Eurographics Conference on Visualization (EuroVis) 2012
S. Bruckner, S. Miksch, and H. Pfister
(Guest Editors)

Volume 31 (2012), Number 3

Parallel Computation of 3D Morse-Smale Complexes

Nithin Shivashankar [1] and Vijay Natarajan [1,2]

[1]Department of Computer Science and Automation, Indian Institute of Science, Bangalore-12, India
[2]Supercomputer Education and Research Center, Indian Institute of Science, Bangalore-12, India

Abstract

The Morse-Smale complex is a topological structure that captures the behavior of the gradient of a scalar function on a manifold. This paper discusses scalable techniques to compute the Morse-Smale complex of scalar functions defined on large three-dimensional structured grids. Computing the Morse-Smale complex of three-dimensional domains is challenging as compared to two-dimensional domains because of the non-trivial structure introduced by the two types of saddle criticalities. We present a parallel shared-memory algorithm to compute the Morse-Smale complex based on Forman's discrete Morse theory. The algorithm achieves scalability via synergistic use of the CPU and the GPU. We first prove that the discrete gradient on the domain can be computed independently for each cell and hence can be implemented on the GPU. Second, we describe a two-step graph traversal algorithm to compute the 1-saddle-2-saddle connections efficiently and in parallel on the CPU. Simultaneously, the extrema-saddle connections are computed using a tree traversal algorithm on the GPU.

Categories and Subject Descriptors (according to ACM CCS):
I.3.5 Computational Geometry and Object Modeling

1. Introduction

The Morse-Smale (MS) complex partitions the domain of a scalar function into regions with uniform gradient behavior. It provides an abstract representation that enables multi-scale analysis and visualization of 2D and 3D scalar functions [LBM*06, GDN*07, RKG*11, KRHH11]. Motivated by the increasing data sizes, recent approaches towards the computation of MS complexes focus on memory efficiency and scalability in addition to performance [GBPH08, GRWH11, PRG*11]. We present a parallel shared-memory algorithm to compute the MS complex of a three-dimensional scalar function. Our algorithm makes a synergistic use of the multi-core CPU and GPU of a desktop computer to compute the MS complex of large three-dimensional data sets. Experiments indicate that this approach can process data consisting of over 1 billion vertices within minutes while utilizing less than 2GB of memory.

1.1. Related work

MS complexes were initially introduced in the context of dynamical systems [Sma61a, Sma61b]. Later, Edelsbrunner et al. [EHZ03] studied the problem of computing the MS complex for two-dimensional piecewise linear functions. They computed the cells of the MS complex while restricting the bounding arcs of the complex to edges of the input mesh. This approach was extended to construct MS complexes of three-dimensional functions [EHNP03]. These early approaches traced the gradient paths from saddle critical points and produced a boundary representation of cells in the MS complex. Bremer et al. [BEHP04] also followed this approach and developed a multi-resolution representation of 2D scalar functions via controlled topological simplification, and demonstrated the application of the MS complex to feature identification, noise removal, and view-dependent simplification.

Gyulassy et al. [GDN*07, GNP*06] focused on three-dimensional functions and employed an approach based on repeated cancellations of critical point pairs applied on an artificial complex created from the input mesh by including dummy critical points. The cancellations were appropriately scheduled in order to remove the dummy critical points leaving behind the true critical points and cells of the MS complex. The order of cancellation determines the quality of the resulting MS complex and the algorithm efficiency.

Another approach to compute the MS complex is based on a discrete analog of Morse theory [Mil63] introduced by Forman [For02] to study discrete functions defined on cells of a cell complex. Reininghaus et al. [RH11, RLH11] discussed an application of discrete Morse theory to analyze vector fields. Bauer et al. [BLW11] computed simplified two-dimensional scalar functions while ensuring that the input function is modified by no more than a threshold δ and all surviving critical point pairs have persistence greater than 2δ. Cazals et al. [CCL03] and Lewiner et al. [LLT04] successfully employed Forman's discrete Morse theory to compute MS complexes of piecewise-linear functions and demonstrated applications to segmentation, visualization, and mesh compression. Gyulassy et al. [GBPH08] also used a discrete Morse theory based formulation to develop an efficient algorithm for computing MS complexes of large 3D data that do not fit in main memory. They partition the data into blocks called "parcels" that fit in memory, compute gradient flows on the boundary of the parcels, propagate the flows to the interior and compute the MS complex restricted to a parcel. Critical cells that are created on the boundary are canceled during a subsequent merge step resulting in the MS complex of the union of the parcels. This serial method scales well for large data but the geometry of the MS complex is sensitive to the order of cancellations chosen during the merge step.

Robins et al. [RWS11] proposed an algorithm to compute the Morse complex of 2D and 3D gray-scale digital images modeled as discrete functions on cubical complexes. The algorithm computes the Morse complex with provable guarantees on its correctness with respect to the critical cells. However, the algorithm does not guarantee polynomial time execution since they use a modified breadth first search algorithm that traverses all possible paths between two nodes, which can be exponential in the number of nodes.

More recently, Peterka et al. [PRG*11] introduced a set of building blocks for implementing parallel algorithms, which leverage high performance computing clusters. In particular, they discuss a parallel implementation of the discrete Morse theory based algorithm proposed by Gyulassy et al. [GBPH08] using their framework. Unlike the work of Peterka et al., we focus on a parallel implementation on a desktop computer. Günther et al. [GRWH11] described a memory efficient algorithm to compute the MS complex for 3D data and use the complex to compute persistent homology groups. The discrete gradient field is computed using a parallel variant of the method proposed by Robins et al. followed by an efficient computation of the boundary map that represents the MS complex. They employ a modified breadth first search based algorithm to traverse the gradient field, the maintained of visited flag per cell for each traversal. Thus the number of traversals that can be launched in parallel is limited by available memory. In comparison, our parallel algorithm for computing the discrete gradient field is based on a novel design of the discrete Morse function followed by a two-step algorithm to compute the cells of the MS complex. Our hybrid multi-core approach for implementing this algorithm results in a method that is fast in addition to being memory efficient.

