CONSISTENT REPRESENTATION OF TWO-DIMENSIONAL FLOW

by

Shreeraj Jadhav

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computing

School of Computing

The University of Utah

August 2012

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Shreeraj Jadhav**

has been approved by the following supervisory committee members:

Valerio Pascucci	, Chair	**4-30-2012** Date Approved
Christopher Johnson	, Member	**4-30-2012** Date Approved
Elaine Cohen	, Member	**4-30-2012** Date Approved

and by **Alan Davis** , Chair of

the Department of **Computer Science**

and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

Analysis and visualization of flow is an important part of many scientific endeavors. Computation of streamlines is fundamental to many of these analysis and visualization tasks. A streamline is the path a massless particle traces under the instantenous velocities of a given vector field. Flow data are often stored as a sampled vector field over a mesh.

We propose a new representation of flow defined by such a vector field. Given a triangulation and a vector field defined over its vertices, we represent flow in the form of its transversal behavior over the edges of the triangulation. A streamline is represented as a set of discrete jumps over these edges. Any information about the actual path taken through the interior of the triangles is discarded. We eliminate the necessity to compute actual paths of streamlines through the interior of each triangle while maintaining the aggregate behavior of flow within each of them. We discretize each edge uniformly into a fixed number of bins and use this discretization to form a combinatorial representation of flow in the form of a directed graph whose nodes are the set of all bins and its edges represent the discrete jumps between these bins.

This representation is a combinatorial structure that provides robustness and consistency in expressing flow features like the critical points, streamlines, separatrices and closed streamlines which are otherwise hard to compute consistently.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I take this opportunity with great pleasure to express my gratitude towards all the people without whom this thesis would not have happened. I would like to thank my advisor Dr. Valerio Pascucci for all the guidance that he provided during my graduate studies. I am thankful for all the discussions, ideas and insights that he provided and for his patience all the way through.

I am also thankful to Dr. Joshua Levine and Dr. Peer-Timo Bremer for being great mentors. They have shown immense patience and have spent a lot of their valuable time for helping me with my difficulties. Their contribution cannot be overstated. I would also like to thank my other collaborators, i.e., Harsh Bhatia, Dr. Luis Gustavo Nonato and Dr. Guoning Chen for their important contribution to this work.

Lastly, I am grateful to my friends and family for their emotional support and for keeping me motivated. I dedicate this thesis to them.

CHAPTER 1

INTRODUCTION

Study of fluid flow is common in a variety of engineering and scientific endeavors like combustion simulations, aerodynamics of automobiles and airplanes, study of climate and oceanic currents, and high energy physics. Analysis and visualization of fluid flow is crucial for all such investigations.

There are many different techniques for analyzing and understanding fluid flow. For example, for visualizing behaviors like rotation and stretching, scalar quantities like vorticity [13, 23] and FTLE fields [12, 22] are computed. The variation in magnitude of these quantities indicates the distribution of such behavior across the domain. For analyzing global behavior and structural properties of flow, techniques rely on computing topological features [7, 17, 24]. This is done by computing streamlines that are paths of massless particles under instantaneous velocities of flow. If advected for infinite time in forward and backward direction, streamlines converge to features such as critical points and closed streamlines which are the limit sets of these streamlines. For obtaining a global structure, flow is subdivided into regions of uniform flow, such that streamlines within each of these regions have a common pair of limit sets. Streamlines that bound these regions are called separatrices.

The above mentioned techniques are based on mathematical concepts that assume a smooth vector field defined over a smooth domain. They also assume exactness in computations. However, while representing and analyzing flow in a computer, assumptions made under this model do not hold. First, flow is traditionally represented in a discrete manner using a sampled vector field over the vertices of a mesh. The vector field is extended to the interior of the simplices using interpolation. This is an approximation of the vector field as well as the underlying domain. Based on the interpolation chosen and whether the domain is nonplanar, this may violate the assumption that the vector field and its domain are smooth. Second, though flow features like streamlines, critical points, etc. are mathematically well defined, the algorithms to compute them are based on numerical methods that are approximate. This approximation can cause violations of mathematical invariants of these features. For example, by definition, streamlines are pairwise disjoint, i.e., they do not touch or intersect each other. Since, however, the numerical methods used to compute them are prone to errors, streamlines may cross over each other. Figure 1.1 shows an example of an invalid topological skeleton due to numerical integration errors. Similarly, due to numerical errors, critical points may go undetected or may get wrongly classified. Thus, computing these features consistently becomes a difficult task. Third, a computer can use only a finite set of bits to represent real numbers. Thus, it represents only a discrete set of values. This causes truncation error in every arithmetic operation, which may contribute to above mentioned inconsistencies.

Analysis techniques compute results disregarding above mentioned effects of representing and analyzing vector fields in a computer. Consequently, this prevents reliable interpretation of data since consistency of computed features cannot be guaranteed. Furthermore, downstream analysis techniques such as those developed for scalar field analysis [11, 16] cannot be applied without a valid underlying structure.

To address these issues, a combinatorial representation of flow [21] was recently developed based on the theory of combinatorial vector fields [6]. This theory defines a combinatorial equivalent of smooth vector fields. It constructs a vector field as a simplicial graph of the underlying mesh. Under this graph, computations can be performed consistently and robustly. However, conversion of a vector field into a simplicial graph causes some coarsification. Finer mesh may be required for complex flow behaviors in such a representation.

We describe a new representation of flow that avoids above mentioned issues in its analysis. This representation combines the characteristics of combinatorial and traditional methods of representing flow. This representation uses a combinatorial structure for robustness and consistency and it can also represent flow at approximation levels equivalent to that of traditional representation. We establish equivalent definitions of flow features from smooth vector fields such as critical points, streamlines, separatrices, closed streamlines, etc. thus forming an analogous representation of flow.

Using a triangulation of the domain, we represent flow in the form of its transversal behavior over the edges of this triangulation. We only maintain the relation between the

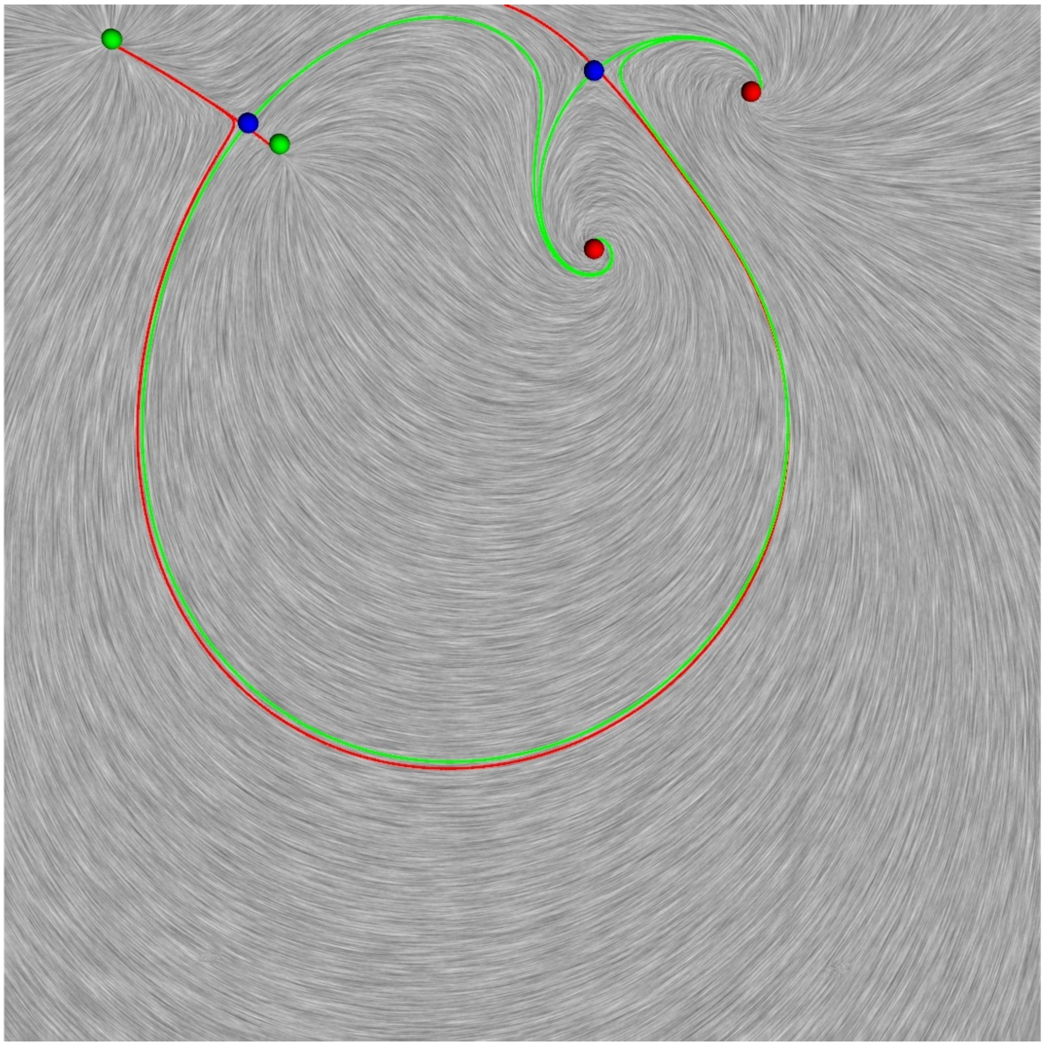


Figure 1.1. Inconsistent topology generated by computing separatrices using numerical integration. Compounding integration error causes two separatrices to intersect, producing an invalid topological skeleton.

entry and exit points in form of a map through each triangle. Flow is represented as discrete jumps through the interior of triangles. This notion of capturing boundary behavior of flow follows from a recently developed representation called edge maps [1, 15]. We handle the discretization of real numbers explicitly using quantization of each edge into a uniform set of bins represented by integers. This provides a combinatorial structure to the representation which we exploit to form graph based algorithms for analysis of flow.

Given a smooth vector field, we can approximate flow to an abitrary accuracy level while converting to our representation. Though approximation errors are incurred during this conversion, any further computations on the flow can be performed without incurring additional errors. In other words, by converting flow to our representation, we isolate accuracy from consistency. We provide consistent algorithms for computing features like critical points, streamlines, separatrices and closed streamlines to demonstrate this capability of the representation.

