LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-463396

Robust Computation of Morse-Smale Complexes of Bilinear Functions

G. Norgard, P. T. Bremer

December 1, 2010

TopoInVis
Zurich, Switzerland
April 4, 2011 through April 6, 2011

Disclaimer

Chapter 1
Robust Computation of Morse-Smale Complexes of Bilinear Functions

Gregory Norgard and Peer-Timo Bremer

Abstract The Morse-Smale (MS) complex has proven to be a useful tool in extracting and visualizing features from scalar-valued data. However, existing algorithms to compute the MS complex are restricted to either piecewise linear or discrete scalar fields. This paper presents a new combinatorial algorithm to compute MS complexes for two dimensional piecewise bilinear functions defined on quadrilateral meshes. We derive a new invariant of the gradient flow within a bilinear cell and use it to develop a provably correct computation which is unaffected by numerical instabilities. This includes a combinatorial algorithm to detect and classify critical points as well as a way to determine the asymptotes of cell-based saddles and their intersection with cell edges. Finally, we introduce a simple data structure to compute and store integral lines on quadrilateral meshes which by construction prevents intersections and enables us to enforce constraints on the gradient flow to preserve know invariants.

1.1 Introduction

Topological structures such as contour trees [1, 2], Reeb graphs [3, 4], and Morse-Smale (MS) complexes [5, 6] have become a cornerstone of scalar field analysis and visualization. They provide a succinct description of the global behavior of a scalar field and in their latest incarnations the corresponding algorithms are highly efficient and numerically stable. However, the stability and efficiency of these techniques depends on the restriction to either piecewise linear or even discrete functions, particularly for MS complexes. While this makes the computations tractable it is unclear how valid such approximations are for typical simulation data. In practice, many of the largest and most advanced simulations use higher-order kernels, often on regular or block-regular grids. Unfortunately, there exist no techniques to reliably extract topological information from these interpolations creating a potentially significant gap between the data that is computed and the data that is analyzed.

In this paper we take a first step towards closing this gap by introducing a provably correct algorithm to compute MS complexes of two-dimensional bilinear functions defined on quadrilateral meshes. We derive the somewhat unintuitive behavior of integral lines of the bilinear interpolant, introduce a discrete algorithm to detect and classify critical points in bilinear functions, and describe a simple and efficient data structure to store the paths of integral lines through cells of the mesh. Similar to Bhatia et. al. [7], we construct maps from the cell boundaries to themselves encoding sufficient restrictions on the gradient flow to guarantee the validity of the resulting MS complex. Subsequent numerical integration is constrained by the maps and thus guaranteed to be valid and correct up to the numerical accuracy of the integration operator. Furthermore, as shown in Section 1.7, the maps themselves contain a surprisingly large amount of information about the

Gregory Norgard
Lawrence Livermore National Laboratory, Livermore CA 94551 e-mail: norgard3@llnl.gov

Peer-Timo Bremer
Lawrence Livermore National Laboratory, Livermore CA 94551 e-mail: bremer5@llnl.gov

gradient flow suggesting that the use of an "oracle" similar to that described in [2] could potentially extend this approach to even higher orders of interpolation. Finally, we show that level sets of saddles on the interior of cells are always aligned with mesh edges which introduces some unexpected artifacts. In particular, our experiments suggest that a piecewise bilinear interpolation while visually more pleasing creates on average more topological artifacts than its piecewise linear counterpart.

Related Work. As mentioned above, topological techniques have become popular in a large variety of applications and even a cursory review of all related research is beyond the scope of this paper. Instead, this section focuses on techniques designed for interpolations beyond piecewise linear, as well as general algorithms for MS complex computation. The most common use of bilinear and trilinear interpolation is in the marching cubes algorithm [8] and its many variations and corrections [9] [10]. Of particular interest for this paper is the asymptotic divider of Nielson and Hamann [11] which implicitly describes the gradient flow in cells containing saddles. In general, much of the research focused on topologically correct iso-surfaces [12] [13] can be seen as aimed towards the robust detection of critical points and the corresponding topology of the level sets [14]. Pascucci and Cole-McLaughlin [2] describe a general algorithm to compute the augmented contour tree encoding the topology of all level sets of a function defined on a regular grid. The approach is based on an oracle constructing the contour tree within each cell and the merging of neighboring trees. They describe the oracle for a piecewise trilinear interpolation, but the approach could be extended to other interpolation schemes. Nevertheless, the techniques discussed above focus on the structure of the level sets not on the gradient flow behavior.

While clearly related, gradient based structures such as the MS complex [15, 16] contain significantly more information than, for example, contour trees and are correspondingly more difficult to compute. Here we are using a variant of the tracing-based algorithm of Bremer et al. [5] extended to bilinear functions. By first detecting all critical points and tracing separatrices from saddles this algorithm constructs the MS complex iteratively maintaining the consistency explicitly through constraints in the data structure. More recently, Gyulassy and other [17, 6] introduced discrete MS complex algorithms which are based on a region growing approach. This approach is significantly easier to generalize to three and higher dimensions, but has the drawback of effectively reducing the interpolation further. There also exist a number of techniques aimed at computing vector field topology [18] that could be adapted to scalar topology. For example, Theisel et al. [19] use streamline tracing to compute saddle connectors in a three-dimensional potential field. However, these techniques rely on the numerical integration of streamlines, and thus are not guaranteed to produce valid and/or consistent results.

1.2 Theory

This section will briefly introduce the necessary mathematical background in Morse theory as well as formally define the piecewise bilinear interpolation and discuss some non-degeneracy conditions enabling robust computation.

