Homology and Cohomology in Visualization: From Vector Fields to Memory Reference Traces

Bei Wang

Scientific Computing and Imaging Institute (SCI), University of Utah

Joint work with Primoz Skraba, Paul Rosen, Guoning Chen, Harsh Bhatia, Valerio Pascucci, Brian Summa, A.N.M. Imroz Choudhury, Mikael Vejdemo-Johansson

Oct 31, 2013

Overview: Applications of (co)homology in visualization

- Part 1: Robustness of critical points for vector field data
  - A Visualizing robustness of critical points for 2D time-varying vector fields
  - B Vector field simplification
  - C Interpreting feature tracking through the lens of robustness
- Part 2: Detecting branching and circular structures in data
  - A Branching and circular structure detection
  - B Application in software visualization

Part 1-A: Robustness

Visualizing Robustness of Critical Points
for Time-Varying Vector Fields

Joint work: Primoz Skraba, Paul Rosen, Harsh Bhatia, Valerio Pascucci

[Wang, Rosen, Skraba, Bhatia and Pascucci 2013]

Motivation

- Vector fields (VFs) represent an important and ubiquitous class of data that arise in many scientific disciplines
- Analyzing critical points and their temporal evolutions plays a crucial role in understanding the behavior of VFs.
- A key challenge is to quantify the stability of critical points: more stable points may represent more important phenomena.

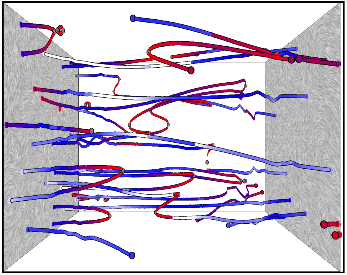
[Video: 2D time-varying VF, combustion dataset]

Robustness [Chazal, Patel and Skraba 2011, 2012]

- Quantify rigorously the stability of critical points.
- Intuitively, the robustness of a critical point is the minimum amount of perturbation necessary to cancel it within a local neighborhood, measured under an appropriate metric.

Contributions

- Interactive exploration of robustness of critical points for 2D time-varying vector fields
- Investigate how the stability of a critical point evolves over time: depends heavily on the global properties of the VF and that structural changes can correspond to interesting behavior
- Study stable and unstable features in real world datasets: combustion and ocean eddie simulation



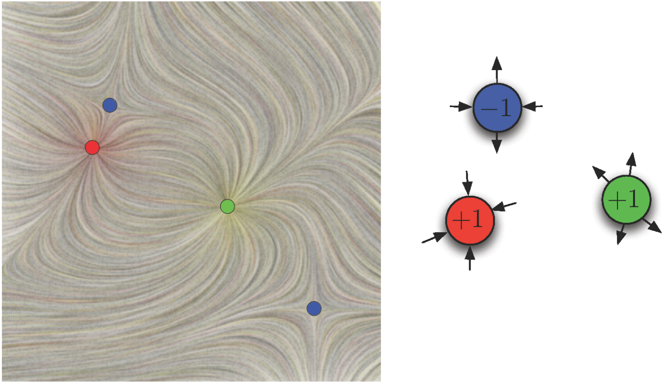
[Videos: Combustion interactive]

Background Via Example: Degree Theory, Merge Tree, Robustness, Well Groups

Well groups [Edelsbrunner, Morozov, Patel 2010, 2011]
Merge tree, robustness [Chazal, Patel and Skraba 2011, 2012]

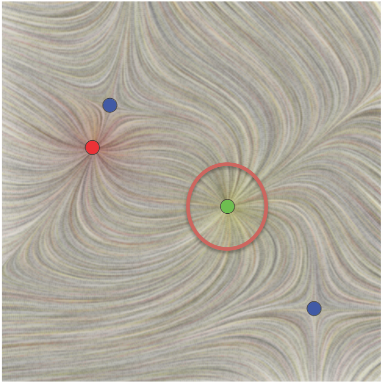
Degrees

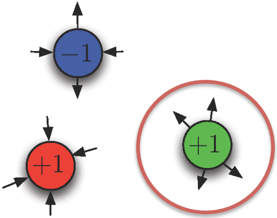
- In 2D, $\deg(x)$ of a critical point x equals its Poincaré index.