The goal of this thesis is to provide a solution to inconsistencies in conventional methods of data analysis for fluid flow data. This thesis argues that use of discrete structures to approximate flow datasets can be effective for consistent analysis which cannot be guaranteed by conventional methods that depend on numerical methods and floating point computations. We describe a new combinatorial representation for two dimensional steady state vector fields. This representation is analogous to smooth vector fields, that can represent equivalent structural features like critical points, streamlines, separatrices, closed streamlines, etc. We provide equivalent definitions of these features in our representation and provide a method to compute them in a robust and consistent manner.

CHAPTER 2

RELATED WORK

Many analysis and visualization techniques have been developed that use traditional numerical tools like interpolation and integration. For example, computation of topological skeleton of vector fields. The idea of using topology for vector field visualization was introduced by Helman and Hesselink [14]. Helman and Hesselink define a two-dimensional vector field's *topological skeleton* as a graph constructed using a special set of streamlines, called *separatrices*, that connect the critical points of the field. Separatrices are the four streamlines that (asymptotically) travel to and from each saddle point. The majority of approaches in both two and three dimensions for computation of topological skeleton are based on numerical streamline computations for separatrices [5, 9, 19, 28, 25]. Similar techniques have been extended to multiresolution representations as well as time-dependent flows [8, 27]. However, it is well known that computing the topological skeleton can be numerically unstable due to errors inherent in the integration of separatrices and inconsistencies among neighboring triangles [4].

Subsequent techniques in two [18, 26, 29] and three [30] dimensions augment even more sensitive features like closed streamlines to vector field visualization. Closed streamline detection approaches are perhaps even more sensitive to numeric integration, as the periodic nature of the flow quickly exposes any inconsistencies in the integration scheme. Consequently, many of these approaches prefer progressive techniques, where the orbits are first detected at a combinatorial level (for example, in [29] a sequence of triangles containing a closed streamline is first identified) and then relying on numeric computations to compute the exact orbit. To address consistency and robustness issues in the analysis of vector fields, an approach by Chen et. al. [3] computes a directed graph using triangles of the underlying mesh as nodes called the Morse Connection Graph (MCG). The construction of this graph is based on discretization of paths taken by streamlines through these triangles. This graph is simplified by colapsing strongly connected components and removing redundant paths to converge to *Morse Sets*. These sets together with the graph form a conservative decomposition of flow and is guaranteed to be consistent. More recently, a combinatorial representation of flow [21] was recently developed based on the theory of combinatorial vector fields [6]. This theory defines a combinatorial equivalent of smooth vector fields. It constructs a vector field as a simplicial graph of the underlying mesh. Under this graph, computations can be performed consistently and robustly since they avoid numerical computations entirely. While similar approach have been successful for scalar field topology [10], due to the coarse nature of discrete vector fields, expressing complex flow behaviors like the one in piecewise linear vector fields may be difficult without drastically increasing the mesh resolution.

CHAPTER 3

TWO-DIMENSIONAL STEADY STATE VECTOR FIELDS

In this section, we introduce concepts from smooth vector fields which are crucial for data analysis and visualization and are important for establishing an equivalent combinatorial representation (quantized flow). We will revise the definitions of topological features like critical points, separatrices, limit sets and stable/unstable manifolds which form the topological skeleton of a two-dimensional vector field. Using these features of a vector field it is possible to form a topological segmentation of the domain called as Morse decomposition. Later, we will use such visualizations for demonstrating the capabilities and advantages of the quantized flow over traditional methods of computation.

A two-dimensional steady state vector field is defined as a map $\vec{V} : \mathcal{M} \to \mathbb{R}^2$ where $\mathcal{M}$ is a smooth 2-manifold. $\vec{V}$ defines a velocity (a vector) at every point on $\mathcal{M}$. Each point can be classified as either a critical point or a regular point. A point $c \in \mathcal{M}$ is a critical point if $\vec{V}(c)$ is a zero vector. All other points that do not satisfy this condition are regular points. A critical point is classified based on the behavior of vector field in its neighborhood. If the Jocobian matrix i.e. derivative of the vector field at a critical point is full rank, then the critical point can be classified as a first order critical point based on the eigen values of the matrix. First order critical points are broadly classified as sources, sinks, saddles and centers as shown in Figure 3.1.

More complex classes of critical points can exist if one or more eigen values are zero. These are referred to as degenerate or higher order critical points. A more general classification of critical points is given by the Poincaré-Hopf Formula. This formula maps the classes of critical points to an index number. This number is known as Poincaré index or winding number. By definition of a manifold, neighborhood of a critical point is locally Euclidean, such that the vector field in that neighborhood can be represented as

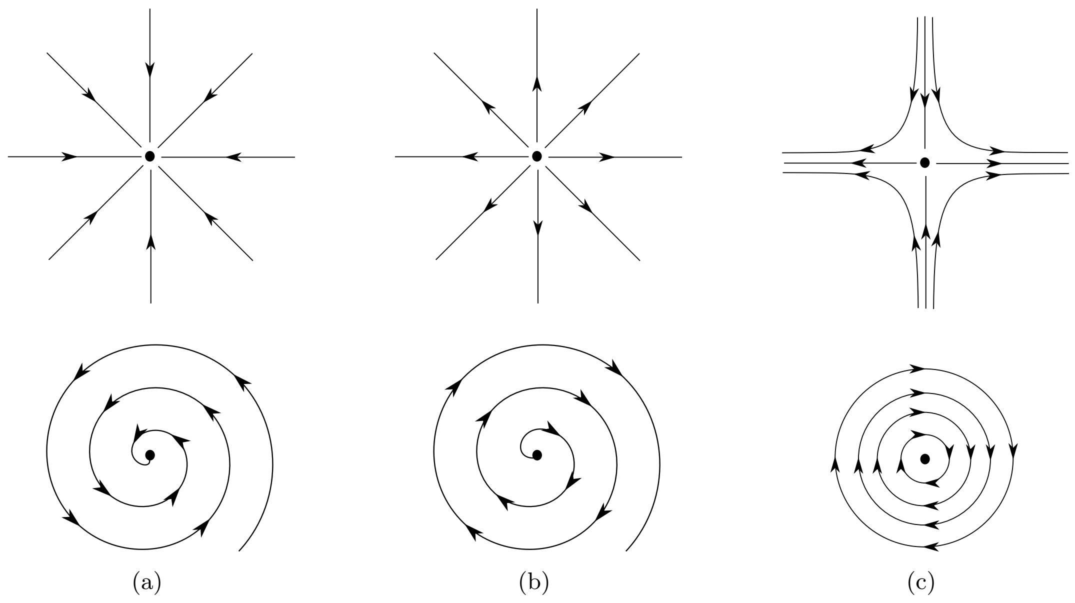


Figure 3.1. Types of first order critical points. (a) Sink and attracting focus; (b) source and repelling focus; (c) saddle and center

$$(x_1, x_2) \mapsto (v_1, v_2)$$

The Poincaré-Hopf formula is then given by:

$$i_c = \frac{1}{2\pi} \oint_{\Gamma} d\theta$$

where $\theta = \arctan \frac{v_2}{v_1}$ and Γ is a Jordan curve enclosing the critical point in that neighborhood. It basically computes the number of rotations that the vector makes while sliding along the curve around that neighborhood.

Starting at any point $x \in \mathcal{M}$, a path can be traced following the instantaneous velocities of $\vec{V}$. In a steady state vector field, this curve is called a streamline. A streamline can be defined as a solution to the differential equation,

$$\frac{d\phi(x,t)}{dt} = \vec{V}(x)$$

with the initial condition $\phi(x, 0) = x_0$. A vector field defines a continuum of streamlines. Asymptotic behavior of these streamlines reveal topological features or invariants of the vector field. Conventionally, numerical methods are used to compute these streamlines i.e. solve the differential equation mentioned above. An important invariant of vector field apart from critical points are cycles. A cycle is a streamline such that there exists $t_1 > 0$ where $\phi(x, t_1) = x$. If one traces a streamline from a point in a cycle, the streamline will flow back to that starting point making a closed streamline. Invariants like critical points and cycles that define the topological structure of a vector field are limit sets of the streamlines in that vector field. There are two types of limit sets for each streamline in a vector field without a boundary. An α-limit set (S_α) and ω-limit set (S_ω) of a streamline $\phi(x_0, t)$ is defined as follows:

$$S_\alpha = \{y \in \mathcal{M} | \exists (t_n)_{n \in \mathbb{N}} \subset \mathbb{R}, t_n \to -\infty, \lim_{n \to \infty} \phi(x_0, t_n) \to y\}$$
$$S_\omega = \{y \in \mathcal{M} | \exists (t_n)_{n \in \mathbb{N}} \subset \mathbb{R}, t_n \to \infty, \lim_{n \to \infty} \phi(x_0, t_n) \to y\}.$$

Intuitively, α-limit set of a streamline is obtained by advecting the streamline for negative infinite time and indicates the origin of that streamline. Similarly, ω-limit set is obtained by advecting a streamline for positive infinite time and it indicates the destination or end of that streamline. In steady state two-dimensional vector fields, the α- and ω-limit sets of all streamlines collectively form the set of all critical points and closed streamlines and describe the topological skeleton.

Each limit set defines a stable and an unstable manifold. Consider a limit set Y. Stable and unstable manifolds W_s and W_u of Y are defined as follows:

$$W_s = \{p \in \mathcal{M} | \lim_{t \to \infty} \phi(p, t) = y, y \in Y\}$$

$$W_u = \{p \in \mathcal{M} | \lim_{t \to -\infty} \phi(p, t) = y, y \in Y\}$$

The intersections of the (un)stable manifolds of all limit sets describes a topological segmentation of $\mathcal{M}$ called the Morse decomposition. Generating such a segmentation of the domain provides an insight into the global behaviour of a vector field. Generating a topological segmentation or detecting flow features like the ones mentioned above requires streamlines as a fundamental computation. It is very important to compute streamlines without any inconsistencies for the downstream analyses to be valid. Discrete models of flow like graph based approaches show promise since the computation of streamlines in such a representation is analogous to a graph traversal which ensures consistency by eliminating any kind of numerical computation.

CHAPTER 4

DISCRETE REPRESENTATION

In this chapter we define a discrete representation of vector field which helps guarantee consistent analysis of flow. To avoid inconsistencies and robustness issues due to numerical errors, we develop a completely combinatorial structure that describes flow over a triangulation based on its transversal behavior over the edges. Theoretically, conventional representation of vector field using interpolation represents a continuum of streamlines. Due to infinite number of streamlines, it is difficult to maintain consistency of computed streamlines specially due to errors incurred by numerical integration methods and floating point truncations. Also, it is difficult to compute unstable features like closed streamlines or periodic cycles.