1.2. Contributions

The above mentioned methods for computing the MS complex are either slow or not easily applicable because of one or more of the following reasons: (a) they compute and trace the gradient serially, (b) they do not guarantee that they trace the correct geometry of the gradient flow, or (c) they require specialized computing resources such as HPC clusters.

Our contributions to solve the above problems for scalar functions on three-dimensional structured grids is based on a characterization of MS complexes using Forman's discrete Morse theory that leads to a parallel algorithm. We describe an extension of the parallel discrete gradient field construction algorithm introduced earlier [SMN11] to compute 2D MS complexes. The presence of two types of saddle criticalities in 3D introduces considerable complexity into the geometry of the MS complex. The discrete gradient paths between saddles may split and merge causing an explosion in the number of paths to be traced. Moreover, the data sizes are much larger compared to 2D scalar functions. The key contributions of this paper towards efficient computation of 3D MS complexes are

- An extension of the independent gradient pairing algorithm discussed in [SMN11] that significantly reduces the number of cells that remain unpaired, thereby producing fewer artifacts in the form of spurious critical cells.
- An efficient two-step algorithm to compute gradient paths between 1-saddles and 2-saddles.
- Synergistic use of CPU and GPU to ensure maximum utilization of computational resources.
- Implementation of a split and merge approach that supports computation of MS complexes for large datasets.

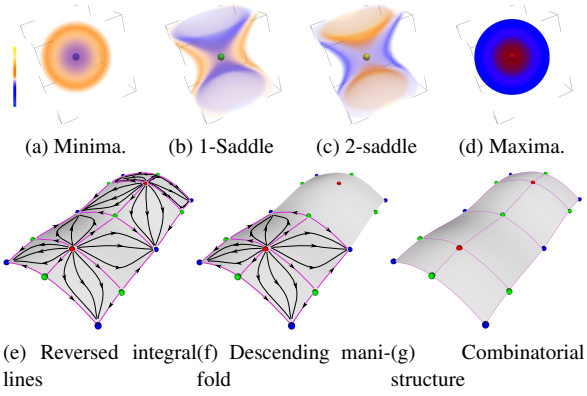
We present experimental results on both synthetic and real-world data sets to demonstrate the efficiency of the algorithm in terms of performance and memory requirement.

2. Background

This section reviews the necessary background on Morse functions and discrete Morse functions required for the algorithm description. While the focus of the paper is on 3D domains, we often consider 2D domains for ease of illustration and to describe prior work.

2.1. Morse functions

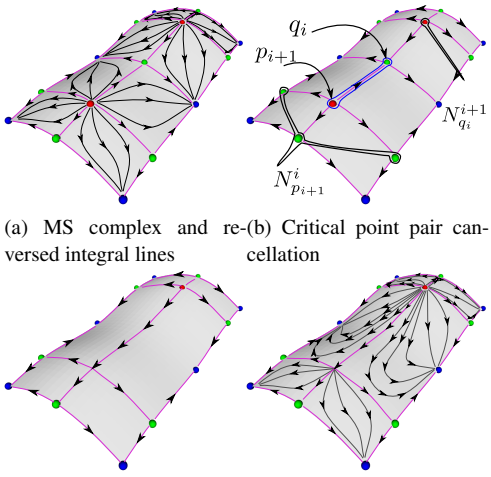
Consider a smooth scalar function $f : \mathbb{R}^3 \to \mathbb{R}$. A point $p \in \mathbb{R}^3$ is called a *critical point* with respect to f if the *gradient* of f, ∇f, is identically zero at p. A critical point is non-degenerate if the *Hessian* of f, equal to the matrix of second



(a) Minima. (b) 1-Saddle (c) 2-saddle (d) Maxima.

(e) Reversed integral lines (f) Descending manifold (g) Combinatorial structure

Figure 1: (a)-(d) Behavior of a Morse-Smale function near the critical points in a 3D domain. (e) The reversed integral lines of a 2D function. (f) The descending manifold of a maximum shown as the closure of the set of reversed integral lines that originate from the critical point. (g) Combinatorial structure of the MS complex where nodes are critical points and connecting integral lines are arcs.



(a) MS complex and reversed integral lines (b) Critical point pair cancellation

(c) Combinatorial realization (d) Geometric realization

Figure 2: (a) MS complex for a simple height function. (b) Canceling a pair of critical points, q_i, p_{i+1}, of index i, $i+1$ that are connected by a single arc. (c) Combinatorial realization: connect all index i critical points ($N^i_{p_{i+1}}$) that are connected to p_{i+1} except q_i, to index $i+1$ critical points ($N^{i+1}_{q_i}$) that are connected to q_i except p_{i+1}. (d) Geometric realization: compute the union of the descending manifold of p_{i+1} with the descending manifolds of all index $i+1$ critical points connected to q_i. Compute the union of the ascending manifold of q_i with the ascending manifolds of all index i critical points connected to p_{i+1}.

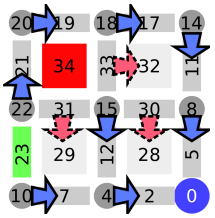
order partial derivatives, is non-singular. We call f a *Morse function* if all of its critical points are non-degenerate. The *index* of a critical point is the number of negative eigenvalues of the Hessian matrix. An *integral line* passing through a point p is a maximal curve in the domain, whose tangent at every point equals the gradient of f at that point. The function f increases along the integral line and its limit points are the critical points of f.

The set of all integral lines that share a common source together with the point p, is called the *ascending manifold* of p and the set of all integral lines that share a common destination together with the point p is called the *descending manifold* of p, see Figure **??**. The ascending manifolds (similarly, the descending manifolds) of all critical points partition the domain. The *Morse-Smale complex* is a partition of the domain into cells formed by the collection of integral lines that share a common source and a common destination. The ascending manifold of a critical point of index d is a $(n-d)$-dimensional manifold, where as its descending manifold is a d-dimensional manifold. A Morse function f is called a *Morse-Smale function* if all ascending and descending manifolds of two critical points intersect transversally. Thus, if the index of two critical points differ by one then their ascending / descending manifolds either do not intersect or intersect along a one-dimensional manifold connecting the critical points. The critical points, referred to as *nodes*, together with the 1-manifolds that connect them, referred to as *arcs*, form the 1-skeleton of the MS complex, which is referred to as the combinatorial structure of the complex.