Degrees

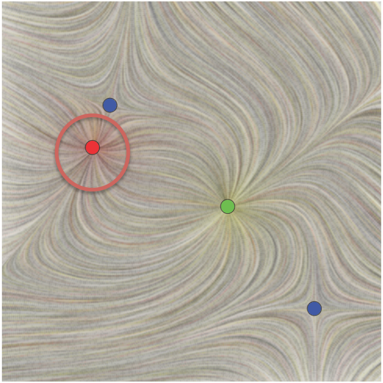
- In 2D, $\deg(x)$ of a critical point x equals its Poincaré index.
- Source $+1$, sink $+1$, saddle -1.

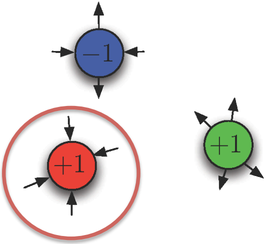




Degrees

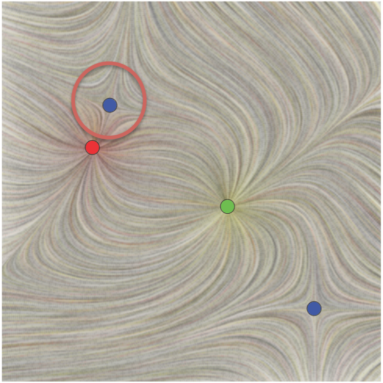
- In 2D, $\deg(x)$ of a critical point x equals its Poincaré index.
- Source $+1$, sink $+1$, saddle -1.

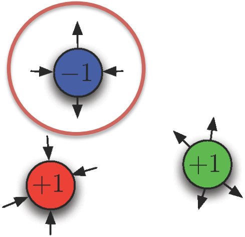




Degrees

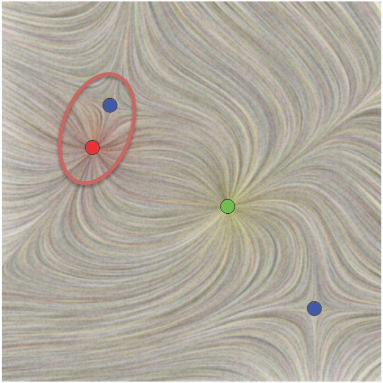
- In 2D, $\deg(x)$ of a critical point x equals its Poincaré index.
- Source $+1$, sink $+1$, saddle -1.

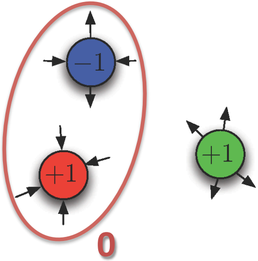




Degrees

- In 2D, $\deg(x)$ of a critical point x equals its Poincaré index.
- Source $+1$, sink $+1$, saddle -1.
- A path-connected component (c.c.) C, $\deg(C) = \sum_i \deg(x_i)$.
- If C in $\mathbb{R}^2$ has degree zero, then it is possible to replace the VF inside C with a VF free of critical points





Merge tree

Given $f : \mathbb{R}^2 \to \mathbb{R}^2$, define its Euclidean norm $f_0 : \mathbb{R}^2 \to \mathbb{R}$ as

$$f_0(x) = ||f(x)||_2$$

For some $r \geq 0$, define the sublevel set of f_0 as

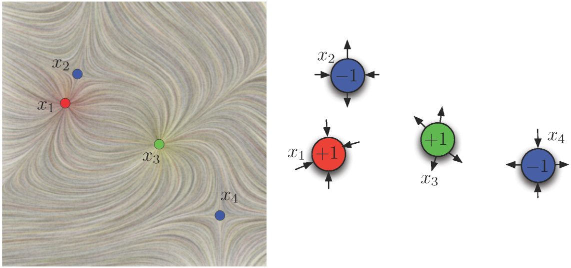
$$\mathbb{F}_r = f_0^{-1}[0, r].$$

A value $r > 0$ is a regular value of f_0 if

- $\mathbb{F}_r$ is a 2-manifold;
- $\forall$ sufficiently small $\epsilon > 0$, $f_0^{-1}[r - \epsilon, r + \epsilon]$ d.r. to $f_0^{-1}(r)$;
- o.w. it is a critical value.