Our representation consists of a finite number of streamlines that approximate the actual flow. We refer to these as quantized streamlines. The granularity of this representation is based on the number of quantized streamlines used. Quantized streamlines are formed by discretizing each edge of the triangulation in a finite number of bins such that all streamlines entering a bin merge into a single quantized streamline and are not allowed to diverge. In this manner, we ensure that streamlines can merge but cannot cross over each other.

We first discuss how we discretize the domain, i.e., the triangulation over which the flow is defined and how we define a graph called *bin-graph*, which discretely represents flow. We also discuss how this conversion impacts accuracy of represented flow as compared to its interpolated counterpart. We describe a method to represent flow at varying degrees of accuracy based on user defined error threshold.

4.1 Local Representation

In each triangle, we build a map that connects inflowing and outflowing bins, thereby maintaining all the quantized streamlines flowing through it. In this section, we describe the local construction of this combinatorial representation within each triangle.

4.1.1 Discretizing Edges

Flow through each triangle is represented as a map between its edges. An entry and exit point on the boundary of a triangle will be mapped if there exists a streamline between them that is completely contained in the interior of that triangle. We call such a pair of points an origin-destination pair. If we identify the set of all such origin points and destination points, then we can define a map from the set of origin points to the set of destination points which represents all streamlines flowing through the triangle.

We discretize each edge of the triangulation into a set of 2^k equal sized bins using a $k-$bit integer, such that each bin is either an inflow or an outflow bin. Consider streamlines of the smooth vector field flowing through these bins as they cross over the edges of the triangles. Streamlines entering a triangle flow through inflow bins and exit through outflow bins. In this discrete model, all streamlines entering an inflow bin are considered merged and are not allowed to diverge again. Thus, each inflow bin maps either to an outflow bin; or maps to the interior of a triangle in case of a critical point like a sink or a source. Let O be the set of all inflow bins and D be the union of all outflow bins and the interior of triangle. Then we can define a map $\xi : O \to D$ that we call a *Quantized Edge Map*. Figure 4.1 shows an example of a quantized edge map. Quantized edge maps are similar to the *Edge Maps* developed by Bhatia, Jadhav et al. in [1, 15]. Edge maps are continuous maps derived to represent piece-wise linear flow. Edge maps too represent flow in form of boundary map between edges of triangulation. However, since the maps are continuous, there are infinite number of streamlines passing through any triangle. Therefore, it is difficult to construct edge maps without knowning the exact nature of flow within a triangle. Computation of features like closed streamlines is also difficult. On the other hand, quantized edge maps represent flow in form of a graph based on discretized edges of triangulation. In concept, due to finite granularity, it is possible to construct a quantized edge map within a triangle without knowing the exact nature of flow. We provide a simple and efficient algorithm to contruct quantized edge maps in the piecewise linear case. Quantized edge maps in each triangle collectively provide a global combinatorial structure describing flow over a given triangulation of domain. Varying the size of bins can provide variable granularity or precision in representing flow. This representation can be considered as a large graph constructed using the bins (bin-Graph) on each edge as nodes and a map for each triangle

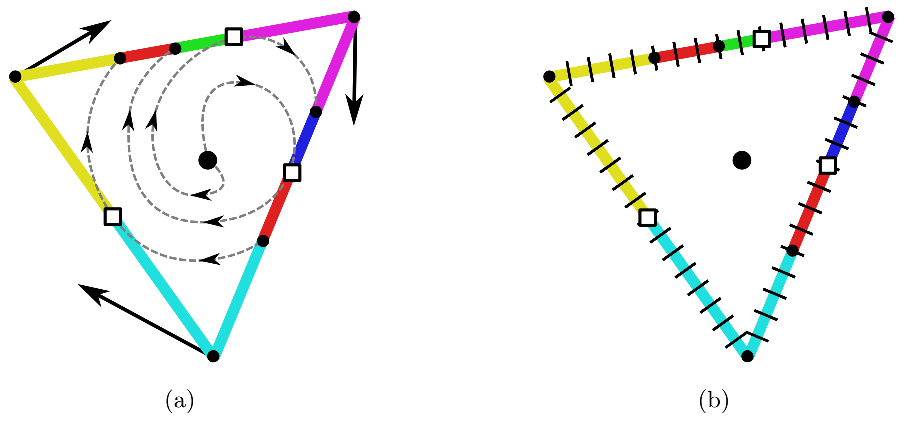


Figure 4.1. Construction of quantized edge maps. (a) Formation of links within a triangle. Each color represents a pair of origin and destination interval creating a link. (b) Mapping of inflow and outflow bins under such flow.

defining connections between the nodes. In the text to follow, we refer to edges of this graph as connections to disambiguate it from the edges of triangulation.

4.1.2 Restrictions

Though map within each triangle can be constructed independent from each other, there are certain requirements that they must satisfy to agree with the maps of their neighboring triangles as well as represent flow consistently in the interior. There are certain constraints that we apply on the bins and their mapping. These constraints and the properties thus implied are enumerated below. Properties 1, 2, and 3 are enforced, while properties 4, 5 and 6 are derived properties.

1. For any triangle T, each bin on ∂T is either an infow or an outflow bin.
2. On an edge, if a bin is an inflow bin for one triangle, then it has to be an outflow bin for the other triangle.
3. We do not represent separatrices of a saddle as bins. Thus these are implicitly represented as boundary between two bins.
4. From property 1, a switch from inflow to outflow on ∂T is represented as boundary between an inflow and an outflow bin.
5. From properties 1 and 2, there cannot exist a streamline flowing along the edge.

6. From properties 1 and 2, there cannot exist a sink or a source on an edge.

Note that sinks and sources have an explicit existence in quantized flow in the form of bins mapped to the interoir of T, while saddles and centers exist implicitly and should be detected by examining the neighborhood behavior. Above mentioned constraints help enforce consistency of flow.

4.1.3 Feasibility and Accuracy

To efficiently represent flow in each triangle, we approximate the map between inflow and outflow bins as a linear map between connected subsets of the boundary of a triangle. We call these linear maps as *links*. A collection of such links that form a covering of the entire boundary of a triangle defines a quantized edge map. Links can be identified as continuous set of streamlines entering and exiting a triangle. Since peicewise linear vector fields are commonly used, the notion of links and edge maps was used to study all possible structures in a piecewise linear vector field in [15]. It is a detailed study of local flow behaviors of such vector fields in the form of links and enumerate different structures that they can exhibit. Since flow can switch direction only once per edge, there are a limited number of possibilities. This study helps to establish a bound over the complexity of data structure required in this representation of flow. To approximate the flow within each link, the map between inflow and outflow bins can be linearly approximated using rasterization. Due to the explicit quantization, each link defines a map from m inflow bins to n outflow bins. We use Bresenham's algorithm [2] to compute the next point along an integration path, or more generally, given an inflow bin $i < m$, we compute its corresponding outflow bin $\xi(i)$ by rounding equation $n * i/m$ to the nearest integer. Clearly, there can be a one-to-many or a many-to-one mapping of bins. While, we allow streamlines to merge into a bin, we do not allow bifurcation. This condition is necessary for maintaining consistency of streamlines .If $m < n$, then by the rasterization algorithm an i^{th} inflow bin may map to more than one outflow bins. In such a case we always choose the rightmost bin to make a consistent choice. This is done for both forward and backward streamline computation. This ensures that streamlines do not cross over each other once they merge into a bin. Based on whether we represent forward flow or backward flow, the links remain the same but the direction of flow is inverted. We designate forward rasterization map as ξ^+ and backward rasterization map as ξ^-.

The linear mapping of bins within each link is an approximation of actual flow. The actual flow can be more complex than assumed. This is a source of error in our representation. We estimate this error by using a regular sampling within each link for tracing streamlines of the actual flow. We find the maximum offset between the destination of the actual streamlines and their approximated counterparts. Figure 4.2 illustrates this estimation process. Consider an inflow bin b and a true streamline $\phi(x_0)$ sampled at any point $x_0 \in b$. Let x_0' be the point where $\phi(x_0)$ first exits the triangle. The approximation error ϵ_a can be formulated as follows:

$$\epsilon_a = |x_0' - \xi(b)|$$

In the above expression, $\xi(b)$ is considered to be the midpoint of the destination bin. If this error is higher than a user provided threshold, we can split the link into smaller links using the regularly sampled streamlines of actual flow. The mapping of bins within the old link is now represented by a polygonal curve rather than a single line which increases the accuracy with which we approximate a given vector field. This process can be repeated recursively to achieve arbitrary levels of accuracy.

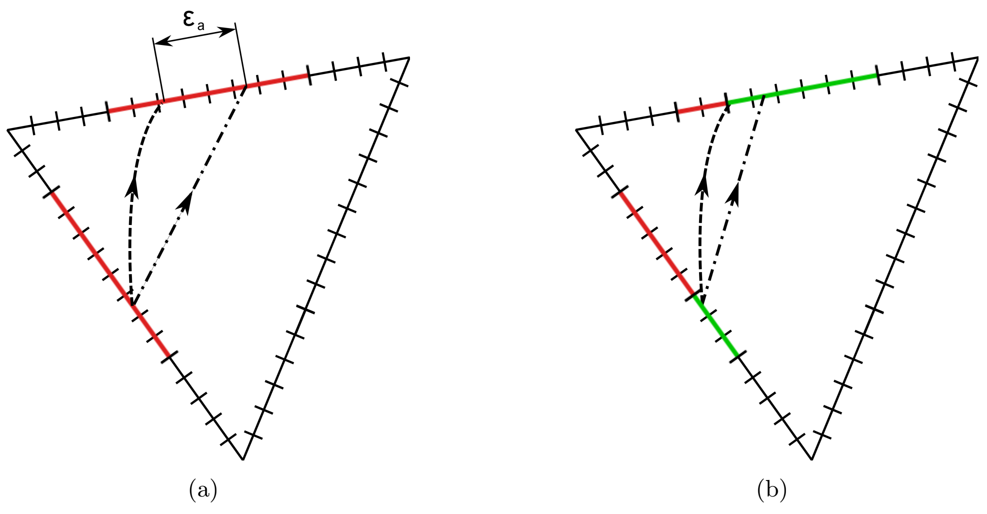


Figure 4.2. Refinement of quantized edge maps. (a) Estimating error and (b) refinement procedure. The dashed curve indicates an actual streamline computed from original flow. The dash-dotted line indicates linear approximation of the streamline within its link. Red lines in (a) indicate the origin and destination intervals of the initial link, while the red and green lines in (b) indicate two different links created by splitting the initial link.