Morse-Smale Complex. Given a smooth manifold $\mathbb{M}$ and a smooth function $f : \mathbb{M} \to \mathbb{R}$ the Morse-Smale (MS) complex segments $\mathbb{M}$ into regions of uniform gradient behavior. Let ∇f be the gradient of f then all points p on $\mathbb{M}$ are classified as either *critical* if $\nabla f(p) = 0$ or *regular* otherwise. Furthermore, a critical point p is call *non-degenerate* if the *Hessian*, the matrix of second partial derivatives, of f at p is not singular. If all critical points of f are non-degenerate and have pairwise distinct function value f is said to be *Morse*. Given a Morse function f an *integral line* $L(t) : \mathbb{R} \to \mathbb{M}$ of f is defined as a line whose tangent is aligned with the gradient of f: $\delta L/\delta t = \nabla f(L(t))$. For

$t \to \infty$ all integral lines converge towards a maximum where they are said to *end*. For each maximum p the union of all integral lines ending at p is called the *stable manifold* of p. Symmetrically, the *unstable manifold* of p is defined as the union of all integral lines starting at p. Intersecting the stable and unstable manifolds of f defines its MS complex a segmentation in which within each cell all integral lines start at the same minimum and end at the same maximum.

Piecewise Bilinear Functions. In this paper we are interested in two-dimensional manifolds defined by a quadrilateral mesh given as a collection of vertices V, edges E, and cells Q. The function f is defined at the vertices v and for brevity we will use v to indicate a vertex as well as the function value at that vertex depending on the context. For each cell q we define a function $f^q : q \to \mathbb{R}$ through *bilinear interpolation*. More specifically, consider the cell shown in Figure 1.1 with vertices a, b, c, and d. Since none of the computations depend on the embedding we assume wlg. $\mathbb{M} = \mathbb{R}^2$ and $q = [0,1] \times [0,1]$. We then define f^q as:

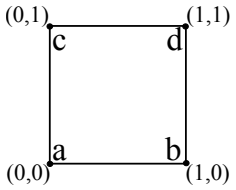


Fig. 1.1 The standard reference cell.

$$f^q(x,y) = (a-b-c+d)xy + (-a+c)y + (-a+b)x + a. \tag{1.1}$$

Finally, we define f as the collection of all $f^q, q \in Q$. Note that, by construction, the f^q's agree along shared edges/vertices and thus f is continuous. Finally, to avoid degeneracies we assume that no two vertices of the mesh have equal function value which can easily be enforced through symbolic perturbation [20].

Clearly, even a perturbed f is not Morse and thus does not define an MS complex. However, similar to the work on piecewise linear functions [15, 5] one can compute a quasi-MS complex of f by consistently detecting and classifying critical points and computing monotone, non-intersecting integral lines from saddles to extrema.

1.3 Bilinear Gradients

In order to consistently compute an MS complex of f, one must first understand the structure of f within each cell and in particular the behavior of f's gradient flow. This section will derive some general properties of the gradient flow within a bilinear cell q and discuss their implications for critical point detection and integral line computation. Section 1.4 will then show how to guarantee that these properties are preserved during the computation. Finally, we will prove in Section 1.6 that this leads to a consistent MS complex.

Zero Lines. By definition, the bilinear interpolation f^q within a cell q reverts to the linear interpolation along the edges of q. Since all vertices have pairwise distinct function values, f^q is strictly monotone along each edge and we indicate the direction of its gradient restricted to the edge by an arrow along the edge, see Fig. 1.2. Assuming the standard cell (the unit cube in $\mathbb{R}^2$, see Fig. 1.1) the direction of the gradient along the edges is equivalent to the sign of f^q_x and f^q_y on the respective edges. For example, if the gradients of edges (a,b) and (c,d) are aligned and positive, $f^q_x > 0$, then $f^q_x > 0$ everywhere on q. Conversely, if the gradients are anti-aligned: $f^q_x < 0$ on (a,b) and $f^q_x > 0$ on (c,d) then there exist a line with $f^q_x = 0$ separating the edges, which we will call a *zero-line*. Considering the gradient of f^q

$$\nabla f^q = \begin{bmatrix} (a-b-c+d)y + (-a+b) \\ (a-b-c+d)x + (-a+c) \end{bmatrix}, \tag{1.2}$$

it is clear that the zero-line is indeed a horizontal line (f_x^q is linearly interpolated depending only on y). We call the intersection of the zero-lines with edges *zero-points*. The definition of zero-lines leads to three possible cases of gradient behavior within q: (i) no zero-line; (ii) one zero-line; and (iii) two orthogonal zero-lines, see Fig. 1.2. One interesting property of zero-lines is that, since zero-lines of a bilinear function f^q are by definition orthogonal to gradients they are always level sets of f^q.