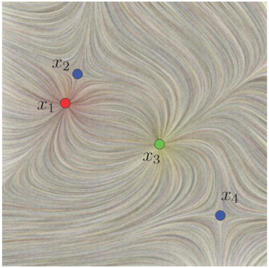
Merge tree of f_0

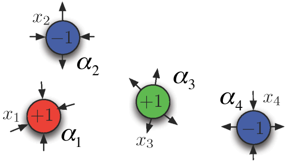
Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
# of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.

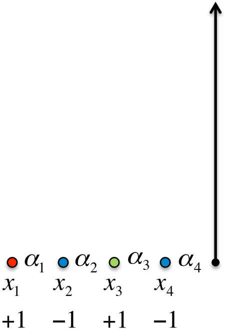


Merge tree of f_0

Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
$\#$ of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.

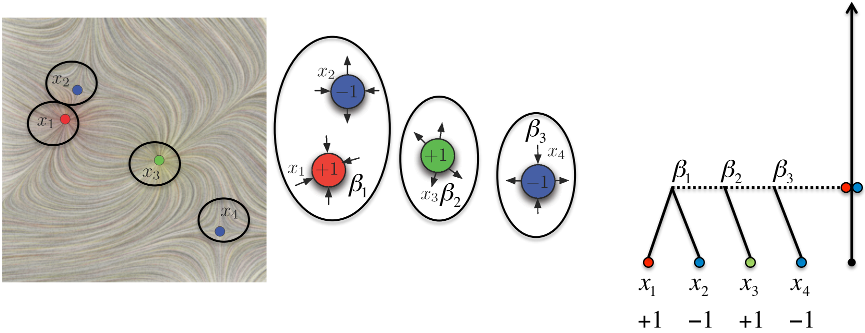






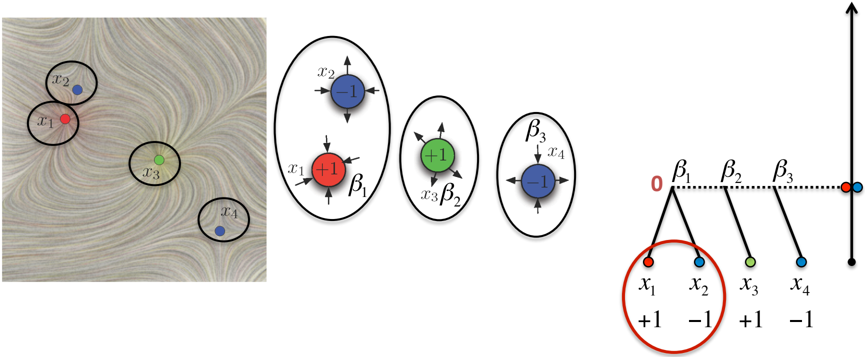
Merge tree of f_0

Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
$\#$ of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.



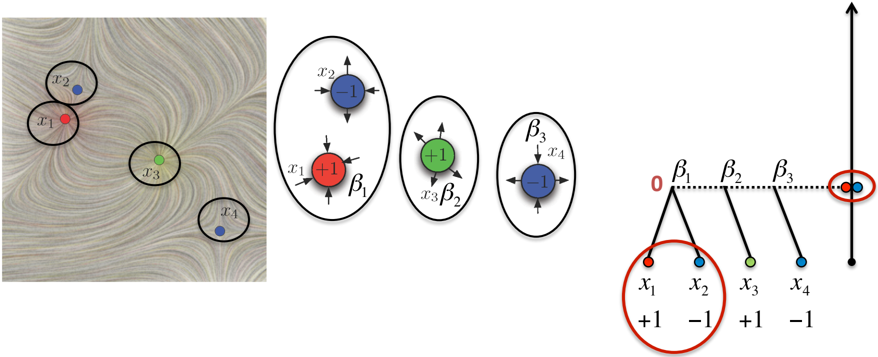
Merge tree of f_0

Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
$\#$ of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.