2.2. Simplification

A Morse-Smale function f can be simplified to a smoother function by repeated application of a cancellation operation that removes a pair of critical points connected by an arc in the MS complex. Critical point pairs correspond to topological features and are ordered based on the notion of *persistence*, which is equal to the absolute difference in function value between the two critical points. Persistence measures the importance of a critical point pair [ELZ02]. The least persistent critical point pair is always connected by an arc in the MS complex [EHZ03].

Cancellation of a pair of critical points is achieved by a local smoothing of the function within the ascending / descending manifolds containing the critical points. For example, consider the case of a two-dimensional Morse-Smale function after a maximum-saddle cancellation (see Figure 2). The 1-skeleton is updated by deleting the two corresponding nodes, deleting the arcs incident on the saddle, and re-routing the arcs incident on the maximum to the surviving maximum adjacent to the saddle. The embedding of a new arc is obtained by extending the old arc along the arc between the removed maximum and the saddle. We allow only cancellations between a pair of critical points that are connected by a single arc. Canceling a pair of critical points



3: A 2D discrete Morse function. Gradient pairs are shown as arrows oriented towards higher dimensional cells. Dashed red arrows denote edge-quad pairs and solid blue arrows denote vertex-edge pairs. Critical cells are shown in red (maxima), green (saddle), and blue (minima).

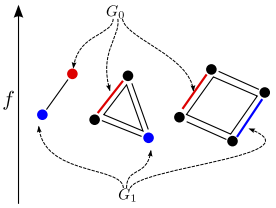
that are connected by two distinct arcs in the Morse-Smale complex results in a *strangulation*, which cannot be realized by a local smoothing of the function [GNP*06].

2.3. Discrete Morse functions

Discrete Morse theory was developed by Forman [For02] to study the topology of cell complexes. A d-cell α^d is a topological space homeomorphic to a d-ball $B^d = \{x \in \mathbb{R}^d : |x| \leq 1\}$. For example, a vertex is a 0-cell, an edge between two vertices is a 1-cell, a polygon is a 2-cell, and in general a d-dimensional polytope is a d-cell. We will restrict our attention to cells of the above kind, which can be represented by a set of vertices. A cell α is a *face* of β, denoted $\alpha < \beta$, if α is represented by a subset of vertices of β. The cell β is called a *coface* of α. A face α is called a *facet* of β if $\alpha < \beta$ and $dim(\alpha)+1 = dim(\beta)$. In this case β is a *cofacet* of α denoted by $\alpha \lessdot \beta$. The set of zero-dimensional faces of a cell α is called the vertex set of α denoted by V_α.

A *cell complex* K is a collection of cells that satisfies two properties: (a) If α belongs to K then so do all faces of α, and (b) If α_1 and α_2 are two cells in K then either they are disjoint or they intersect along a common face. A *regular cell complex* is a cell complex in which, given two incident cells, β^{d+1} and γ^{d-1}, there are exactly two cells α_1^d, α_2^d such that $\gamma^{d-1} < \alpha_1^d, \alpha_2^d < \beta$. In this paper, we consider only finite regular cell complexes.

Given a regular cell complex K representing the domain, a function $f : K \to \mathbb{R}$ is said to be a *discrete Morse function* if for all $d-$cells $\alpha^d \in K$, (a) at most one of its cofacets has a lower function value, and (b) at most one of its facets has a higher function value (see Figure 3). A cell is critical if none of its cofacets have a lower function value and none of its facets have a higher function value. A *discrete vector*, referred to as a *gradient pair*, is a pairing between two incident cells that differ in dimension by one. A *discrete vector field* on K is a set of discrete vectors such that every cell in K is represented in at most one pair of the field. A *V-path* is a sequence of cells $\alpha_0^d, \beta_0^{d+1}, \alpha_1^d, \beta_1^{d+1}, \ldots, \alpha_r^d, \beta_r^{d+1}, \alpha_{r+1}^d$ such that α_i^d and α_{i+1}^d are facets of β_i^{d+1} and $(\alpha_i^d, \beta_i^{d+1})$ is a vector for all $i = 0..r$. A V-path is called a *gradient path* if it contains no cycles. A *discrete gradient field* is a discrete vector field that contains no non-trivial closed V-paths. Maximal gradient paths of the discrete Morse function correspond to the notion of integral lines of Morse functions.



4: The weighted discrete function is defined recursively as a weighted sum of the function value at facet G_0 and face G_1. Facets G_0 (in red) and faces G_1 (in blue) for an edge, triangle and quad cell are shown. The function value at vertices increases along the vertical axis.

Ascending / descending manifolds are similarly defined for discrete Morse functions.

3. 2D MS complex Algorithm

In this section, we briefly describe the algorithm for 2D MS-complexes by [SMN11], without proof and detailed justifications. In the next section, we describe how the algorithm can be extended to three-dimensional scalar functions focusing on the additional challenges present in 3D. Given an input scalar function f, it is extended to a discrete Morse function F_w which imposes a total ordering on all cells of the domain K. The gradient field is interpreted as a directed acyclic graph and gradient paths as traversals on this graph. The gradient field decomposes into a union of trees and the graph traversal is performed in parallel as an iterative search procedure towards the root of the trees.