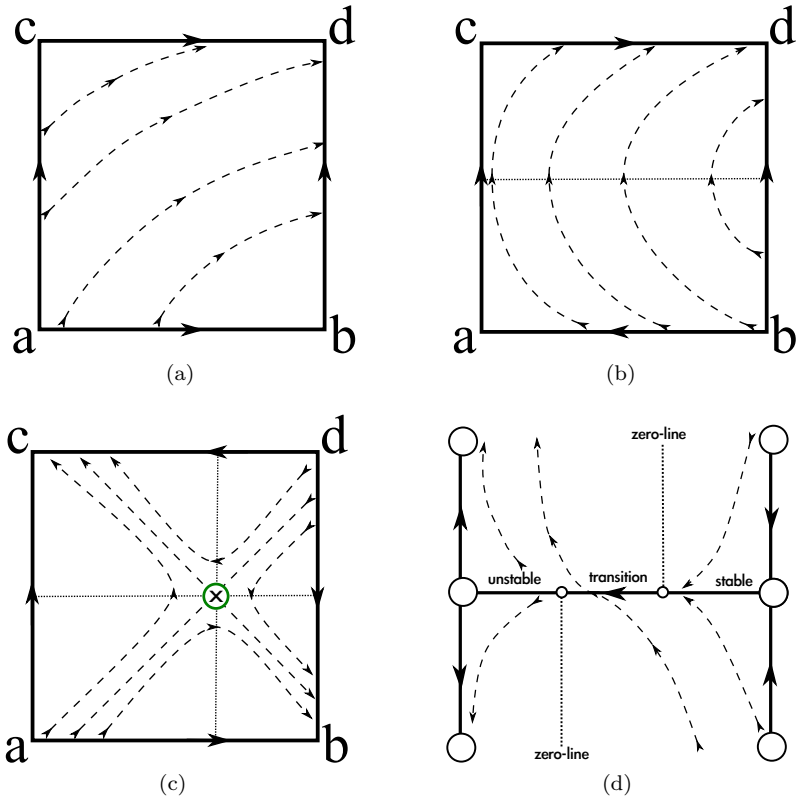


Fig. 1.2 The three different classes of bilinear cells containing: (a) no; (b) one; and (c) two zero-lines. Gradients restricted to edges are indicated by arrows; Approximate integral lines are shown as dashed and zero-lines are shown dotted.(d) Possible gradient flow across an edge with the different edge sections labeled.

Critical Points. The flow classification within a cell leads directly to a combinatorial critical point detection. Clearly, only cells with two zero-lines contain critical points which lie at the intersection of the zero-lines. Furthermore, as indicated in Fig. 1.2(c) the two pairs of anti-aligned edges force the critical point to be a saddle. Using symbolic perturbation there cannot exist a critical point on an edge. Thus, all other critical points of the combined f exist at vertices of the mesh and are detected and classified in the standard manner [15] using their neighboring vertices:

- A vertex with only higher neighbors is a minimum;
- A vertex with only lower neighbors is a maximum;
- A vertex switching between higher and lower neighbors more than twice is a saddle; and
- All other vertices are regular.

Besides the classification, zero-lines also imply another interesting fact for saddles:

Fact 1 *Given a bilinear function containing a saddle at* $(0,0)$ *the level sets containing this saddle are always the coordinate axes and its asymptotes are the* $y = -x$ *and* $y = x$ *lines.*

Practically, this implies that independently of the global structure of the level sets of f, level sets around any saddle in the interior of a cell are always axis aligned. As will be discussed in Section 1.7 this can create noticeable artifacts that, in fact, may be worse than those of a piecewise linear interpolation.

By definition, the asymptotes of a saddle form the first segments of the integral lines connecting saddles to extrema. Thus, determining where these integral lines intersect the boundary of a cell is crucial. Interestingly, the question of which integral line intersects which edge can be answered combinatorially.

Lemma 1. *Let* v *be a saddle of bilinear function* f^q *in the standard cell* q *then* v*'s asymptotes do not pass through vertices of* q*.*

Proof. Wlg. we assume $a < b$, $a < c$, $d < c$, and $d < b$ as shown Fig. 1.2(c). From equation(1.1) follows that v will be located at $x_{sad} = \frac{c-a}{a-b-c+d}$ and $y_{sad} = \frac{b-a}{a-b-c+d}$. For an asymptote to pass through a vertex we must have either $y_{sad} = x_{sad}$ or $y_{sad} = 1 - x_{sad}$. The former reduces to $b - a = c - a$ which violates our assumptions. Similarly, multiplying the later with denominator results in $b - a = (a - b - c + d) - c + a \Leftrightarrow 2b + 2c = 3a + d$. Since, $a, d < b, c$ this is also a contradiction.

The consequence of Lemma 1 is that through comparing vertices across the diagonals, a saddle can be assigned a unique closest edge in a cell. For example, in Fig. 1.2(c), the saddle is determined to lie closest to the right edge, by noting that $a < d$ and $b < c$. This comparison between diagonal vertices will also prove to be key in proving consistency in Section 1.5. Since the asymptotes are aligned with the diagonals, it follows that two asymptotes intersect the closest edge and one each of the two neighbor edges.

Cross-Edge Flow. Next we consider the different flow patterns across shared edges. Within a cell q, the gradient of f^q is smooth and continuous. However, the gradients of two neighboring cells p, q may not agree and thus f's gradient is generically discontinuous across edges. At any given point along an edge, the gradient field of a cell will either point into that edge (indicating outflow) or away from the edge (indicating inflow). The gradient can switch between inflow and outflow at most once and that switch will occur at zero-points. We classify a section of an edge based on the inflow/outflow combinations. If both cells indicate outflow this section of the edge is called *stable* (flow will converge and continue along the edge). If both cells indicate inflow, the section is called *unstable* (no flow can reach this section). Finally, if inflow and outflow are combined we consider this a *transition* section (flow will cross into the neighboring cell), see Fig. 1.2(d). Furthermore, a stable and unstable section will always have a direction associated with it depending on the values of the vertices of the edge. An edge can have at most three different section and generically a stable and unstable sections will always be separated by a transition section. Clearly, this classification reverses itself when considering the inverse flow.

However, if an edge contains two zero-points (one from each cell) it is difficult to decide their order along the edge combinatorially. This reduces to determining the sign of the determinant of a 3x3 matrix involving all six vertices of both cells, which would require a second-order simulation of simplicity [20]. Fortunately, as will be discussed in more detail later, the ordering among zero-points does not effect the consistency of the algorithm. Therefore, we simply decide the ordering numerically and break ties with a random choice.