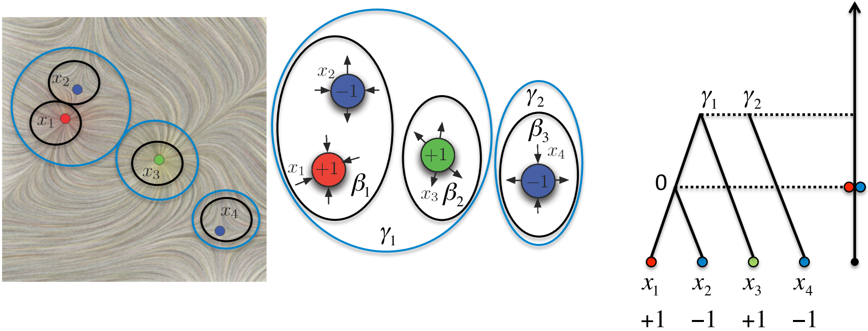
Merge tree of f_0

Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
# of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.



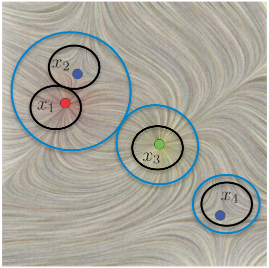
Merge tree of f_0

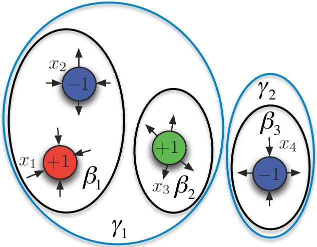
Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
$\#$ of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.

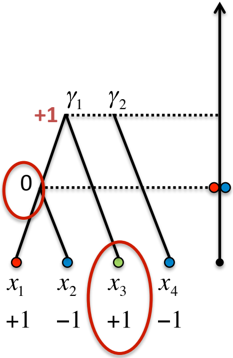


Merge tree of f_0

Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
$\#$ of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.

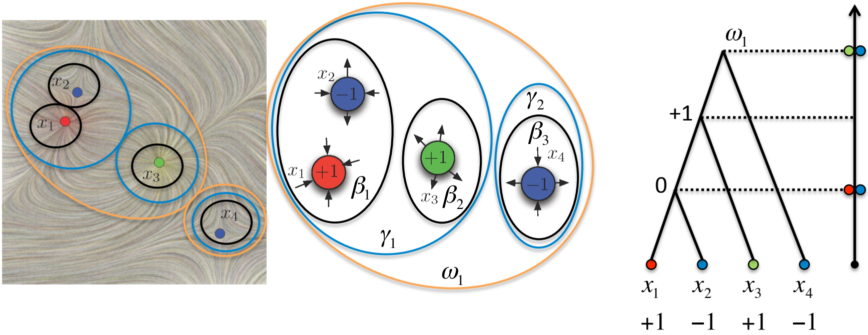






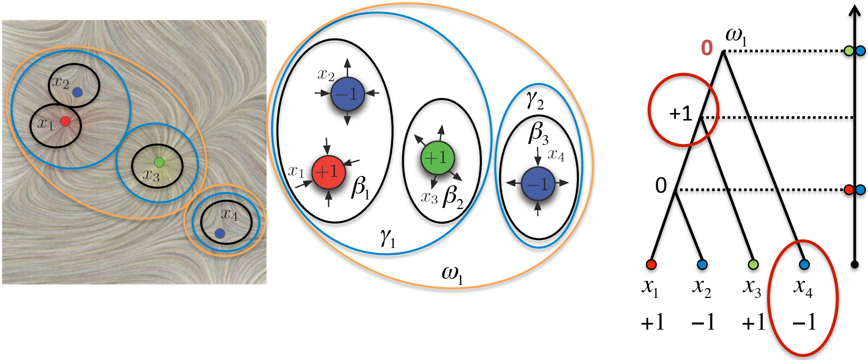
Merge tree of f_0

Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
$\#$ of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.