3.1. Weighted discrete function

The weighted discrete function F_w is equal to f at vertices.The value of F_w on a d-dimensional cell α^d is recursively determined by its facet with highest function value, while ensuring that the value of F_w at two different cells are never equal. Specifically,

$$F_w(\alpha^d) = F_w(G_0(\alpha^d)) + \varepsilon^d \times F_w(G_1(\alpha^d)),$$

where ε is an infinitesimally small positive real number,

$$G_0(\alpha^d) = \arg\max_{\gamma \lessdot \alpha^d} F_w(\gamma), \text{ and}$$

$$G_1(\alpha^d) = \arg\max_{\gamma < \alpha^d, V_\gamma \cap V_{G_0(\alpha^d)} = \phi} F_w(\gamma).$$

$G_0(\alpha)$ represents the highest facet of α^d and $G_1(\alpha)$ represents the highest face of α that is disjoint from $G_0(\alpha)$. $V_{G_0(\alpha)}$ is the vertex set of $G_0(\alpha^d)$, and arg max denotes the value of the argument γ that maximizes the function. F_w is equal to f at mesh vertices. F_w ensures that when two cells share their highest facet, then the tie is broken using their respective second maximum face whose vertex sets are disjoint from the common facet. Figure 4 shows the definition of the weighted discrete function for some common cell types. Subsequent

algorithms require only the order on the cells induced by F_w. This order may be computed in terms of f instead of explicitly computing F_w, assuming that the function values at the vertices are totally ordered. We use simulation of simplicity [EM90] to handle datasets that do not satisfy this requirement. In the following discussion we will use the analogy of height while referring to the value of F_w at a cell.

The ε-lower-star of a d-cell α is defined as the set of cells, including α, that have α as their highest d-face. As shown in [SMN11], the cells in the ε-lower-star of a cell appear contiguously in the ordering induced by F_w.

3.2. Computing gradient pairs

All cells are critical with respect to F_w. Algorithm 1 uses the ordering based on the weighted discrete function defined above to determine the gradient pairs of a simpler discrete Morse function that contains fewer critical cells. The gradient pairs are unique and independent of the order in which cells are processed. Algorithm 1 processes all cells $\alpha \in K$ individually to compute gradient pairs. The cell α belongs to the complex K, and β is a cofacet of α. The set P_α is the collection of cofacets, β, of α such that α is the highest facet of β, *i.e.*, $\alpha = G_0(\beta)$. The cell α is paired with the lowest cell in P_α.

Algorithm 1 ASSIGNGRADIENT (Cell complex K)

```
for all α ∈ K do
    P_α = {β | α ≺ β and α = G_0(β)}
    if P_α ≠ ϕ then
        β = Min_{F_w}(P_α)
        pair_cells (α,β)
```

3.3. Computing the 2D MS complex

Once the discrete gradient field is computed, the descending / ascending manifolds and hence the combinatorial MS complex are extracted as a closure of the set of gradient paths that originate at a critical cell / closure in the dual of the set of inverted gradient paths that originate at a critical cell. These are computed using a breadth first traversal of gradient paths. A combinatorial connection between any two critical cells is established if there is a gradient path that connects them. For multi-core environments, multiple BFS traversals are launched, one from each critical point. The gradient paths of a 2D domain decomposes to a union of trees [CCL03]. In massively parallel environments, each cell iteratively queries its parent for its parent, to locate the root.

4. 3D MS Complex computation

In this section, we discuss how the algorithm discussed in Section 3 is extended to work for three dimensional scalar functions. In particular, Algorithm 1 is conservative and may leave multiple cells unpaired thereby declaring them critical (See Figure 5). We introduce an additional pairing procedure that avoids the creation of such ε-persistent pairs within the ε-lower star. This procedure executes during a second pass over the cells and essentially seeks to pair cells with their second highest facet consistently when their highest facet is paired with another cell. Next, we describe a traversal algorithm to compute the incidence of 2-saddles on 1-saddles. This algorithm is adapted to determine the ascending/descending manifolds of 2-saddles and 1-saddles. Finally, the descending manifolds of maxima and ascending manifolds of minima are computed using an iterative search procedure marching towards the root of the tree.

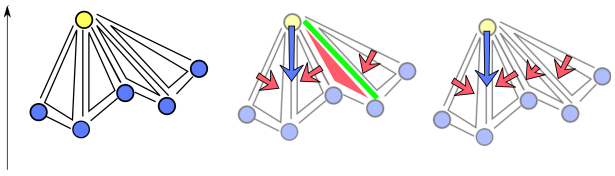


Figure 5: (left) The ε-lower-star and the lower link of a vertex with the height function defined. (center) Algorithm 1 declares the green edge and the red triangle as critical because the edge is not the highest facet of any of its cofacets. (right) Algorithm 2 pairs these cells because the edge is the second highest facet of the unpaired triangle.

4.1. Gradient pairing

Figure 5 illustrates a configuration where Algorithm 1 leaves a pair of cells unpaired because the green edge is not the highest facet for either of its two cofacets. However the edge is the second highest facet of its lowest cofacet. Further this cofacet remains unpaired, because it is not the lowest cofacet of its highest facet. Such situations occur frequently and in these cases we pair the cell with its second highest facet during a second pass over all d-cells ($d > 0$). We omit vertices since the change in the discrete Morse function required to realize the pairing is no longer arbitrarily small but is determined by the input scalar function. The pairing procedure is outlined in Algorithm 2.

Algorithm 2 ASSIGNGRADIENT2 (Cell complex K)

```
for all α ∈ K \ K^(0) do
    if α not paired by Algorithm 1 then
        P_α = {β | α is the second highest facet of β}
        if P_α ≠ ϕ then
            β = Min_{F_w}(P_α)
            if β not paired by Algorithm 1 then
                pair_cells (α,β)
```

Independence and Correctness. We first state and prove a Lemma to show that the pairs determined by Algorithm 2 are unique and hence the algorithm can be parallelized. The following Lemma states that if a cell is paired by Algorithm 2, then the pairing is unique, *i.e.*, the cell is either paired with one of its cofacets or with its second highest facet, independent of the order in which cells are paired.

Extended Order Independent Pairing Lemma. *If Algorithm 2 pairs a cell* β *with its second-highest facet* α *then it will not pair* α *with its second-highest facet* γ.