Cell-Based Flow. As discussed above, the flow on the interior of a cell can be classified into the three cases of no, one, or two zero-lines. The generic flow pattern for the first and last case follows directly from the flow pattern at the edges, see Fig. 1.2(a), 1.2(c). The case of a single zero-line, however, is governed by an interesting and not necessarily intuitive restriction:

Lemma 2. *Given a bilinear function f^q on the standard cell q with a single zero-line connecting edges (a,c) and (b,d) no integral line exists that intersects both edges.*

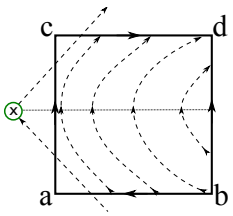


Fig. 1.3 And illustration of a cell with one zero-line.

Proof. Wlg. assume that $d < c$, $c < a$, $a < b$, and $d < b$, as shown in Fig. 1.3 and consider the cells interpolant to extend beyond the boundary of the cell. Since there exists a zero-line, there must exist a saddle s contained along this line. Furthermore, as indicated by the integral lines in Fig. 1.3 the assumptions guarantee that s lies on the left side of q. The asymptotes of s form a 45 degree angle with the zero-line and thus cannot cross (b,d). By construction, the asymptotes are the lines where $|f_x^q| = |f_y^q|$ and thus in the region between two asymptotes we have either $|f_x^q| < |f_y^q|$ or $|f_x^q| > |f_y^q|$. Under the current assumptions the edge (b,d) lies completely within a region where $|f_x^q| < |f_y^q|$. No integral line can cross an asymptote (which is just another integral line) and therefore all integral lines crossing (b,d) must fulfill $|f_x^q| < |f_y^q|$ everywhere along their path. Combining this with the fact that q is the standard cell with unit width and unit height, any integral line crossing (b,d) must either cross (a,b) or (c,d).

Lemma 2 seems counter-intuitive since one can image stretching a cell in one direction thus making it more "likely" for an integral line to cross. However, any such cell can always be bijectively mapped into the standard cell without effecting the structure of the integral lines. This lemma encapsulates one of the most crucial constraints on the path of integral lines in bilinear flow and will be key to prove the consistency of the algorithm in Section 1.5

1.4 Data Structure

As discussed above computing an (quasi)-MS complex requires reliable critical point detection as well as the ability to compute non-crossing, monotone integral lines. This section describes a simple data structure that guarantees integral lines do not cross. Section 1.5 will then show how to initialize this structure with a number of carefully chosen integral lines to ensure all integral lines are consistent with the theory established in Section 1.2.

We represent (pieces of) integral lines as a list of their intersections with mesh edges or vertices. Each intersection is stored as a *tracing element* each of which can have a forward and a backward pointer to another tracing element. If the forward/backward pointer of a tracing element is set the forward/backward path of the integral line is considered known and will not be recomputed. Tracing elements are stored in vertices and in ordered lists on edges, in which case each tracing element contains its geometric location along the edge. The ordering explicitly enforces the fact that no two tracing elements can exist at the same place even though their geometric location might be numerically identical. During the integral line computation algorithm discussed below, the order information will supersede the geometric information and, if necessary, the geometry will be adapted accordingly. Finally, there exist geometric locations, e.g. zero-points and saddle asymptotes, in which any perturbation of the starting point will cause fundamental changes in the path of an integral line. In these situations, we will use multiple tracing elements in the same

geometric location each storing one possible outcome. Since these groups of elements represent a single (unstable) location, they are considered *linked* and no other integral line is allowed to be inserted between them.

Since edges are shared between cells, so are the corresponding tracing elements. However, for a cell q, only tracing elements initialized by q or traced through q are *relevant* to that cell. For example, zero-points of neighboring cells are not relevant to the gradient flow within q. Therefore, we store a flag with each element indicating whether it is relevant for one or both of its neighboring cells.

1.5 Algorithm

Using the critical point detection discussed in Section 1.2, this section describes how to compute integral lines from saddles to extrema to create the MS complex. Similar to [21], we first compute all ascending integral lines from saddles to maxima and then, using the inverse gradient, all descending lines from saddles to minima.

The algorithm first finds all saddles and initializes their neighboring cells, see below. This creates four tracing elements for each saddle each with either the forward or backward pointer unset. The algorithm proceeds by picking one of these tracing elements and computing its next step. Each time the algorithm reaches an un-initialized cell it initializes the cell before continuing. The integral line stops once an extremum is reached, see Algorithm 1. Below, we first describe how cells are initialized and then how tracing elements are propagated.

Cell initialization. Cells are initialized by splitting their boundary into inflow and outflow sections. By definition, this switch can only occur at zero-points and vertices. For vertices, we create two tracing elements on the two neighboring edges and set their corresponding pointers according to the gradient flow, see Fig. 1.4(a). For zero-points, we can determine combinatorially whether the integral of this point lies in the interior of the cell. If not, we link the zero-point to itself. Otherwise, we compute the integral line of the zero-point forward and backward through the cell and insert the corresponding tracing elements. Note that Lemma 2 requires both intersection points to lie on neighboring rather than opposite edges and, if necessary, we enforce this condition explicitly. Furthermore, the new intersections are unstable in the following sense. Considering Fig. 1.4(b) any integral line entering on the right of the lower zero-point will cross the cell, while all integral lines on the left of the zero-point will exit through the left edge. To handle this instability and enforce consistency with the known gradient structure, we duplicate each newly inserted tracing element creating two linked pairs. For cells containing a saddle, we also insert tracing elements for its asymptotes, which will be included in the starting points for the integral line computation algorithm, see Fig. 1.4(c). As discussed in Lemma 1, we can decide which edges of the cell intersect an asymptote combinatorially and add the corresponding tracing elements. Similar to the integral lines from zero-points, the tracing elements for asymptotes represent an instability. Therefore, we create a linked triplet for each tracing element associated with an asymptote and connect them according to the flow, see Fig. 1.4(c).