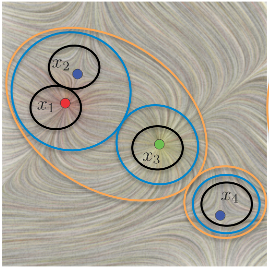
Merge tree of f_0

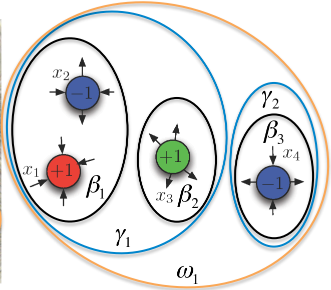
Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
# of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.

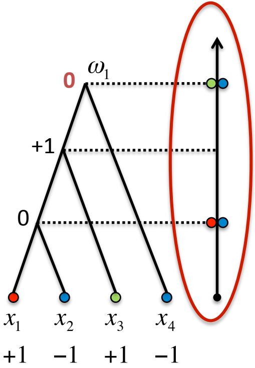


Merge tree of f_0

Track c.c. of $\mathbb{F}_r$ as they appear and merge, as r increases from $-\infty$.
# of c.c. $=$ the rank of 0-dim homology groups for some r, $\mathsf{H}(\mathbb{F}_r)$.

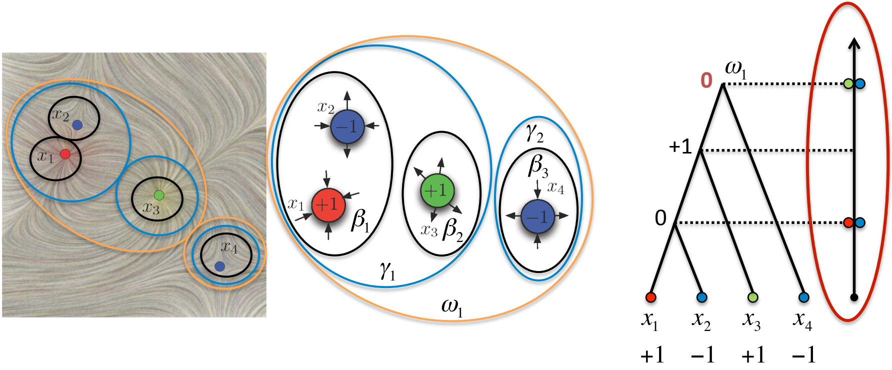






Robustness [Chazal, Patel and Skraba 2011]

The (static) robustness of a critical point is the height of its lowest degree zero ancestor in the merge tree.
Interpretation: robustness is the min amount of pert. necessary to cancel a critical point.



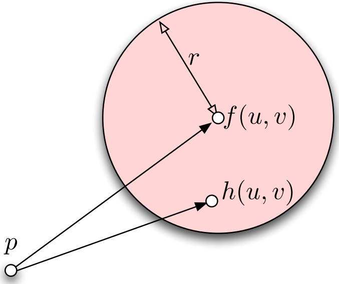
Robustness: $\mathrm{rb}(x_1) = \mathrm{rb}(x_2)$, $\mathrm{rb}(x_3) = \mathrm{rb}(x_4)$.

r-perturbation

Let $f, h : \mathbb{R}^2 \to \mathbb{R}^2$ be two continuous 2D vector fields. Define the distance between the two mapping as

$$\mathrm{d}(f, h) = \sup_{x \in \mathbb{R}^2} ||f(x) - h(x)||_2.$$

We say h is an r-perturbation of f, if $\mathrm{d}(f, h) \leq r$.