Proof. Consider the incidence relationships shown in Figure 6a between a d-cell β, its highest facet α', second highest facet α, highest $(d-2)$ face γ', highest $(d-3)$ face ψ', and α's second highest facet γ. Since the input cell complex K is regular, there exists exactly two facets of β, say α_1 and α_2, incident on γ'. Further γ' is the highest facet of α_1 and α_2, which in turn implies that any third facet of β does not contain γ' and is hence lower than both α_1 and α_2. So, α_1 and α_2 are the highest and second highest facets of β, namely α' and α. A similar argument on α and its highest $(d-3)$ face ψ' shows that γ and γ' are incident on ψ' and ψ' is their highest facet.

We will prove the existence of a facet, α'', of β that contains γ as its highest or second highest facet. In either case, γ will be paired with a cell different from α. If γ is the highest facet of α'' then Algorithm 1 would have paired it with a cell different from α because α remains unpaired until it is processed by Algorithm 2. If γ is the second highest facet of α'', then Algorithm 2 will seek to pair it with the lowest cofacet in P_γ. The cell α'' is lower than α and belongs to P_γ. So α will not be paired with γ.

We now show the existence of the cell α''. Consider the $(d-2)$ cell γ as a face of β. Since the input is a regular cell complex, there exists exactly two facets of β incident on γ. The cell α is one such facet of β. Let α'' be the other. The regularity of the input cell complex also implies the existence of exactly two facets, γ and γ'', that are incident on ψ'. Further, ψ' is the highest $(d-3)$ face of α''. It follows that γ is either the highest facet or second highest of α'' using the same argument as above to show that α and α' are incident on γ'. □

We now prove that Algorithm 2 produces a valid gradient field. A pairing between a cell α and its cofacet β is valid if there exists a corresponding discrete Morse function. Such a function can be realized via a perturbation if none of the cofacets of α are lower than β. Let $\tilde{\beta}$ be a cofacet of α different from β. The cell α is clearly not the highest facet of $\tilde{\beta}$ since it remains unpaired after being processed by Algorithm 1. If α is the second highest facet of $\tilde{\beta}$, then $\tilde{\beta}$ is higher than β because it was not selected by Algorithm 1. If α is neither the highest nor the second highest facet of $\tilde{\beta}$, then the highest facet of α, say γ', is not the highest $(d-2)$ face of $\tilde{\beta}$. This follows from the converse of the argument used in the above proof to show that α' and α, the highest and second highest facets of β, are incident on γ'. Cells in the ε-lower-star of γ' appear contiguously in the ordering induced by F_w. Since $\tilde{\beta}$ does not lie in the ε-lower-star of γ' and β does, it follows that $\tilde{\beta}$ is not lower than β.

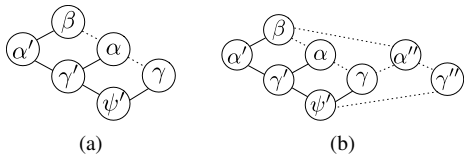


Figure 6: (a) α is the second highest facet of β and γ is the second highest facet of α. Respective maximal facets are shown (α', γ'). Solid lines represent maximal facet relation. Dotted lines represent incidence relation. (b) The regularity of K implies the existence of faces α'' and γ''.

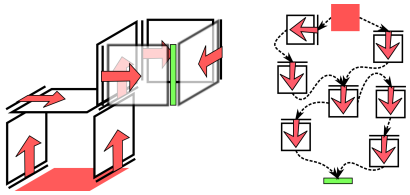


Figure 7: (left) The (1,2) sub-structure of a possible gradient field between a the red 2-saddle and the green 1-saddle. (right) The gradient field interpreted as a directed acyclic graph. The nodes are 2-saddles, (1,2) pairs and 1-saddles. Dashed curves show directed edges from 2-saddles or from the 2-cells of (1,2) gradient pairs to incident 1-cells of distinct (1,2) pairs or to 1-saddles. The gradient paths from the 2-saddle split and merge twice before they reach the 1-saddle resulting in four possible paths between them. Repetition of this configuration causes an exponential growth in the number of paths connecting the 2-saddle to the 1-saddle.

4.2. Saddle connection Algorithm

The sub-structure of the gradient field consisting of 1-saddles, 2-saddles and (1,2) gradient paths between them can be very intricate. This is because 1-saddle-2-saddle gradient paths in a three-dimensional domain may both split and merge. Figure 7 depicts the sub-structure of a gradient field that originates from a 2-saddle, splits and merges twice, before reaching a 1-saddle.

We trace the (1,2) gradient paths by interpreting the sub-structure as a directed acyclic graph (DAG) induced by them. The number of paths between a 2-saddle and 1-saddle may be counted as the number of paths between 2-saddles and 1-saddles nodes in this DAG. We do not employ the standard breadth first search algorithm to traverse the graph because

this would necessitate the use of an array of flags to maintain if every cell is visited or not. Parallelizing the traversal will require a buffer, whose size equals that of the input, for each thread. This approach is clearly not scalable. We note that the number of critical cells reachable from a given cell tends to be small. Algorithm 3 describes a priority-queue based traversal method to determine the paths between 2-saddles and 1-saddles. The algorithm computes the number of gradient paths from a given 2-saddle to all (1,2) gradient pairs and 1-saddles that are reachable from it.

The algorithm begins by first initializing a priority queue, PQ, that can contain 2-cells and 1-cells of K. The priority queue is ordered based on the simpler discrete Morse function computed by the gradient pairing algorithms. We associate with each element α of PQ the number of paths that arrive from σ. PQ is initialized with the pair $(\sigma,1)$. The algorithm pops the first cell α from PQ. It is possible for copies of the same cell to be entered into PQ. Since all these cells have the same priority, PQ is repeatedly popped until all copies of α are removed and the number of paths ($npaths$) that reach α are summed over all copies. If α is a critical 1-cell, then an arc with $npaths$ multiplicity is inserted between σ and α. If α is a 2-cell, then all the 1-saddles and 2-cells of (1,2) pairs incident on the boundary of α (other than itself) are inserted into PQ. The newly inserted pairs/saddles lie on $npaths$ number of paths from σ through α. Newly inserted cells are lower than α. So, α never re-enters PQ. A cell is inserted into PQ when processing one of its neighboring cells. So, the number of copies of the cell in PQ is upper bounded by the number of its neighbors.