One important aspect of the initialization is that it implicitly creates a map between inflow and outflow sections of the cell boundary. Each inflow section is bounded by two tracing elements whose forward pointers lead to two elements bounding its corresponding outflow section. Furthermore, by construction, any integral line that observes this map is consistent with the given set of zero-lines, cell-based saddles, and most importantly Lemma 2.

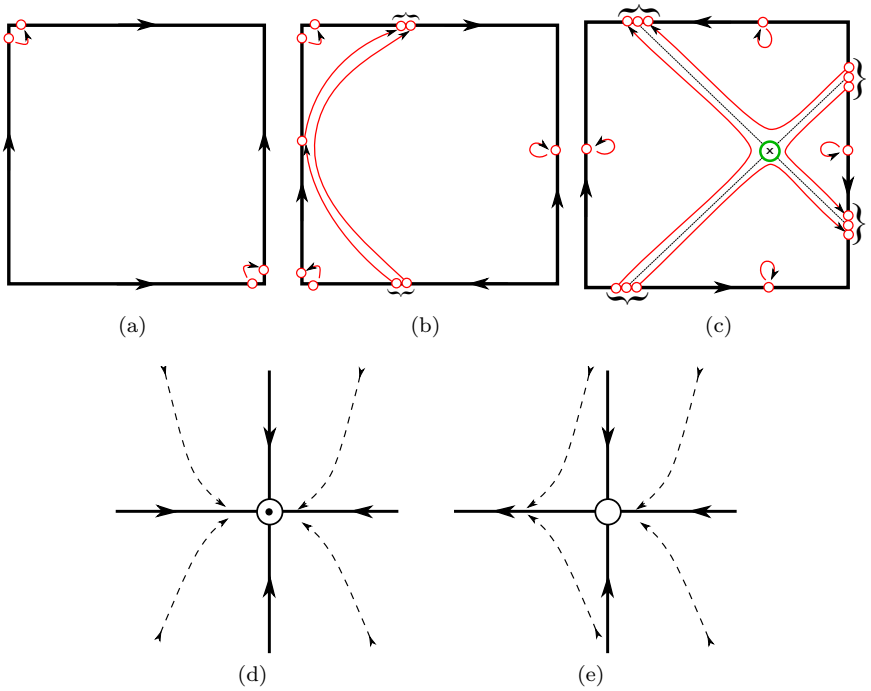


Fig. 1.4 (a)-(c)Cell initialization for the three types of cells. Arrows along edges represented the corresponding restricted gradient and circles the tracing elements. Curved brackets indicate linked groups of tracing elements. (a) No zero-line; (b) One zero-line; (c) Two zero-lines. (d),(e) Possible flow patterns for vertices with incoming stable sections. (d) Maximum; (e) Regular vertex.

Integral Line Propagation. An integral line is computed by selecting a tracing element starting at a saddle and progressively advancing the integral line through the mesh by either creating new tracing elements or following along existing pointers. At any time, we initialize cells in a lazy manner before any integral line computation is done within a cell. In the following we describe the integral line computation algorithm for an ascending line and the descending case is handled symmetrically.

If a tracing element t already has a forward pointer we follow this pointer until we either find an extremum or an element that has not been propagated. An element without a forward pointer can exist either in the interior of an edge or on a vertex. If on an edge, such a t must lie either in an inflow or an outflow section, since all zero-points already have forward pointers from the initialization.

Assume t lies on an inflow section, see Fig. 1.5(a): We find its right and left neighboring tracing elements relevant to the current cell (ignoring zero-points and untraced elements from neighboring cells). Following the forward pointers of these elements leads to (part of) an outflow section, see Fig. 1.5(b). By construction, the current tracing element must end within this outflow section. Using any preferred numerical technique, we compute the location of the integral line's forward intersection with the cell boundary (Fig. 1.5(c)) and insert the corresponding tracing element into the edge list, see Fig. 1.5(d). If for any reason the numerically computed intersection lies outside of the predetermined outflow section (Fig. 1.5(e)), we correct its location to the nearest valid position, see Fig. 1.5(f). We then continue the integral line computation with the newly created tracing element in the neighboring cell. Notice that, when computing ascending lines no tracing element can reach an inflow section of that cell.