Well group (Technical)

Suppose h is an r-perturbation of f.
$\mathbb{H}_0 = h^{-1}(0)$ is the set of critical points of h. We have inclusion:

$$i : \mathbb{H}_0 \to \mathbb{F}_r$$

i induces linear map:

$$j_h : \mathsf{H}(\mathbb{H}_0) \to \mathsf{H}(\mathbb{F}_r)$$

The well group, $\mathsf{U}(r)$, is the subgroup of $\mathsf{H}(\mathbb{F}_r)$, whose elements belong to the image of each j_h, for all r-perturbation h of f:

$$\mathsf{U}(r) = \bigcap_h \operatorname{im} j_h$$

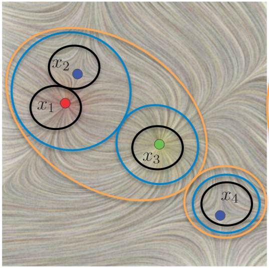
Intuitively, an element in $\mathsf{U}(r)$ is considered a stable element in $\mathsf{H}(\mathbb{F}_r)$ if it does not disappear with respect to any r-perturbation

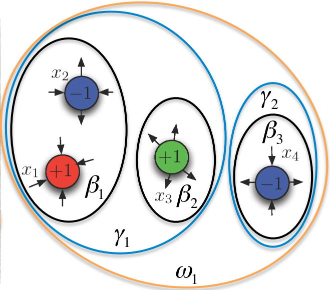
Rank of $\mathsf{U}(0) = \#$ critical points of f.

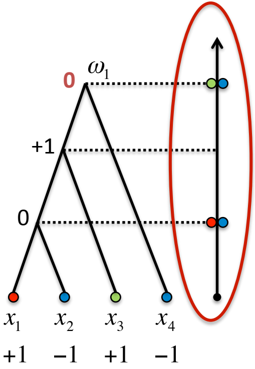
Well diagram $\mathrm{Dgm}(f_0)$ of f_0

A point $r \in (0, \infty)$ belongs to the well diagram of f_0, $\mathrm{Dgm}(f_0)$, with multiplicity k if the rank of the well group drops by k at r.

The augmented merge tree is sufficient to derive its well diagram.







(Static) Robustness Properties

Intuition

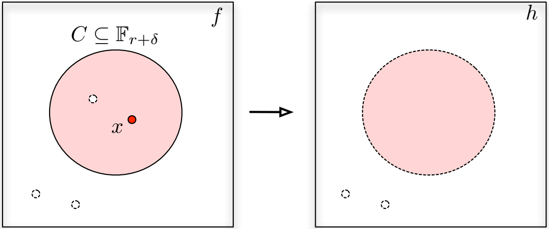
Robustness quantifies the stability of a critical point with respect to perturbations of the vector fields.

Critical Point Cancellation Lemma

Lemma

Suppose a critical point x of f has robustness r. Let C be the c.c. of $\mathbb{F}_{r+\delta}$ containing x, for an arbitrarily small $\delta > 0$. Then, there exists an $(r+\delta)$-perturbation h of f, such that $h^{-1}(0) \cap C = \emptyset$ and $h = f$ except possibly within the interior of C.

Key: need $(r+\delta)$-pert. to cancel critical point of robustness r.



rb(x) = r

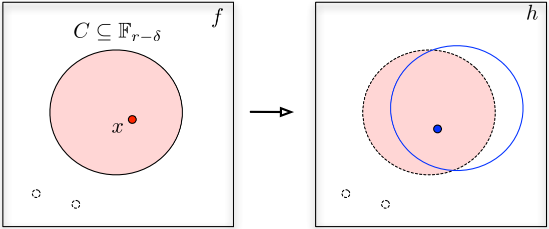
h: $(r+\delta)$-pert s.t. $h^{-1}(0) \cap C = \emptyset$

Degree and Critical Points Preservation Lemma

Lemma

Suppose a critical point x of f has robustness r. Let C be the connected component of $\mathbb{F}_{r-\delta}$ containing x, for some $0 < \delta < r$. For any ϵ-perturbation h of f where $\epsilon \leq r - \delta$, the sum of the degrees of the critical points in $h^{-1}(0) \cap C$ is $\deg(C)$. If C contains only one critical point x, we have $\deg(h^{-1}(0) \cap C) = \deg(x)$. That is, x is preserved as there is no ϵ-perturbation that could cancel it.