Each cell enters PQ only a constant number of times. So, the complexity of the algorithm is $nlog(n)$, where n is the number of 2-cell-1-cell pairs and 1-saddles. Descending manifolds of saddles are computed by modifying Algorithm 3 to save the cells popped out of the priority queue at each iteration of the main loop. Ascending manifolds of 1-saddles are computed by employing the same procedure after reversing the priority, and reversing the role of 1-cells and 2-cells.

Algorithm 3 CONNECTSADDLES
INPUT: Cell complex K, 2-Saddle σ,
OUTPUT: Updated Morse-Smale Graph G

```
1:  PQ := CreatePriorityQueue()
2:  PQ.push(σ, 1)
3:  while PQ ≠ ϕ do
4:      α := PQ.top().cell
5:      npaths := PQ.top().npaths
6:      PQ.pop()
7:      while PQ ≠ ϕ and PQ.top().cell = α do
8:          npaths = npaths + PQ.top().npaths
9:          PQ.pop()
10:     if dim(α) = 1 then
11:         G.connect(σ, α, npaths)
12:     if dim(α) = 2 then
13:         for all γ < α do
14:             if is_critical(γ) then
15:                 PQ.push(γ, npaths)
16:             else if dim(pair(γ)) = 2 and pair(γ) ≠ α then
17:                 PQ.push(pair(γ), npaths)
```

5. Handling large data

The 3D MS complex algorithm is extended to handle large data that do not fit in memory using a split and merge technique similar to that described in [SMN11] for 2D scalar functions. The MS-complex is computed in a five-stage process:

- Split dataset
- Compute MS complex on sub-domains
- Merge sub-domains
- Traverse merge history
- Extract geometry

The splitting of the data in stage 1 and simplification in stages 2 and 3 are different from the method described in [SMN11]. We describe these steps below. The complete algorithm is described in the Appendix.

First, the dataset is split into sub-domains. The domain is subdivided successively along each axis till sub-domains of manageable size are obtained. Then, we compute the MS complex within each sub-domain using the algorithm discussed in Section 4. To eliminate noisy critical cells early, we perform critical cell pair cancellations within sub-domains directed by topological persistence [ELZ02]. To ensure combinatorial consistency of the MS complex after merging, we allow cancellation of a pair of critical cells only if (a) neither of the two cells belongs to a gradient pair that crosses a shared boundary, (b) neither of the two cells lie on a shared boundary, and (c) at most one of the two cells is connected to a gradient pair that crosses a shared boundary. Next, we merge the MS complexes of sub-domains in reverse order of the subdivision to create the combined MS complex, similar to [SMN11]. The validity and equivalence of the resulting MS complex is a consequence of the ORDER INDEPENDENT CANCELLATION lemma [SMN11]. After each merge we again perform cancellations directed by persistence to remove low persistent pairs that may have been created. Merge history traversal and extracting geometry are described in the Appendix.

6. Implementation

We implemented the above discussed techniques to leverage both GPU computing, and multi-core CPU architectures. We used the OpenCL framework to implement the gradient algorithm discussed in Section 4.1 on the GPU. We implemented the Algorithm 3 to process individual 2-saddles in

parallel on the CPU. We use the Boost threading [Boo] library to manage multiple threads. We now discuss various implementation and optimization issues with respect to the five stages.

Gradient pairing and MS complex. The gradient pairs and MS-complex are computed within each sub-domain. The gradient pairs are computed in two passes using Algorithm 1 and Algorithm 2. A three-dimensional array of bytes is used to store information per cell. Three bits are used to represent the six possible directions of the pair. Three bits are used to represent the six possible directions of the highest facet. One bit is used to mark cells as critical or not. The 3D array is created on the GPU and transferred to the CPU once the gradient computation is completed.

Next, the decomposition of the domain into descending/ascending manifolds of maxima / minima are computed on the GPU. This is done similar to the the algorithm described in Section 3. Simultaneously, the CPU executes Algorithm 3 to determine connections between 2-saddles and 1-saddles. This step is optimized to traverse only those paths that reach 1-saddles by executing a single breadth first search traversal that begins from all 1-saddles and marking all (1,2) pairs that are reached when traversing the gradient field upwards. While processing paths that descend from the 2-saddles, only pairs that are marked reachable from a 1-saddle are inserted into the priority queue PQ. One bit of the 3D array, which records per cell information, is used to represent visited or not visited state of a cell.

The Morse-Smale complex is represented as a graph with nodes as critical cells. Adjacencies are represented as list of associative arrays, one for each critical point and the multiplicity of paths associated with each adjacency. Hence the complexity to access a particular adjacency of critical cell is $log(n)$, n being the maximum number of adjacent critical cells.

Merging proceeds by cancellation of gradient pairs that cross a shared boundary. So, the descending connections of lower index critical cell of the pair and the ascending connections of the higher index critical cell are discarded by the cancellation. We further optimize by not recording such connections in the MS complex and not launching Algorithm 3 from 2-saddles that are paired with maxima.

Merging sub-domains. The merging procedure proceeds in the reverse order of subdivision of the domain. Two sub-domains are merged into a single MS complex while ensuring that gradient pairs that cross the shared boundary are identified. Then these pairs are canceled out. This procedure ensures that there is no duplication of critical cells (and their combinatorial connections) that lie on the shared boundary. Similar to the previous step, a persistence based simplification is performed on the combined MS complex after each merge.

Traversing merge history and extracting geometry. The traversal of merge histories computes paths from critical cells that enter a sub-domain through a shared boundary. Since we simultaneously perform simplification during merging, we consider critical cells within a sub-domain that are paired with other critical cells, possibly outside the sub domain, as entry points of gradient flow from surviving critical cells.