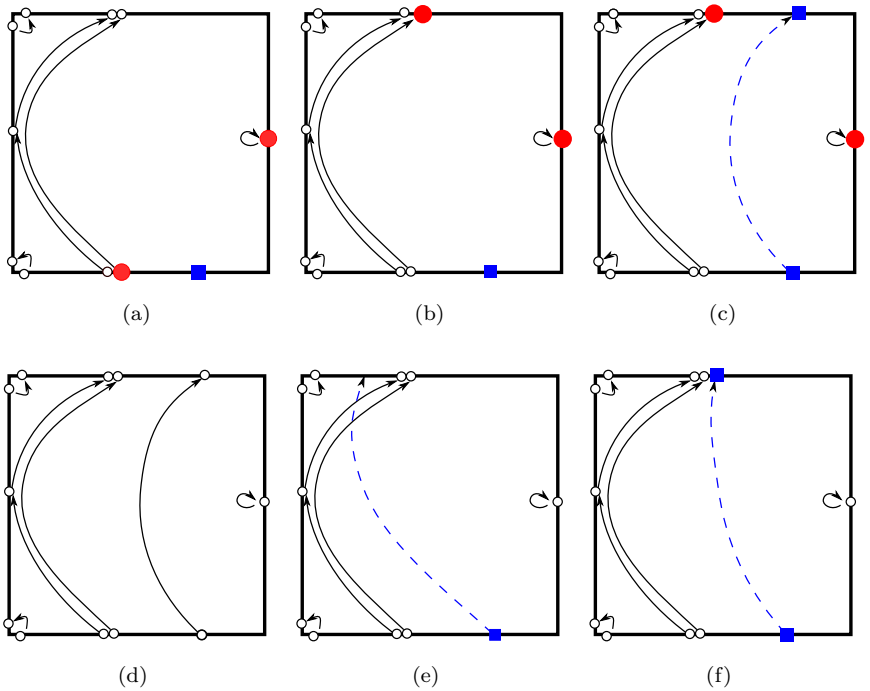


Fig. 1.5 An illustration of how the integral line computation interacts with the data structure. (a) When computing the integral line of a tracing element (blue square), find its left and right neighbors along the cell's boundary (red circles). (b) Follow the neighboring elements forward pointers to find the elements (red circles) that bound your potential outflow section of the cell's boundary. (c) Numerically compute the gradient to the boundary and insert a new tracing element (blue square) into that edge's ordered list. Assign backwards and forwards pointers appropriately. (d) The integral line now further subdivides the cell's boundary. (e) If a tracing element is numerically determined to trace outside of its valid outflow section in the cell's boundary, (f) then the numerical computation is corrected to the furthest point in the outflow section and a new tracing element is inserted there.

If t lies on an outflow section it follows that this section is part of a stable section of that edge (otherwise t could not have reached this point). Each stable section has an associated direction and t is connected to the appropriate endpoint of this section. By construction, this endpoint is either a vertex or a zero-point, see Fig. 1.2(d). The former case will be treated next, and in the later case the zero-point has a forward pointer which we follow. Stable sections are the only places where ascending integral lines merge. Since these sections are unstable for the inverse flow, they cannot be reached by descending lines. Therefore, ascending and descending lines cannot merge. For general meshes, however, they can touch at vertices as discussed below.

By construction, integral lines only reach vertices through stable edge sections. For regular (all valence four) grids only maxima and regular vertices can have stable inflow sections attached to them, see Fig. 1.4(d). Furthermore, any regular valence four vertex with an incoming stable section must also have an outgoing stable section, which represents the only outflow, see Fig. 1.4(e). Therefore, in regular grids integral lines pass through vertices along stable edge sections. Consequently, on regular grids ascending and descending lines can never merge since no vertex can have both stable inflow and unstable outflow (stable inflow of the inverse gradient). On meshes with higher valency vertices ascending and descending integral lines can touch at vertices with both incoming stable and outgoing unstable edge section. However, on edges and in the interior of cells ascending and descending lines remain disjunct. Therefore, intersection of ascending and descending lines can

be avoided by querying the local neighborhood of a vertex. Since the mesh implies an orientation, preventing intersections is straight forward. The complete pseudo-code of the algorithm on a regular grid is shown below.

Algorithm 1 void TraceForward(TracingElement Element)

```
if Element had previously been traced. then
  TraceForward(Element → forward)
end if
if Element is on a edge then
  if  Element is on a transition section then
    Trace through a bilinear cell. Insert a new tracing element in the boundary of the cell. Set Element's forward
    pointer to the new tracing element
    TraceForward(Element → forward)
  else
    Trace on a stable section to a zero-point or vertex and set Element's forward pointer to that tracing element.
    TraceForward(Element → forward)
  end if
else
  if Vertex is a maximum then
    RETURN
  else
    Find the stable edge leading out, and trace to the next zero-point or vertex on that edge.
    TraceForward(Element → forward)
  end if
end if
```

1.6 Proof of correctness

In order to guarantee that the MS complex is topologically correct the computed integral lines must be monotonic and non-crossing. The non-crossing of integral lines has been addressed by the data structure. The monotonicity of an integral line is a numerical calculation, which is susceptible to floating point error may will not reflect the symbolic perturbation we have employed with simulation of simplicity. Thus we prove that integral lines are monotonic with respect to the data structure describing the function. This will prove that integral lines terminate at extrema and do not cycle.

We establish this monotonicity property by assigning each tracing element a value and proving each ascending integral traverses them in an increasing manner. On an edge, a tracing element has value equal to the sum of the edges' neighboring vertex values, and on a vertex has value twice that of the vertex. We then establish the following lemma.

Lemma 3. *If a integral line connects two edges of a cell, then the sum of the vertices of the edge where the integral line terminates is greater than the sum of the vertices of the edge where the integral line originates.*

Proof. Consult Figs. 1.2 and 1.4 for clarity. If an integral line connects neighboring edges, this reduces to comparing the values of vertices across diagonals. In the no and one zero-line cases, the vertices on the terminating edge will have a greater sum for any integral line connecting neighboring edges. This can be seen by comparing values of vertices connected by edges. For two zero-lines, the diagonal vertex comparison is not immediately clear, however from Lemma 1, the saddle point's closest edge is determined from the diagonal vertex comparison. In Fig. 1.2(c) for the right edge to be closest to the saddle point $a < d$ and $b < c$. The result for the sum of the vertices follows.

An integral line can also connect opposite edges. The lemma is clear for the no zero-line case again by comparing values of vertices connected by edges. With one zero-line, due to the cell initialization and integral line computation maintaining Lemma 2, the only opposite edge connection is across the zero-line, where the sum of the terminating edge's vertices is greater than that of the originating edge's. For the two zero-line case, the saddle's asymptotes allow only one valid opposite edge connection and with the diagonal vertex comparison from above, the sum of the terminating edge's vertices is again greater.