4.2 Topological Features in Descrete Form

Two triangles sharing an edge also share the bins on that edge. The maps of such triangles represent inflow and outflow consistently across their shared edges. These maps connect up across the triangle edges to form a large graph which we call the *bin-graph.* Based on whether we are considering forward flow or backward flow, two bin-graphs can be defined. Both, forward and backward bin-graphs have the same nodes (bins) but different connections (mapping) between them.

Definition 1 (Bin-Graph) *Let B be the set of bins on a triangulation. The* forward bin-graph *is a directed graph ${G_B}^+(B, E^+)$ with $E^+ = \{(b, \xi^+(b)) \mid b \in B\}$. The* backward bin-graph*, G_B^-, is defined symmetrically.*

We represent flow as a finite number of streamlines in form of traversals through this graph starting from any bin. Each bin uniquely represents one forward and one backward streamline. Together, forward and backward bin-graphs describe the complete flow of a given vector field in a discrete manner. We will now define quantized streamlines and structural features in the context of this discrete structure.

4.2.1 Quantized Streamlines

In smooth vector fields, streamlines are computed using numerical integration methods. In our representation, we compute streamlines as a graph traversal through the bin-graph. Given the above definition of a bin-graph, it is important to understand the properties of flow represented by it. To simplify the discussion below, we define a quantized streamline or a *q-streamline* represented by a bin-graph as:

Definition 2 (Q-streamline) *A* forward q-streamline*, $S^+(b_0)$ starting at bin b_0 is an ordered sequence of bins $\{b_0, b_1, b_1, \ldots, b_n\}$ such that $\xi^+(b_i) = b_{i+1}$. A* backward q-streamline *$S^-(b_0)$ is defined symmetrically.*

Q-streamlines are illustrated in Figure 4.3 and the corresponding forward and backward Bin-Graph is shown in Figure 4.4. This image shows both a forward and a backward q-streamline integrated through five triangles. Based on the above definition and the properties of Bin-Graph, q-streamlines have some interesting properties that we state explicitly here:

- Used as a starting point, each bin b defines, at most, two q-streamlines $S^+(b)$ and $S^-(b)$. However, each bin can be part of multiple q-streamlines and different bins may define the same q-streamline.
- If two forward (or backward) q-streamlines merge, they do not bifurcate again (this includes self merging). For example, in the red link of the second triangle in Figure 4.3, three forward q-streamlines merge when entering the cyan link of the third triangle. Traced forward these lines will never split.
- Q-streamlines do not cross. Clearly, the rasterization procedure preserves the order of q-streamlines except when two lines merge. However, two merged lines never split guaranteeing that no two q-streamlines can cross.
- However, if a forward and backward q-streamline merge, they may bifurcate in their respective directions (as shown by the two q-streamlines in Figure 4.3).

We can exploit the combinatorial structure of this representation to identify important structural features of the flow like critical points, cycles, separatrices, and stable/unstable manifolds. Since the underlying structure is discrete we can avoid numerical methods for computing these features, thereby generating consistent and robust results.

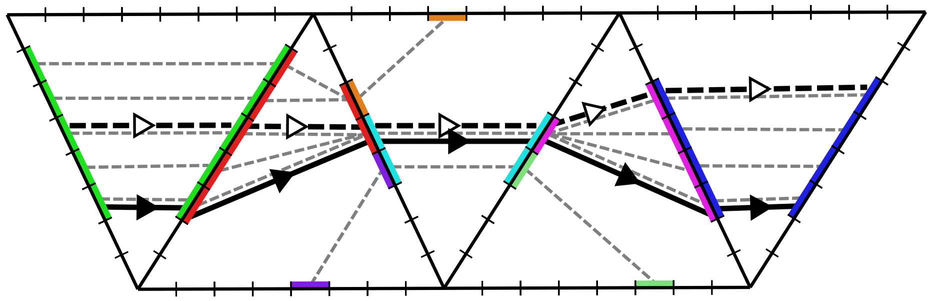
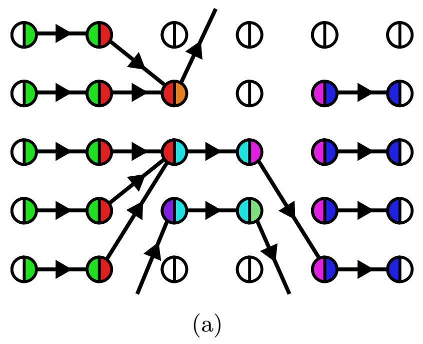


Figure 4.3. Subgraphs shown with colored bins indicating links and grey dashed lines for each bin-to-bin rasterization. Two q-streamlines are shown, a forward q-streamline with a solid black line and black triangles and a backward q-streamline is shown with a dashed black line and white triangles. Triangle direction indicates the direction of vector field, not the direction of integration. In both cases, the q-streamlines choose the rightmost (with respect to the integration direction) bin.



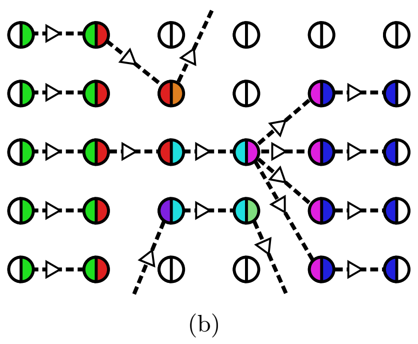


Figure 4.4. Forward (a) and backward (b) bin-graphs for the links illustrated in Figure 4.3. For each edge in this sequence, the bins are displayed as a column of circles, colored to indicate the links they fall in on either side of the edge.

4.2.2 Quantized Critical Points

In smooth vector fields, critical point in a neighborhood can be classified based on the Poincaré index or the winding number as mentioned in Chapter 3. This index is calculated as the number of rotations the vector at a point makes as it slides along a Jordan curve which does not have a zero vector on it. The point is moved in counter clockwise direction. Counter clockwise rotation of vector is considered positive while clockwise rotation of vector is considered negative rotation. The number of rotations is always an integer because the sliding point comes back to its starting position.

Our representation inherently stores behavior of flow on boundary of each triangle, we use this to detect and classify critical points based on Poincaré Index or the winding number. The index can be computed using a simple count of transition points. Transition points on a quantized edge map are the switch points along the boundary of a triangle where flow switches from the inflow to outflow. If the bins adjacent to a transition point map to each other, then it is called an *external transition point* (ETP), else it is called an *internal transition point* (ITP). This classification indicates how the vector defining such a flow would rotate. Since all these points are accounted for, the number of rotations can be computed independent of how the vectors behave between transition points. By developing an equivalent formula for computing Poincaré index ($ind_{\partial T}$) for the discrete representation, we detect and classify critical points consistently.

$$ind_{\partial T} = \frac{1}{2\pi}\left(\sum_{\|T_I\|}\pi + \sum_{\|T_E\|} -\pi + 2\pi\right) = (\|T_I\| - \|T_E\|)/2 + 1$$

T_I is the set of ITPs and T_E is the set of ETPs. This formula ignores any existence of a critical point on the edges or vertices of a triangle. Any such point will be treated as a transition point based on the quantized map thus computing Poincaré index of the interior of triangles. For detecting and classifying critical points that might lie on an edge, we compute Poincaré index of the quadrilateral formed by the two triangles sharing the edge and subtract the indices of the individual triangles. Similarly for a vertex, we compute the index for the polygon surrounding that vertex and subtract the indices of the surrounding triangles and edges. This can be done because classification of transition points existing at the vertices of quadrilaterals and polygons can be performed based on the local map of bins.

4.2.3 Quantized Cycles

Like critical points, there is another topological feature of vector fields that is important is a cycle. A cycle is also called as closed streamline or periodic orbit. Such a streamline acts as a limit set in two-dimensional vector fields, i.e., it can act as a sink, source or a saddle. If the trajectory of a point on a closed streamline is traced, one returns to the same point completing a loop. Exact closed streamlines are particularly difficult to compute using numerical techniques due to the inherent computation errors and inconsistencies caused due to the same. Since in quantized flow we represent flow discretely as a graph, a closed streamlines exist explicitly as closed loops of connections (graph-edges) in a bin-graph.

Definition 3 (Quantized Cycle) *A* closed q-streamline $S(b_0)$ *is a q-streamline such that there exists an* $n > 0$ *such that* $b_0 = b_n$*.*

4.2.3.1 Classification of Quantized Cycles

Since forward and backward maps differ, there exist three primary classes of cycles: those stable in ξ^+; those stable in ξ^-; and those stable in both. Also, closed streamlines are limit sets and can be classified based on whether they are attracting, repelling or neither on the sides of their trajectories. Based on these possibilities and eliminating symmetrical cases, we classify cycles in a total of 12 classes. Figure 4.5 shows examples of forward stable cycles. ξ^+.

Consider a forward stable cycle C. A forward stable cycle is defined by As discussed in section 4.1.3, in case of one to many mapping of bins, we always choose the rightmost bin

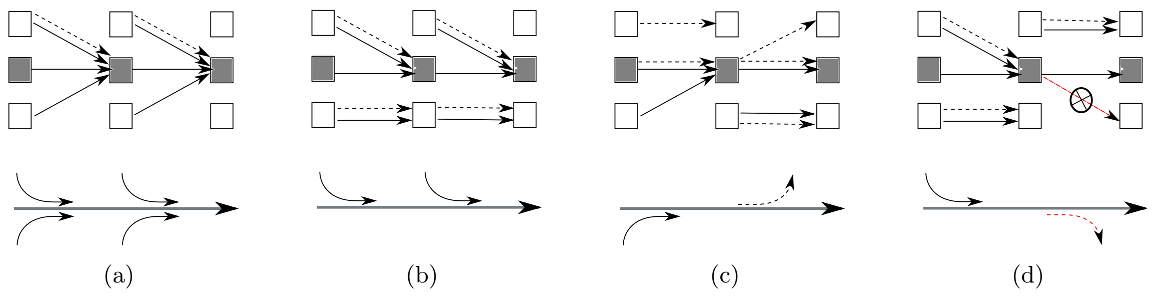


Figure 4.5. Classification of cycles. Images show different examples of forward stable cycles with corresponding bin graph. Grey bins refer to the bins in the cycle. Solid arrows indicate forward q-streamlines; dashed arrows indicate backward q-streamlines. (a) Forward stable cycle attracting on both sides. (b) Forward stable cycle attracting on one side and neutral on the other side. (c) Forward stable cycle attracting on right side and repelling on left. (d) A forward stable cycle cannot repel on its right-hand side.

in both ξ^+ and ξ^-. Thus, cycle C cannot have a diverging streamline on the right hand side. i.e. It cannot be repelling on right hand side w.r.t direction of forward flow. It can either be attracting if there exists at least one forward streamline approaching the cycle from the right side or neutral (neither attracting nor repelling). On the left side, cycle C can be either attracting (if there exists at least one streamline that approaches the cycle), repelling (if there exists at least one streamline that diverges from the cycle), or neutral (neither). Symmetrically, a backward-stable cycle cannot be attracting on the right hand side w.r.t direction of backward flow. It can only be repelling or neutral. On its left, it can be attracting, repelling or neutral. Note that a forward stable cycle which is repelling on its left is also backward stable. The property of repelling implies that backward streamlines converge to the cycle making it backward stable. Symmetrically, a backward cycle which is attracting on its left is also forward stable. Figure 4.6 demonstrates the computation and classification of exact cycles on the dataset of oceanic currents near Italy.