For extracting geometry, we require the original gradient field information. To obtain this, we reload the original function and recompute the gradient field, since the time taken to store and load this information is significantly higher than recomputing it. The partition of the domain based on gradient paths from extrema is computed on the GPU and the ascending/descending manifolds of saddles are computed using the modified version of Algorithm 3.

7. Experiments

We performed experiments on two different classes of datasets. First, we evaluated our algorithm with synthetic datasets to analyze its efficiency and scaling behavior with varying parameters such as regions of near flat-gradient and large numbers or gradients crossing shared boundaries. Second, we evaluated our algorithm's performance on various volume datasets available from *http://www.volvis.org* and a dataset obtained from the simulation of a 3D Taylor-Green vortex flow on a Cartesian grid. All experiments were performed on an Intel-Xeon 2 GHz CPU with 4 cores and 16 GB of RAM and an NVidia GeForce GTX 460 GPU with 336 cores and 1GB of memory. Data was split into $256 \times 256 \times 256$ sized sub-domains that fit in memory.

Synthetic data. We use a synthetic dataset WGAUSS to stress test the algorithm. The function is defined as the product of a cosine wave with the 3D Gaussian *i.e.* $f(x,y,z) = cos(2\pi\nu d_c) \times G_{c,\sigma}(x,y,z)$, where ν is the frequency of the cosine wave, d_c is the distance of the point from the center, $G_{c,\sigma}$ is the 3D Gaussian centered at c with variance σ, and the domain is the unit cube. The function is sampled at various grid resolutions to study scalability. This dataset is challenging since it contains multiple flat-regions at concentric spheres, where the cosine wave achieves its maximum or minimum.

We use two variants of this function to study our algorithm. First, we place the Gaussian at the center of the domain with variance 0.5 in all directions and set $\nu = 5$. The function contains concentric flat regions distributed across sub-domains resulting in several insignificant critical points. Experiments with this dataset helps study the scalability of the algorithm in the presence of noise and flat regions. Second, we distribute eight Gaussians, such that each one is centered in each octant of the domain. This variant does not possess as many flat regions because the multiple cosine waves

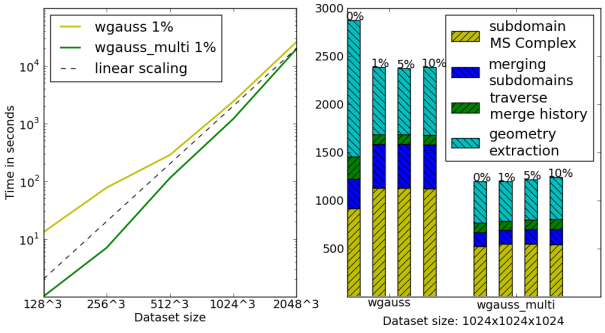


Figure 8: (left) Scaling behavior with varying sizes of the WGAUSS dataset simplified using a threshold of 1%. (right) Computation times for stages 2-5 for the 1024^3 datasets. The persistence threshold for each run is shown above the barplots. Time taken to split the data into sub-domains is approximately 3 minutes.

Dataset	size	#crits.	time	(a)	(b)
Silicium	98×34^2	1375	0.1s	3s	-
Fuel	64^3	773	0.2s	5s	-
Neghip	64^3	5663	0.3s	16s	7s
Hydrogen	128^3	26725	1.5s	69s	47s
Anuerism	256^3	95865	15s	118m	5m
WG	1024^3	13531699	42m	-	-
WG_M	1024^3	4599	20m	-	-
VORTEX_μ	1024^3	1266976	32m	-	-
WG	2048^3	54141119	464m	-	-
WG_M	2048^3	4575	370m	-	-

Table 1: Timings for datasets available from *volvis.org* compared with timings to compute the MS complex as reported in (a) [GRWH11] and (b) [GBPH08] .

superpose and break up the flat regions. This causes several gradients to cross common boundaries thus stressing the scalability of the merge and merge history traversal.

Figure 8 shows the computation time for large data sizes. As can be seen from the figures, our algorithm performs better on the WGAUSS_MULTI dataset which contains fewer flat regions. The time to cancel the gradient pairs that cross shared boundaries is lesser than the time taken to perform a persistence based simplification of the MS complex. This is reflected in the time taken to merge the sub-domains of the WGAUSS dataset. Figure 8 also plots running time for increasing data sizes of the WGAUSS dataset. The scaling results are similar for various values of persistence threshold. We note that the curves deviate away from linear scaling with increasing data sizes. On detailed analysis, we observed that stages two and five scale linearly whereas stages three and four did not. This is because of the representation of the combinatorial MS complex as a list of associative arrays.

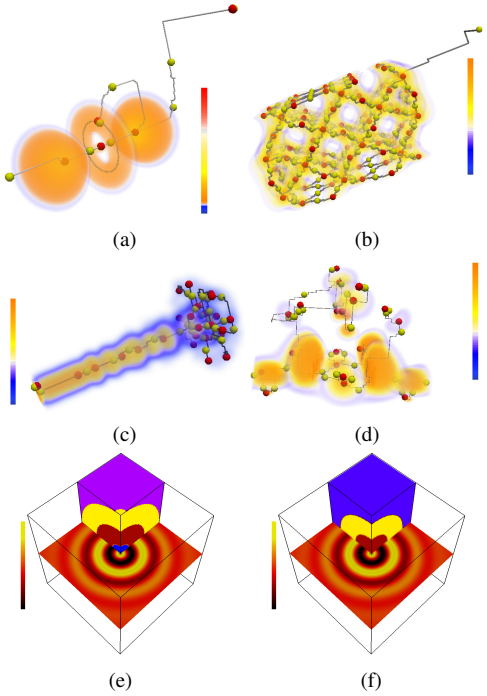


Figure 9: Ascending arcs between 2-saddles and maxima shown with the volume rendered image for (a) Hydrogen, (b) Silicium, (c) Fuel and (d) Neghip Datasets. Segmentation of the WGAUSS dataset into (e) descending manifolds (f) ascending manifolds.