This lemma leads directly into the following theorem.

Theorem 1. *The integral line computation algorithm 1 will create a monotonically increasing integral line and cannot cycle.*

Proof. A integral line computed by algorithm 1, will be a sequence of tracing elements from edge to edge, vertex to edge, and edge to vertex. Lemma 3 establishes that edge to edge transition increases the value of the tracing element. Vertex to edge and edge to vertex transitions as previously described will trivially increase the tracing element value. Thus the integral line's tracing elements will have a monotonically increasing value. As the vertex values are well ordered by simulation of simplicity, this prevents the integral line from visiting an edge again after visiting a different edge and thus cannot cycle.

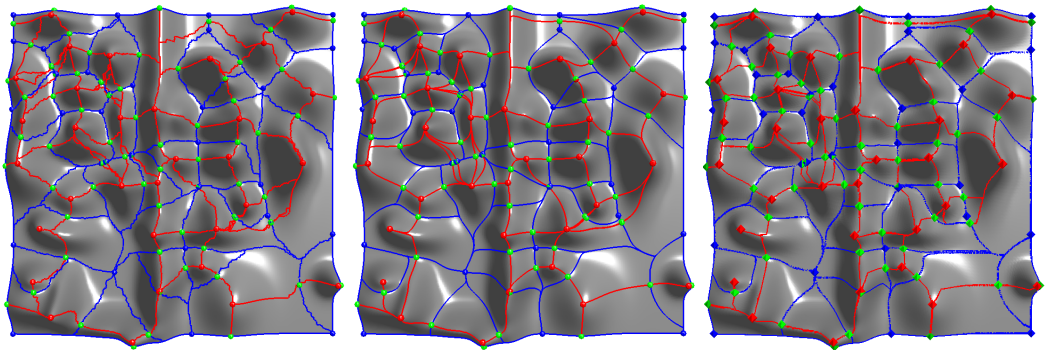
Thus with monotone, non-crossing integral lines we are guaranteed to construct a valid Morse-Smale complex.

1.7 Results

To evaluate the bilinear MS complex computation we first test the robustness of the bilinear technique to numerical error, by comparing two numerical integral line computations. We also compare the resulting MS complexes to those computed using the piecewise linear technique in [5]. Three separate data sets are utilized: an artificial terrain composed on superimposed Gaussians, a Finite Time Lyapunov Exponent (FTLE) field for simulated fluid flow around a cylinder, and a circular ridge.

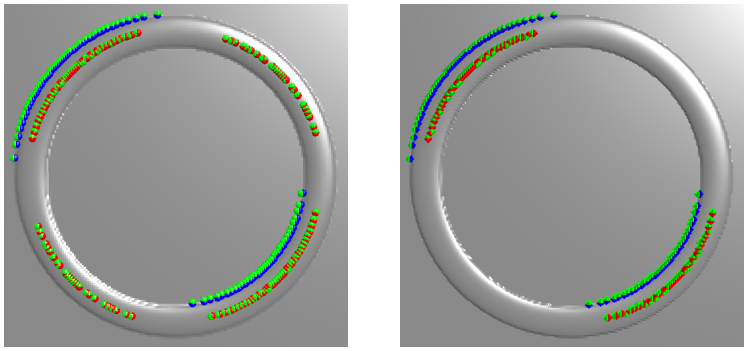
To test robustness, the first integral line computation will output a random location on the cell boundary and rely on the data structure for correction. The second method computes an analytic solution to integral line and then uses bisection to determine the intersection with the cell boundary. Figures 1.6(a) and 1.6(b) show the results on the Gaussian terrain. The random integral line computation produces jagged integral lines, as expected, yet the two structures are surprisingly similar. This demonstrates the algorithm's robustness to numerical error and suggests that a consistent mapping of inflow/outflow regions of cells may be sufficient for creating valid MS complexes in higher order computations.

Next we compare the bilinear and piecewise linear MS complexes. To the authors' knowledge, no rigorous comparison of MS complexes exist, so the comparison is made visually. Both methods were used on the Gaussian terrain (Fig. 1.6(b), 1.6(c)) and the FTLE field (Fig. 1.8(a), 1.8(b)). Visually the methods produce quite similar results with the integral lines of the bilinear method appearing smoother as one would expect from a higher order method. Additionally, we find that the bilinear method, on average, will produce more artifacts in terms of additional critical points, specifically along ridges. Figs. 1.7(a) and 1.7(b) show the critical points detected with bilinear and piecewise linear interpolation. Both methods create additional critical points when the ridge does



(a) (b) (c)

Fig. 1.6 Comparison of bilinear and piecewise linear MS complexes. (a) The MS complex with bilinear interpolation when the integral lines are computed with a random walk computation. (b) The Morse-Smale complex with bilinear interpolation with the integral lines computed analytically and intersected with the boundary with bisection. (c) The Morse-Smale complex computed with piecewise linear interpolation, computed as in [5].