4.2.4 Separatrices

The traditional approach to compute the topological skeleton is to find all saddles and trace their four *separatrices* until they converge towards an orbit or hit another critical point. Some of the challenges for numerical techniques have been that separatrices by definition are numerically unstable (they are asymptotic paths) and moreover it can be especially difficult to determine when a streamline converges towards an orbit. Quantized flow described by

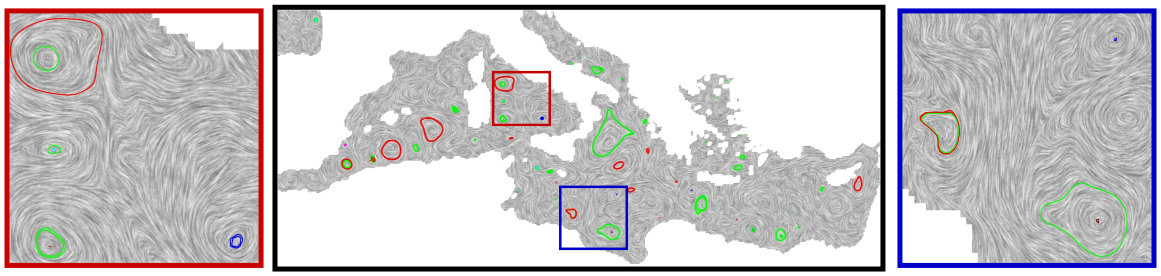


Figure 4.6. Italy (A tile from climate dataset): Exact cycles with classification. Red cycles are attracting on both sides; green cycles are repelling on both sides. There are a total of 412 cycles in this dataset.

bin-graph eliminates these issues making separatrix computation straightforward.

An interesting consequence of the discretized flow is that separatrices are represented as boundaries between bins in stable/unstable manifolds, as opposed to lying on a particular q-streamline. Specifically, every saddle exists on the interior of a triangle with a link structure like the one shown in Figure 4.7. Clearly, the separatrices are represented as the boundaries between the links as shown by the grey dots, as opposed to a specific bin. However, given that q-streamlines when faced with a one-to-many mapping consistently choose the right most bin we simply extract the q-streamlines (forward or backward) of the bins just left of the separatrices. These q-streamlines by definition must be a collection of bins lying on the boundary of a stable or unstable manifold (as anything to the right of them would have crossed the separatrix), and thus would include bins in a different manifold. As a result, we trace a path lying directly adjacent to where the separatrix exists, which is a suitable, (and more importantly) consistent representation of the separatrices.

4.2.5 (Un)Stable Manifolds

As defined earlier in Chapter 3, every limit set has stable and unstable manifolds. All streamlines flowing into (or emerging from) a limit set demarcate the stable (or unstable) manifold of that limit set. We redefine the stable/unstable manifolds in our discrete representation in terms of bins that belong to the q-streamlines that flow into or emerge from the limit sets of a vector field.

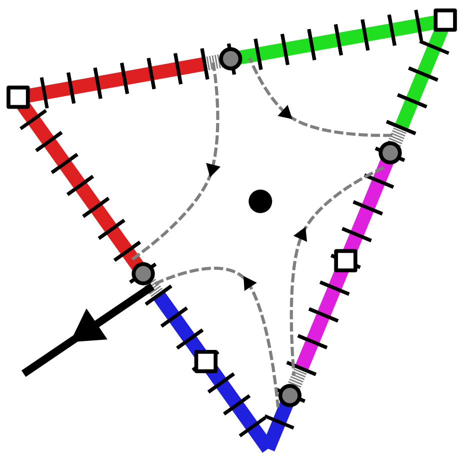


Figure 4.7. In triangles with saddles, we grow q-streamlines at the grey dashed bins to trace separatrices.

CHAPTER 5

IMPLEMENTATION

Conversion of PL vector fields into a quantized form requires construction and efficient storage of quantized edge maps. We describe an algorithm that can construct quantized edge maps that are consistent with each other and make minimal assumptions about the flow in a triangle. We describe a data structure to efficiently store these maps.

5.1 Conversion of Vector Field

In constructing a quantized version of edge map, our objective is to identify contiguous set of inflow bins (inflow interval) that flow to a contiguous set of outflow bins (outflow interval). Such pairs then form a link and linearly map to each other. We first identify all transition points on the edges of a given mesh based on the interpolated vector field in each triangle. An edge is shared between two triangles. A transition point is classified as internal or external independently based on the vector field on either side. Since an interpolated vector field is continuous, we enforce a condition that a transition point cannot be an ITP on both sides but it can be ETP on both sides. This gives us a total of two states of a transition point. In the first state it is an ETP on one side and ITP on the other, while in the second state it is an ETP on both sides. Identification of these transition points also ensures that edge maps agree with the inflow and outflow intervals across the edges.

In the next step, for each triangle, we identify the intial list of inflow and outflow intervals based on its edges and the preidentified transition points. We construct a circular linked list based on the order of the intervals along the triangle boundary. This circular linked list maintains consistency of the map. It avoids any cross linking and overlapping of intervals. A streamline is numerically traced from the mid point of the largest interval in the list and the list is split into two independent lists based on the origin and destination bins identified by the streamline. This can be done because streamlines are not supposed to cross each other. Each time a streamline is traced from an inflow bin to an outflow bin, the triangle region is divided into two regions such that streamlines from one region cannot flow to the other without flowing outside the triangle first. Any streamline traced later cannot have a destination bin outside the list of intervals that it started from. This process is repeated recursively to identify the set of links that can then be put together to construct a quantized version of edge map. Every time a streamline is traced from the largest interval in a list, new links formed are updated into a list using UPDATELINKS() function. We start with a list of intervals ordered along the boundary of triangle. We call the function SPLIT(I, T) recursively until all links are formed. Alogrithm 1 depicts the pseudocode. I is the circular linked list of intervals and T is the set of transition points already identified such that they are consistent across edges with other triangles. This algorithm assumes that the first order critical points in each triangle have been computed and are available as zero length intervals in the list for mapping / forming links. We modify our function for computing numerical streamline such that if a streamline flows to a bin that is not in the list of intervals, then we snap the streamline to the nearest available bin within the list.

5.1.1 Consistency

There are two numerical methods involved in the construction of quantized maps. These are the indentification transition points on edges and computation of streamlines. We compute transition points by solving a simple linear system of equations and compute streamlines using the Local Exact Method (LEM) [20] which are highly accurate but can still cause inconsistencies due to floating point truncations and numerical errors. This is the primary motivation for constructing the quantized edge maps that enforce consistency of flow. Consistency is enforced since links are not allowed to overlap or intersect, and mapping of bins within each link is linear and order preserving.

5.1.2 Map Data Structure

Each map once constructed is converted to a storage efficient data structure. We store a quantized edge map as an array of intervals. Each interval contains a k-bit integer as the starting bin number of that interval on an edge. Bins on each edge of the given mesh are considered to be numbered in an orientation from the lower indexed vertex to the higher indexed vertex for that edge. The number of bins to be used per edge is a user input parameter entered before contruction of quantized edge maps. Thus an interval spans from

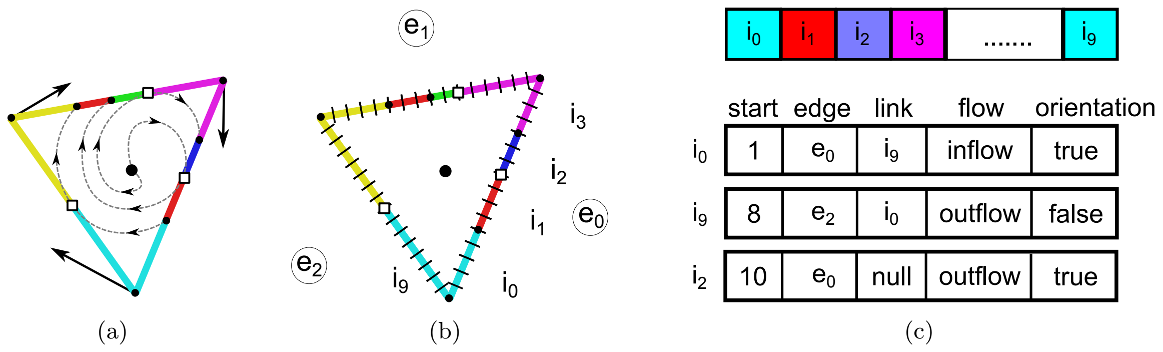
Algorithm 1 Quantized Construction

```
 1: procedure SPLIT($I$, $T$)
 2:     if $|T| > 1$ then
 3:         Cast a streamline from largest interval in $I$.
 4:         Divide $I$ into $I_1, I_2$ and $T$ into $T_1$, $T_2$.
 5:         SPLIT($I_1, T_1$). SPLIT($I_2, T_2$).
 6:     else if $|T| = 1$ then
 7:         if $t \in T$ is ETP and $|I| \leq 2$ then
 8:             updateLinks()
 9:         else if $t \in T$ is ETP and $|I| > 2$ then
10:             Cast a streamline from largest interval in $I$.
11:             Divide $I$ into $I_1, I_2$ and $T$ into $T_1$, $T_2$.
12:             SPLIT($I_1, T_1$). SPLIT($I_2, T_2$).
13:         else if $t \in T$ is an ITP then
14:             Find ITP images
15:             updateLinks()
16:         end if
17:     else
18:         if $|I| > 2$ then
19:             Find SepX points.
20:         end if
21:         updateLinks()
22:     end if
23: end procedure
```

the begin-bin number that is stored in that interval upto the begin-bin number minus 1 of the next interval in the array. This data structure is illustrated in Figure 5.1. Two bytes per interval are bit-packed with following information:

- 2-bits for edge number within the face since bin numbering is only unique per edge;
- a 1-bit flag indicating whether the interval is inflow or outflow;
- 1-bit indicating orientation of the interval relative to the bin numbering;
- and remaining 12-bits for the index of the paired origin/destination interval in the array.