Performance. To verify the benefits of parallelization, we compare our algorithm against existing methods [GBPH08, GRWH11] on datasets available from *volvis.org*, and a vortex flow data set, see Table 1. The experimental results indicate orders of magnitude improvement in the running time. Further, the memory required by our algorithm is less than 2GB even for the larger WGAUSS dataset. Gyulassy et al. and Günther et al. [GBPH08, GRWH11] report 23h and 5h, respectively, to process data sizes close to 1024^3. Figure 9 shows the critical 2- and 3-cells along with the ascending manifolds of 2-saddles for various datasets as well as slice visualizations of the WGAUSS dataset along with the decomposition of one sub-domain into descending / ascending manifolds of maxima / minima.

8. Conclusions

We have described a parallel algorithm to compute MS complexes of three-dimensional scalar functions. A hybrid multicore implementation results in significant speedup and superior performance as compared to existing approaches. Further, the algorithm is also memory efficient. In future, we plan to extend the algorithm to handle unstructured meshes.

Acknowledgement: This work was supported by a grant from Intel.

References

[BEHP04] BREMER P.-T., EDELSBRUNNER H., HAMANN B., PASCUCCI V.: A topological hierarchy for functions on triangulated surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 4 (2004), 385–396. 1

[BLW11] BAUER U., LANGE C., WARDETZKY M.: Optimal topological simplification of discrete functions on surfaces. *Discrete and Computational Geometry* (Apr. 2011), 1–31. 2

[Boo] http://www.boost.org/. 8

[CCL03] CAZALS F., CHAZAL F., LEWINER T.: Molecular shape analysis based upon the Morse-Smale complex and the Connolly function. In *Proc. 19th Ann. ACM Sympos. Comput. Geom.* (2003), pp. 351–360. 2, 5

[EHNP03] EDELSBRUNNER H., HARER J., NATARAJAN V., PASCUCCI. V.: Morse-Smale complexes for piecewise linear 3-manifolds. In *Proc. 19th Ann. Sympos. Comput. Geom.* (2003), pp. 361–370. 1

[EHZ03] EDELSBRUNNER H., HARER J., ZOMORODIAN. A.: Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete and Computational Geometry 30*, 1 (2003), 87–107. 1, 3

[ELZ02] EDELSBRUNNER H., LETSCHER D., ZOMORODIAN. A.: Topological persistence and simplification. *Discrete and Computational Geometry 28*, 4 (2002), 511–533. 3, 7

[EM90] EDELSBRUNNER H., MÜCKE E. P.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph. 9*, 1 (1990), 66–104. 5

[For02] FORMAN R.: A user's guide to discrete Morse theory. *Séminaire Lotharingien de Combinatoire 48* (2002). 2, 4

[GBPH08] GYULASSY A., BREMER P. T., PASCUCCI V., HAMANN B.: A practical approach to Morse-Smale complex computation: scalability and generality. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1619–1626. 1, 2, 9

[GDN*07] GYULASSY A., DUCHAINEAU M., NATARAJAN V., PASCUCCI V., BRINGA E., HIGGINBOTHAM A., HAMANN B.: Topologically clean distance fields. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1432–1439. 1

[GNP*06] GYULASSY A., NATARAJAN V., PASCUCCI V., BREMER P. T., HAMANN B.: A topological approach to simplification of three-dimensional scalar fields. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (2006), 474–484. 1, 4

[GRWH11] GÜNTHER D., REININGHAUS J., WAGNER H., HOTZ I.: Memory efficient computation of persistent homology for 3D image data using discrete Morse theory. In *Sibgrapi 2011 - Technical Papers* (2011). 1, 2, 9

[KRHH11] KASTEN J., REININGHAUS J., HOTZ I., HEGE H.-C.: Two-dimensional time-dependent vortex regions based on the acceleration magnitude. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2080–2087. 1

[LBM*06] LANEY D., BREMER P. T., MASCARENHAS A., MILLER P., PASCUCCI V.: Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1053–1060. 1

[LLT04] LEWINER T., LOPES H., TAVARES G.: Applications of Forman's discrete Morse theory to topology visualization and mesh compression. *IEEE Transactions on Visualization and Computer Graphics 10*, 5 (2004), 499–508. 2

[Mil63] MILNOR. J.: *Morse Theory*. Princeton Univ. Press, New Jersey, 1963. 2

[PRG*11] PETERKA T., ROSS R., GYULASSY A., PASCUCCI V., KENDALL W., SHEN H.-W., LEE T.-Y., CHAUDHURI A.: Scalable parallel building blocks for custom data analysis. In *Proc. IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)* (2011). 1, 2

[RH11] REININGHAUS J., HOTZ I.: Combinatorial 2d vector field topology extraction and simplification. In *Topological Methods in Data Analysis and Visualization*, Pascucci V., Tricoche X., Hagen H., Tierny J., (Eds.), Mathematics and Visualization. Springer Berlin Heidelberg, 2011, pp. 103–114. 2

[RKG*11] REININGHAUS J., KOTAVA N., GÜNTHER D., KASTEN J., HAGEN H., HOTZ I.: A scale space based persistence measure for critical points in 2d scalar fields. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2045–2052. 1

[RLH11] REININGHAUS J., LOWEN C., HOTZ I.: Fast combinatorial vector field topology. *Visualization and Computer Graphics, IEEE Transactions on 17*, 10 (2011), 1433 –1443. 2

[RWS11] ROBINS V., WOOD P. J., SHEPPARD A. P.: Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 99*, PrePrints (2011). 2

[Sma61a] SMALE S.: Generalized Poincaré's conjecture in dimensions greater than four. *Ann. of Math. 74* (1961), 391–406. 1

[Sma61b] SMALE S.: On gradient dynamical systems. *Ann. of Math. 74* (1961), 199–206. 1

[SMN11] SHIVASHANKAR N., MAADASWAMY S., NATARAJAN V.: Parallel computation of 2D Morse-Smale complexes. *IEEE Transactions on Visualization and Computer Graphics 99*, PrePrints (2011). 2, 4, 5, 7