixing process and their automatic segmentation. Using the z-coordinate as Morse function and the iso-surface (not the xy-plane) as domain bubbles can be defined as the descending manifolds of maxima. Nevertheless, the initial segmentation shown in Fig. 12(a) is not optimal as some bubbles have multiple maxima and there exist many superfluous maxima caused by noise in the data set. Using a uniform simplification of the MS complex one can remove most of these artifacts and create a much cleaner segmentation, as shown in Fig. 12(b).

The second application is molecular biology where one is interested in segmenting a molecular surface into cavities and protrusions. We take a skin surface of chain A from the protein complex Barnase/Barstar and compute the atomic density function over this surface. The ascending manifolds of minima of this function segment the surface into protrusions, see Fig. 13. As with bubbles, simplifying the MS complex captures protrusions at coarser and coarser level.

Finally, we use the MS complex to help remesh surfaces. As described in [7], MS complexes of eigenfunctions of the Laplace matrix of a surface are well suited to form an all quadrilateral basemesh, see Fig. 14.



(a) (b)

Fig. 14. Creating a basemesh from Laplacian eigenfunctions. (**a**) Initial MS complex of the 60th eigenfunction of the elephant showing typical noise due to discretization and the iterative eigensolver. (**b**) The MS complex of (a) simplified to a persistence of 0.5%. All noise has been removed and the MS complex now forms a high quality all-quadrilateral basemesh (colorplate on p. 218).

6 Conclusions

We have shown a new algorithm to compute two-dimensional MS complexes which allows a straightforward implementation of the original methods discussed in [9]. Except for few cases during an initial setup phase no refinement of the mesh is necessary. Nevertheless, if desired the algorithm can be trivially adapted to the one discussed in [3] which computes true lines of steepest a-/descent. Additionally, we have provided a novel easy-to-implement data structure for MS complexes which can also be used to encode the complex progressively. Finally, we have introduced restricted tree traversals to allow fast access to a simplified complex and related information as well as to provide efficient rendering of simplified complexes.

Acknowledgments

This work was performed under the auspices of the U. S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

1. C. L. Bajaj, V. Pascucci, and D. R. Schikore. Visualization of scalar topology for structural enhancement. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *Proc. IEEE Visualization '98*, pages 51–58, Los Alamitos California, 1998. IEEE, IEEE Computer Society Press.
2. C. L. Bajaj and D. R. Schikore. Topology preserving data simplification with error bounds. *Computers and Graphics*, 22(1):3–12, 1998.

3. P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. A topological hierarchy for functions on triangulated surfaces. *IEEE Trans. on Visualization and Computer Graphics*, 10(4):385–396, 2004.
4. H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *IEEE Visualization '04*, pages 497–504. IEEE Computer Society, 2004.
5. A. Cayley. On contour and slope lines. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, XVIII:264–268, 1859.
6. A.W. Cook, W. Cabot, and P.L. Miller. The mixing transition in Rayleigh-Taylor instability. *J. Fluid Mech.*, 511:333–362, 2004.
7. S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. C. Hart. Spectral surface quadrangulation. *ACM Trans. on Graphics (TOG) / Proc.of ACM SIGGRAPH*, 25(3):1057–1066, 2006.
8. H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. In *Proc. 19th Sympos. Comput. Geom.*, pages 361–370, 2003.
9. H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete Comput. Geom.*, 30:87–107, 2003.
10. M. Goresky and R. MacPherson. *Stratified Morse Theory*. Springer-Verlag, Heidelberg, Germany, 1988.
11. A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann. Topology-based simplification for feature extraction from 3D scalar fields. In *Proceedings of the IEEE Visualization 2005 (VIS'05)*, pages 535–542. IEEE Computer Society, 2005.
12. R. M. Haralick and L. G. Shapiro. Image segmentation techniques. *CVGIP*, 29(1):100–132, January 1985.
13. J. L. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, May/Jun. 1991.
14. M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In E. Fiume, editor, *Proceedings of ACM SIGGRAPH 2001*, pages 203–212, New York, NY, USA, 2001. ACM.
15. H. Hoppe. Progressive meshes. *Computer Graphics (Proc. SIGGRAPH '96)*, 30(4):99–108, Aug. 1996.
16. Y. Matsumoto. *An Introduction to Morse Theory*. American Mathematical Society, 2002.
17. J. C. Maxwell. On hills and dales. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, XL:421–427, 1870.
18. J. Milnor. *Morse Theory*. Princeton University Press, New Jersey, 1963.
19. L. R. Nackman. Two-dimensional critical point configuration graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(4):442–450, July 1984.
20. J. Pfaltz. Surface networks. *Geographical Analysis*, 8:77–93, 1976.
21. J. Pfaltz. A graph grammar that describes the set of two-dimensional surface networks. *Graph-Grammars and Their Application to Computer Science and Biology (Lecture Notes in Computer Science*, no. 73, 1979.
22. Y. Shinagawa, T. Kunii, and Y. L. Kergosien. Surface coding based on Morse theory. *IEEE Computer Graphics and Applications*, 11:66–78, 1991.
23. S. Takahashi, T. Ikeda, Y. Shinigawa, T. L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. In *Proc. Eurographics '95*, pages C–181–C–192, Sep. 1995.

24. S. Takahashi, Y. Shinagawa, and T. L. Kunii. A feature-based approach for smooth surfaces. In *Proc. of the Fourth Symp on Solid Modeling and Applications, SMA '97*, pages 97–110. ACM, May 1997.
25. M. J. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Symposium on Computational Geometry*, pages 212–220, 1997.
26. W. Warntz. The topology of a socio-economic terrain and spatial flows. *Regional Scientific Associations*, 17:47–61, 1966.