Typically we use 4 bytes for storing begin-bin number and 2 bytes of other information. Hence, the size of an interval amounts to 6 bytes.



	start	edge	link	flow	orientation
i$_0$	1	e$_0$	i$_9$	inflow	true
i$_9$	8	e$_2$	i$_0$	outflow	false
i$_2$	10	e$_0$	null	outflow	true

Figure 5.1. Data structure for quantized edge maps. (a) A piecewise linear flow in a triangle, (b) corresponding quantized edge map, and (c) storage of quantized edge map as an array of intervals in a memory efficient data structure.

5.2 Cycle Detection

Quantized cycles can be considered as strongly connected components (SCC) of the bin-graph. Identification of an SCC is a well known problem in graph theory. However in practice, bin-graphs can be very large to be process directly. Instead, we use a second graph, called the link-graph, as a coarser representation of the vector field. To define the link-graph it is useful to realize that this representation contains two types of mappings: the mapping within each triangle and the mapping (across edges) between neighbor triangles. The bin-graphs directly encode the former and the latter is a trivial one-to-one mapping since neighboring triangles share bins. However, when considering links defined as pairs of sequences of source and destination bins the intratriangle map becomes implicit, while the intertriangle map is nontrivial.

Consider the bin-graph of Figure 4.4. The intratriangle mappings are indicated by the arrows while the intertriangle map is implicit in the ordering of bins. Conversely, Figure 5.2(a) shows the corresponding link-graph where the nodes (boxes) represent the set of bins indicated by the color and the edges encode that links have overlapping sequences of bins across an edge. For example, the red link's destination bins are split between the blue and orange link and thus the red node has two directed edges to the blue and orange nodes, respectively. Thus, the link-graph stores the intertriangle maps explicitly (as edges), while the intratriangle map is maintained internally within each node as a rasterization between two sequences of bins. Initially, nodes in the link-graph represent entire links, but as will be discussed below, our algorithm for cycle detection splits and prunes these sequences,

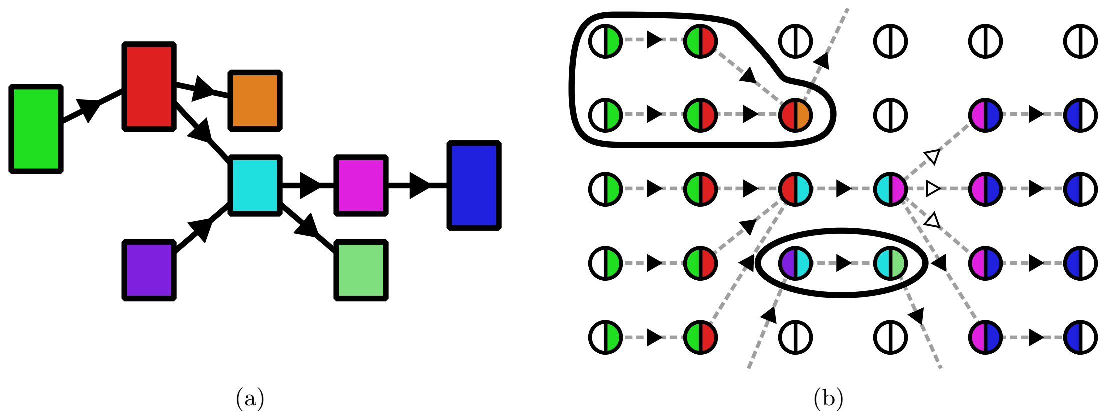


Figure 5.2. Closed streamline detection using graphs. (a) Link-graph corresponding to the links illustrated in Figure 4.3. (b) Assuming there is a closed q-streamline running from the green link-node on the left to the blue link-node on the right, the MSCC computation will remove the orange, purple, and light green nodes in the center. Next, pruning will reduce the intervals in the green, red, and cyan links as shown by the circled bins since their q-streamlines leave the MSCC.

maintaining contiguous subsets of links as nodes.

Definition 4 (Link-Graph) *A* link-graph *is a directed graph $G_L(V_L, E_L)$ such that:*

- $V_L = \{\eta_i\}$, *where each node η_i represents a contiguous sequence of bins from a single link. We call η_i a* link-node.
- $E_L = (\eta_1, \eta_2)$, *such that η_1 and η_2 share at least one bin.*

Since the link-graph by definition ignores the intratriangle mapping it represents paths between links for which no q-streamline exists. For example, the dark green and light green link in Figure 5.2(a) are connected even though no q-streamline exists that shares both light and dark green bins, see Figure 4.3. However, the link-graph is a conservative representation, meaning that if two links share a q-streamline they are connected in the link-graph yet the reverse is not necessarily true. More importantly for cycle detection is the following theorem.

Theorem 5 *Consider a link-graph and its forward and backward bin-graphs. If there exists a closed q-streamline in one of the two bin-graphs, then there exists a corresponding cycle of edges in the link-graph.*

Proof: By construction, each pair of bins b_i, b_j with $\xi^+(b_i) = b_j$ is represented by the same link-node. Thus, if there exists a closed q-streaming $(b_0, b_i, \ldots, b_n)$ with $\xi^+(b_i) = b_{i+1}$ there must exists a cycle in the link-graph. The argument for a cycle in the backward bin-graph is symmetric.

Exploiting the conservative nature of the link-graph we extract closed q-streamlines by iteratively finding maximal strongly connected components (MSCCs) in the link-graph and refining the graph through *graph-cut* and *graph-prune* operations. Graph-cut operations cut a strongly connected component of a link-graph by computing q-streamlines starting at the midpoint of the widest nodes (nodes with widest interval of bins). Since no other q-streamline can cross over the computed one, we can split the link-graph along this q-streamline. This operation allows to quickly cut away large parts of the graph before graph-prune is used. The progressive reduction in link-graphs by the process of graph-cutting is demonstrated in Figure 5.3. Graph-prune operation refers to reducing the bin intervals in each node by identifying the bins that flow into an MSCC from outside or flow outside from it. Graph-prune operation is slower since it requires inspection and possible update of every node of an MSCC. Graph-prune operation, however, allows to converge to the closed q-streamlines once the nodes reach smaller widths.

5.2.1 Link-Graph Data Structure

Even though the link-graph is a coarser representation of flow, storing a link graph of a large dataset can be expensive. We design an efficient data structure to store a link-graph. This data structure is primarily based on maintaining adjacency list in every node. But we want to avoid storing a linked list or a variable length array in every node due to storage costs. Instead, we store only four pointers in every node which accounts for all the connectivities in the neighbourhood. Every node has a foward, backward, origin-side and destination-side pointer. Origin-side pointers are used to connect adjacent nodes inside a face that share the same origin edge in the counter-clockwise direction. Similarly, destination-side pointers are used to connect adjacent nodes inside a face that share the same destination edge in the counter-clockwise direction.

Consider observing a node n with the direction of flow being upwards as shown in Figure 5.4. We know that each node contains an origin interval (O_n) and a destination interval (D_n). All outgoing nodes $F = \{f_0, f_1, f_2, ...\}$ of n are the nodes that belong to the face across the destination edge of n and share bins with D_n. Forward pointer of n_0

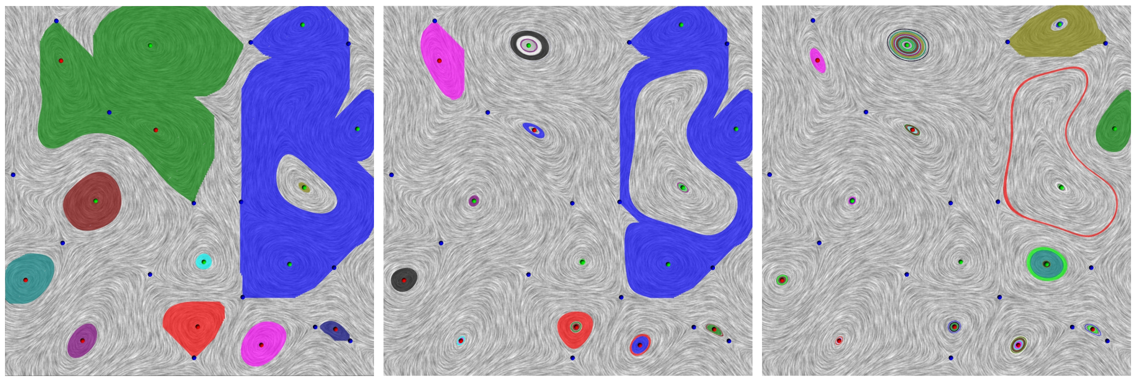


Figure 5.3. Graph-splitting progressively convergence to cycles. Left to right:a gradual reduction in the size of regions.

points to the first outgoing node (f_0), i.e., the first node to the extreme left across the destination edge such that $D_n \cap O_{f_0} \neq \phi$. Remaining outgoing nodes can be found by dereferencing the origin-side pointer of f_0 and the nodes to follow and then testing for the condition $D_n \cap O_f \neq \phi$ where $f \in F$. Incoming nodes for n are found symmetrically. The backward pointer of n points to a node b_0, which is the first node on the extreme right across the origin edge of n such that $O_n \cap D_{b_0} \neq \phi$. Remaining incoming nodes are found using destination-side pointers and testing for the condition $O_n \cap D_b \neq \phi$.

5.3 Complete Topological Segmentation

Computing a complete topological segmentation includes identification of stable and unstable manifolds of all the limit sets of the vector field. For two-dimensional steady vector fields, the limit sets are critical points and closed streamlines. We have already defined the stable and unstable manifolds for these structures in the discrete representation in section 4.2.5. Using this definition we observe that tracing all separatrices and identifying all cycles (closed q-streamlines) delineates the segmentation, which is referred to as topological skeleton. Our algorithm to compute this segmentation first computes the topological skeleton and then uses floodfill approach to identify all regions of segmentation. A region of this segmentation is stored as indices of completely covered triangles and partially touched faces in form of quads. As an approximate rendering of these regions for large datasets, we use only the faces identified to shade these regions. This is a quick way to render the segmentation to get a high level view of a large dataset. Figure 5.5 demonstrates the floodfill approach and region growing approach for computing stable and unstable manifolds.