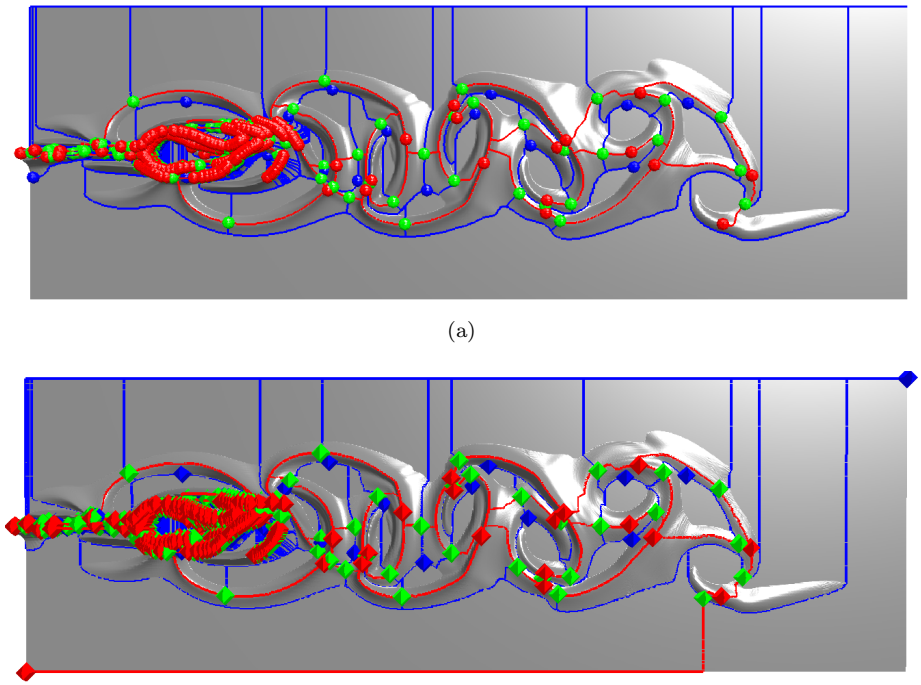
(a) (b)

Fig. 1.7 (a) Critical points when a circular ridge is interpolated bilinearly. (b) Critical points when a circular ridge is interpolated linearly.

not align with the mesh, however the piecewise linear method uses a triangular mesh which aligns with ridges more often than the bilinear quadrilateral mesh.

Summary. In this work, we have investigated the integral line behavior within bilinear cells. Using these results we have developed a robust algorithm that combines combinatorial critical point detection with a data structure that can facilitate and correct any numerical integral line computation. We have proven that using these techniques we preserve the monotone and non-crossing integral line properties that guarantee a valid topological Morse-Smale complex for a bilinearly interpolated function.

Examining the implementation of the algorithm we find it robust to errors in the numerical computation of integral lines and produces surprisingly reasonable results when given random integral line data. When given a numerically reasonable integral line computation, we find that the bilinear Morse-Smale algorithm produces more critical points and smoother integral lines than the piecewise linear approach. Overall the technique is appealing for its higher order approximation and visually smoother integral lines.



(a)

(b)

Fig. 1.8 The Morse-Smale complex of the FTLE field using two different methods. (a) The simplified MS complex for the bilinearly interpolated FTLE field. (b) The simplified MS complex for the piecewise linearly interpolated FTLE field. The general structure of the two methods appear to be visually similar. The simplification canceled critical points with lower than 1.5% persistence.

References

1. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.*, 24(3):75–94, 2003.
2. V. Pascucci and K. Cole-McLaughlin. Parallel computation of the topology of level sets. *Algorithmica*, 38(1):249–268, Oct. 2003.
3. Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of Reeb graphs: Simplicity and speed. *ACM Transactions on Graphics*, 26(3):58.1–58.9, 2007. Proceedings of SIGGRAPH 2007.
4. W. Harvey and Y. Wang. Generating and exploring a collection of topological landscapes for visualization of scalar-valued functions. In G. Melançon, T. Munzner, and D. Weiskopf, editors, *Proc. Symposium on Visualization*, volume 29, page to appear. Eurographics/IEEE-VGTC, 2010.
5. 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.
6. A. Gyulassy, P.-T. Bremer, V. Pascucci, and B. Hamann. A practical approach to Morse-Smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1619–1626, 2008.
7. Harsh Bhatia, Shreeraj Jadhav, Peer-Timo Bremer, Guoning Chen, Joshua A Levine, Luis Gustavo Nonato, and Valerio Pascucci. Edge maps: Representing flow with bounded error. Accepted to IEEE Pacific Visualization Symposium, 2011.

8. W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics (Proc. SIGGRAGPH '87)*, 21(4):163–169, July 1987.
9. Claudio Montani, Riccardo Scateni, and Roberto Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, 10(6):353–355, 1994.
10. Adriano Lopes and Ken Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9:16–29, 2003.
11. G. M. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In G. M. Nielson and L. J. Rosenblum, editors, *Visualization '91*, pages 83–91, Los Alamitos, California, Oct. 1991. IEEE, IEEE Computer Society Press.
12. Allen van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Trans. Graph.*, 13(4):337–375, 1994.
13. E. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical report, 1995.
14. G. Weber, G. Scheuermann, H. Hagen, and B. Hamann. Exploring scalar fields using critical isovalues. In M. Gross, K. I. Joy, and R. J. Moorhead, editors, *Proc. IEEE Visualization '02*, pages 171–178, Los Alamitos California, 2002. IEEE, IEEE Computer Society Press.
15. H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete Comput. Geom.*, 30:87–107, 2003.
16. 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.
17. 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.
18. G. Scheuermann and X. Tricoche. Topological methods in flow visualization. In *Visualization Handbook*, pages 341–356. Elsevier, 2004.
19. H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P.Seidel. Saddle connectors - An approach to visualizing the topological skeleton of complex 3d vector fields. In *Proc. IEEE Visualization '03*, pages 225–232, Los Alamitos California, 2003. IEEE Computer Society Press.
20. H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics (TOG)*, 9:66–104, 1990.
21. P.-T. Bremer and V. Pascucci. A practical approach to two-dimensional scalar topology. In H. Hauser, H. Hagen, and H. Theisel, editors, *Topological-based Methods in Visualizations*, pages 151–171. Springer Verlag, 2007.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.