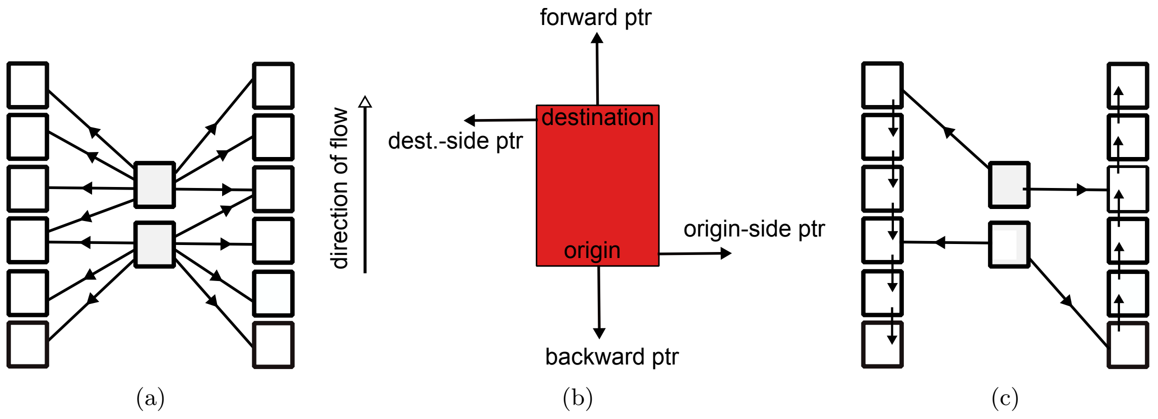


Figure 5.4. Link-graph data structure. (a) A simple link-graph using a variable sized array in every node for storing incoming and outgoing nodes. (b) Efficient data structure for a node. (c) Link-graph stored using the suggested data structure.

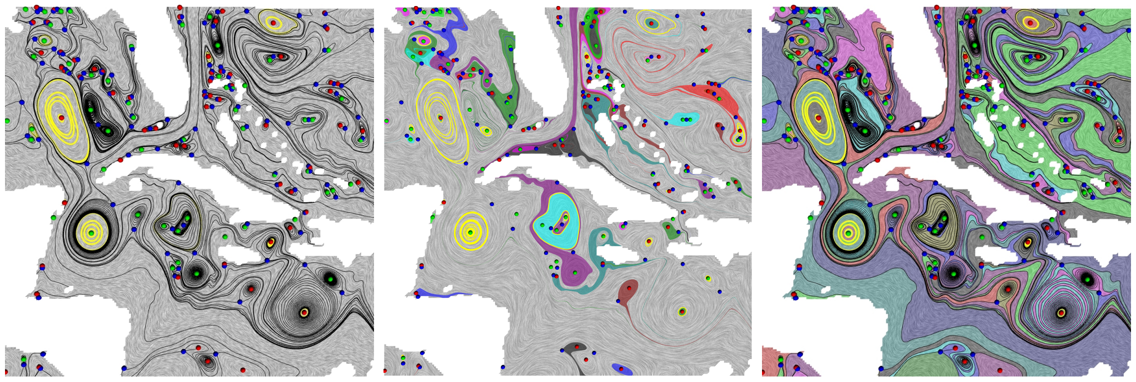


Figure 5.5. Topological skeleton / segmentation: Cuba tile from climate dataset on the left shows its topological skeleton (cycles in yellow and separatrices in black). Image in the center shows topological segmentation generated using region growing approach for computing stable/unstable manifolds. Image on the right shows topological segmentation produced using floodfilling on skeleton.

CHAPTER 6

RESULTS AND DISCUSSION

For demonstrating that our new representation produces consistent results, we show computation of topological skeleton over a synthetic dataset in Figure 6.1 that is known to produce inconsitent result in case of numerical integration method. With this guarantee, it is possible to compute topological skeletons of complex datasets as shown in Figure 6.2.

In Table 6.1 we show the computational performance of our application on all the datasets used in this document. This includes size of the datasets in terms of number of triangles, memory footprint for representing quantized flow in each dataset and time required for conversion of vector field to quantized flow as well as time required for approximate and exact cycle detection.

Cycle detection is the process that requires the most amount of memory given the comprehensive nature of the algorithm. Depending on the size of the dataset and amount of refinement done on the quantized maps, the initial construction of link-graph and subsequent cutting of the graph can increase the number of nodes quickly. To contain the memory footprint during cycle detection, we use a process to cut the domain into smaller pieces that can be processed separately. We partition the domain / mesh into subsets by propagating all separatrices in the quantized flow. We know that quantized streamlines do no intersect each other and hence any cycle is completely contained within such a subset of the domain. Processing each subset separately helps to keep the working memory size under control. Note that this partitioning of domain is flow dependent and the subsets sizes may vary based on flow. But we observe that for most of the time large datasets will provide sufficient separatrices to cut the domain into manageable subsets. Approximate cycles have been detected with an error threshold of 10,000 bins. This threshold indicates that all nodes in the link-graphs that represent approximate cycles are smaller in width than 10,000 bins. Time required to compute exact cycles is higher than approximate cycle computation and it

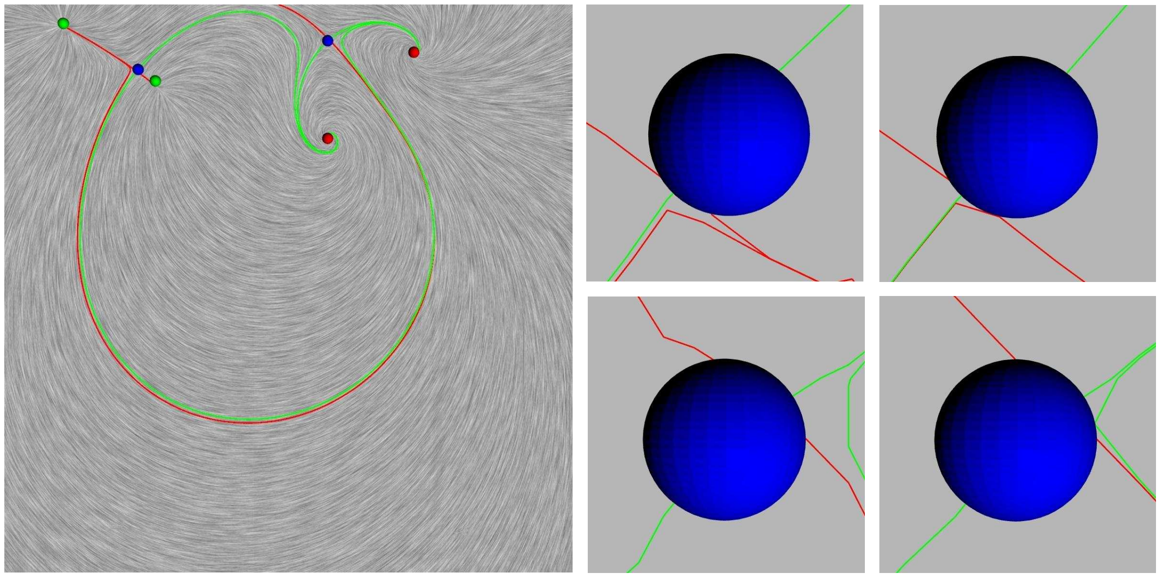


Figure 6.1. Synthetic dataset (100 x 100): **Left:** Shown with inconsistent topological skeleton due to numerical errors. There are two saddles in the dataset. One on the left (saddle1) and another on the right (saddle2). **Middle:** Zoomed view of saddles with unrefined (base) maps. **Right:** Zoomed view of saddles with refined maps. Saddle1 is shown as the top zoom window and saddle2 is shown as bottom zoom window. Both, coarse and refined maps demonstrate consistency. Follow the green (outer) separatrix that flows downward from saddle1. It remains outside of the curved path, i.e., to the right hand side of the red separatrix as shown in the zoomed image of saddle2.

is highly flow dependent. The HCCI dataset takes the longest since there are much higher number of cycles as compared to other datasets and it represents incompressible flow.

For conversion of vector fields into quantized flow, we use multi-threading to process multiple triangles simultaneously. We maintain consistency of flow by setting global flags before hand for inflow/outflow consistency between triangles. The numbers reported in Table 6.1 use 16 threads for quantized flow conversion.

Quantized flow is a discrete representation of two-dimensional steady state vector fields, that represents flow as a large graph of bins over the edges of a triangulation of domain. This representation opens a way for using discrete algorithms for processing flow data with approximation capabilities similar to that of interpolation and numerical integration techniques. Since quantized flow is essentially a graph, efficient graph based alrogithms can be used to extract topological structures from the flow. We essentially focus on PL vector fields over triangle meshes, but the representation itself can be easily extended to quad

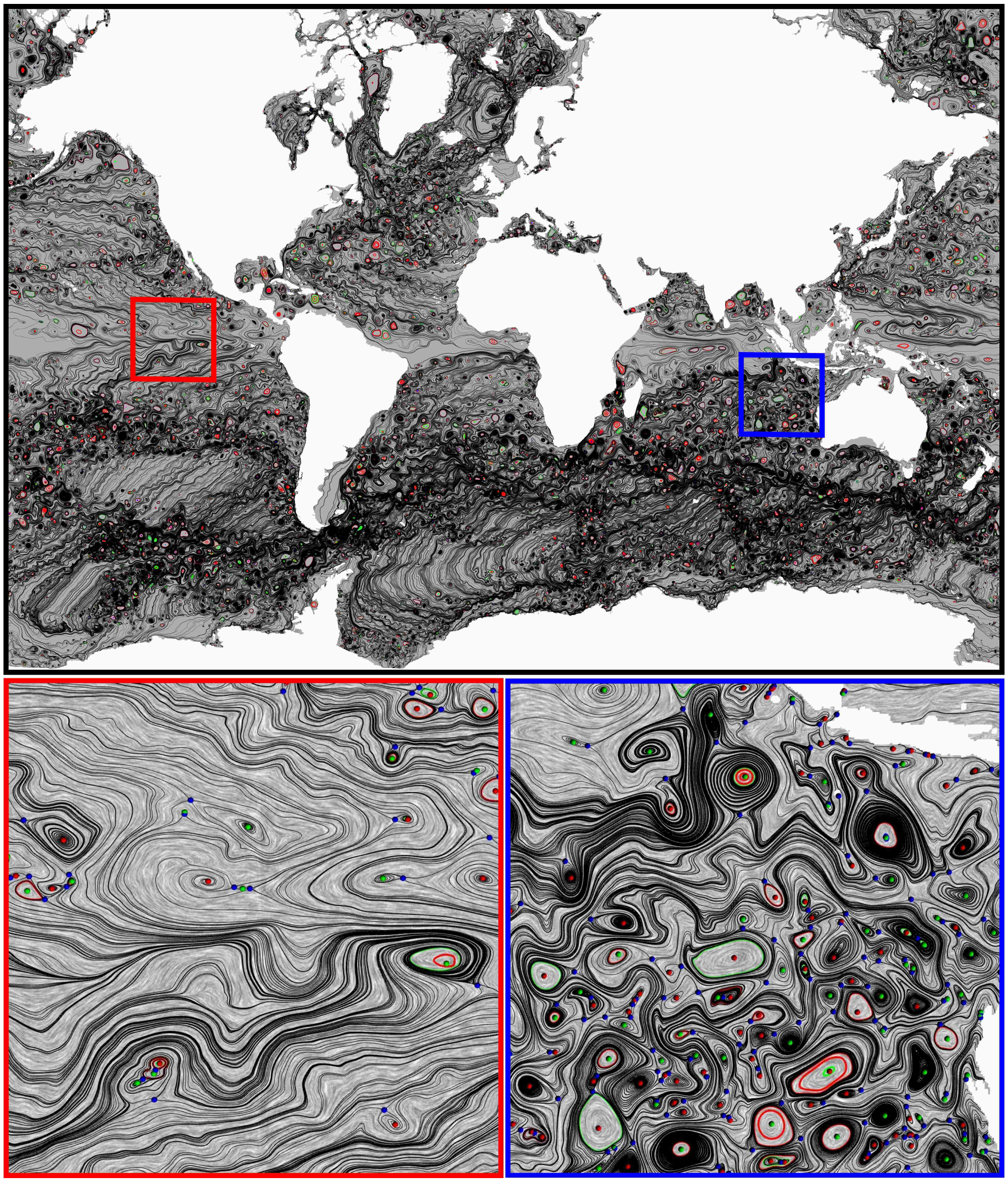


Figure 6.2. Climate dataset (3600 x 2400):Topological skeleton with 54411 cycles. Zoomed windows at the bottom show a dense network of separatrices that are crucial to be computed consistently.

Table 6.1. Performance per dataset:Memory and time are given in MB and Min:Sec, respectively. The approximate orbits converge to roughly 1 million bins (2e-04 of an edge). Machine used:2.64 GHz 8 core CPU with 6GB RAM.

Dataset (Figure)	Intervals	Maps		# Orbits (Time)	
		Memory	Gen. time	Approx.	Exact
Italy currents (4.6)	56,071	38.9	17:01	200 (0:53)	412 (0:57)
Cuba currents (5.5)	62,882	34.4	14:23	149 (0:32)	299 (0:37)
HCCI (5.3)	816,642	196	70:58	376 (05:38)	5018 (78:12)
Climate (6.2)	10.6M	948	47:30	16818 (78:49)	54411 (86:52)

meshes and larger polygons. By developing a more general algorithm for construction of quantized maps, we believe that this representation can also be extended for approximating vector fields with higher interpolation methods. Of course, certain assumptions made for PL vector fields, like a single transition point per edge and a single critical point per polygon will not hold. Since, quantized flow represents boundary behaviour of flow, integration of streamlines within a polygon can be considered a black box. Given such a black box, ideally it is possible to construct a quantized map for each polygon providing a general construction process. Quantized flow also represents vector fields with varying degrees of accuracy as desireable and based on available resources like memory. We provide a method similar to edge maps for improving the accuracy of quantized flow as compared to its interpolated counterpart. Also, varying degrees of precision (granularity) can be easily achieved by using different number of bins per edge. This is particularly useful for faster analysis of data using lesser memory.

Future development of this work should include investigation for a better form of mapping of bins within a link. Currently we use a linear map. Using higher order mapping can provide higher accuracy with fewer links. This will help in reducing the overall memory footprint of the representation. Simplifying detected topological features would be of interest, since the representation is approximate and certain features such as clusters of cycles or critical points could be refinement artifacts rather than stable features of the flow. For example, if a dataset has slowly spiraling sinks and sources, these could appear to be large clusters of cycles under approximation and lesser granularity. Features that appear and disappear at different refinement levels are sensitive to accuracy and hence can be considered less important as compared to more stable features that are present even in the most approximate representation of the flow. Trying for more and more accurate quantized flow with respect to a particular interpolation like linear interpolation may become an overkill given that choosing a different interpolation can give entirely different results. Extending this representation to three-dimensional vector fields will be challenging. Even if it is possible to represent a three-dimensional mesh simplex like a tetrahedron as a set of bins on its boundary, there is no notion of ordering to this set of bins. Constructing an order preserving map that can consistently map inflow bins to outflow bins may not be trivial. Perhaps, it will be helpful to start with a discrete represent of higher forms of flow descriptors like stream surfaces rather than streamlines.

REFERENCES

[1] Bhatia, H., Jadhav, S., Bremer, P.-T., Chen, G., Levine, J. A., Nonato, L. G., and Pascucci, V. Edge maps: Representing flow with bounded error. In *IEEE Pacific Visualization Symposium* (2011), pp. 75–82.

[2] Bresenham, J. Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1 (1965), 25–30.

[3] Chen, G., Mischaikow, K., Laramee, R. S., Pilarczyk, P., and Zhang, E. Vector field editing and periodic orbit extraction using Morse decomposition. *IEEE Trans. Vis. Comput. Graph. 13*, 4 (2007), 769–785.

[4] Chen, G., Mischaikow, K., Laramee, R. S., and Zhang, E. Efficient Morse decompositions of vector fields. *IEEE Trans. Vis. Comput. Graph. 14*, 4 (2008), 848–862.

[5] de Leeuw, W., and van Liere, R. Collapsing flow topology using area metrics. In *Proc. of IEEE Visualization '99* (1999), pp. 349–354.

[6] Forman, R. Combinatorial vector fields and dynamical systems. *Math. Z. 228*, 4 (1998), 629–681.

[7] Garth, C., and Tricoche, X. Topology- and feature-based flow visualization: Methods and applications. In *SIAM Conference on Geometric Design and Computing* (2005).

[8] Garth, C., Tricoche, X., and Scheuermann, G. Tracking of vector field singularities in unstructured 3d time-dependent datasets. In *15th IEEE Visualization Conference* (2004), pp. 329–336.

[9] Globus, A., Levit, C., and Lasinski, T. A tool for visualizing the topology of three-dimensional vector fields. In *IEEE Visualization* (1991), pp. 33–41.

[10] Gyulassy, A., Bremer, P.-T., Pascucci, V., and 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.

[11] Gyulassy, A., Natarajan, V., Pascucci, V., Bremer, P.-T., and Hamann, B. A topological approach to simplification of three-dimensional scalar functions. *IEEE Trans. Vis. Comput. Graph. 12*, 4 (2006), 474–484.

[12] Haller, G. Lagrangian coherent structures and the rate of strain in two-dimensional turbulence. *Phys. Fluids A 13* (2001), 3365–3385.

[13] Helgeland, A., Reif, B., Andreassen, O., and Wasberg, C. Visualization of vorticity and vortices in wall-bounded turbulent flows. *Visualization and Computer Graphics, IEEE Transactions on 13*, 5 (sept.-oct. 2007), 1055 –1067.

[14] Helman, J., and Hesselink, L. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer 22*, 8 (1989), 27–36.

[15] Jadhav, S., Bhatia, H., Bremer, P.-T., Levine, J. A., Nonato, L. G., and Pascucci, V. Consistent approximation of local flow behavior for 2d vector fields using edge maps. In *Proceedings of TopoInVis* (2011). Accepted.

[16] Laney, D., Bremer, P. T., Mascarenhas, A., Miller, P., and 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.

[17] Laramee, R. S., Hauser, H., Zhao, L., and Post, F. H. Topology Based Flow Visualization: The State of the Art. In *Topology-Based Methods in Visualization (Proc. Topo-in-Vis 2005)* (2007), Mathematics and Visualization, Springer, pp. 1–19.

[18] Liu, Z., and II, R. J. M. Robust loop detection for interactively placing evenly spaced streamlines. *Computing in Science and Engineering 9*, 4 (2007), 86–91.

[19] Mahrous, K., Bennett, J., Scheuermann, G., Hamann, B., and Joy, K. I. Topological segmentation in three-dimensional vector fields. *IEEE Transactions on Visualization and Computer Graphics 10* (2004), 198–205.

[20] Nielson, G. M., and Jung, I.-H. Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE Trans. Vis. Comput. Graph. 5*, 4 (1999), 360–372.

[21] Reininghaus, J., and Hotz, I. Combinatorial 2d vector field topology extraction and simplification. In *Topological Methods in Data Analysis and Visualization. Theory, Algorithms, and Applications. (TopoInVis'09)* (2009).

[22] Sadlo, F., and Peikert, R. Visualizing lagrangian coherent structures and comparison to vector field topology. In *Topology-Based Methods in Visualization II* (2008), Springer, pp. 15–30.

[23] Sadlo, F., Peikert, R., and Parkinson, E. Vorticity based flow analysis and visualization for pelton turbine design optimization. In *Visualization, 2004. IEEE* (oct. 2004), pp. 179 – 186.

[24] Scheuermann, G., and Tricoche, X. Topological methods in flow visualization. In *In Visualization Handbook* (2004), C. Hansen and C. Johnson, Eds., Elsevier, pp. 341–356.

[25] Theisel, H., Weinkauf, T., Hege, H.-C., and Seidel, H.-P. Saddle connectors - an approach to visualizing the topological skeleton of complex 3d vector fields. In *Proc. of IEEE Visualization '03* (2003), pp. 225–232.

[26] Theisel, H., Weinkauf, T., Hege, H.-C., and Seidel, H.-P. Grid-independent detection of closed stream lines in 2d vector fields. In *Proceedings of the Vision, Modeling, and Visualization Conference 2004 (VMV)* (2004), pp. 421–428.

[27] Theisel, H., Weinkauf, T., Hege, H.-C., and Seidel, H.-P. Topological methods for 2d time-dependent vector fields based on stream lines and path lines. *IEEE Trans. Vis. Comput. Graph. 11*, 4 (2005), 383–394.

[28] Tricoche, X., Scheuermann, G., and Hagen, H. Continuous topology simplification of planar vector fields. In *Proc. of IEEE Visualization '01* (2001), pp. 159–166.

[29] Wischgoll, T., and Scheuermann, G. Detection and visualization of closed streamlines in planar flows. *IEEE Trans. Vis. Comput. Graph. 7*, 2 (2001), 165–172.

[30] Wischgoll, T., and Scheuermann, G. Locating closed streamlines in 3d vector fields. In *Symposium on Data Visualisation 2002* (2002), VISSYM '02, Eurographics Association, pp. 227–232.