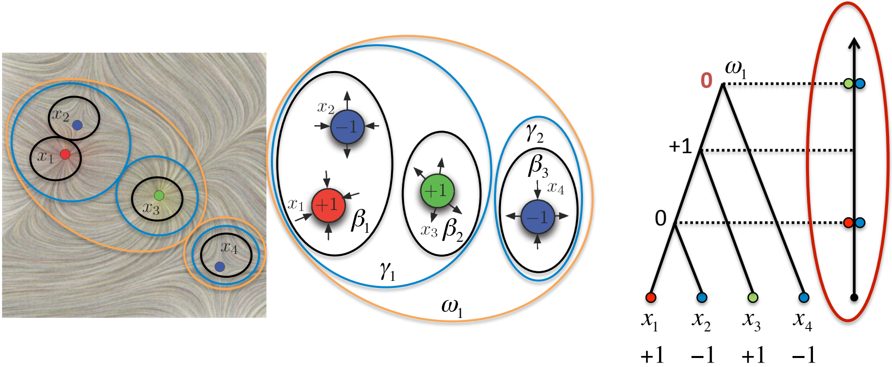
Key: $(r - \delta)$-pert. is not enough to cancel crit. pt. of robustness r.



$\mathrm{rb(x)} = r$

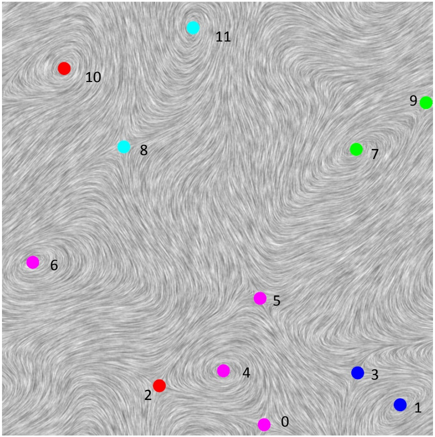
h: ϵ-pert, $\epsilon \leq r - \delta$ s.t.
$\deg(h^{-1}(0) \cap C) = \deg(\mathrm{C})$

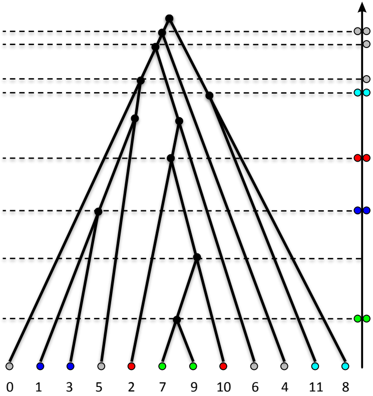
Example revisited



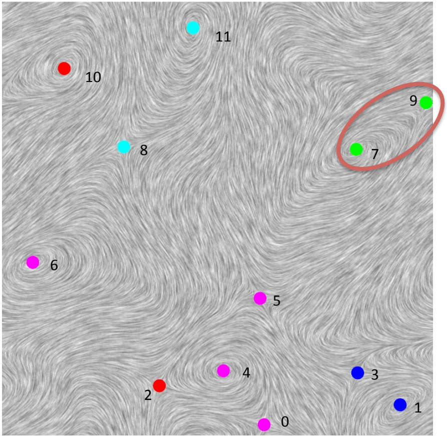
Results

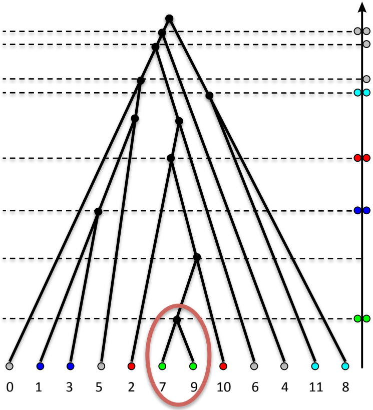
Combustion data



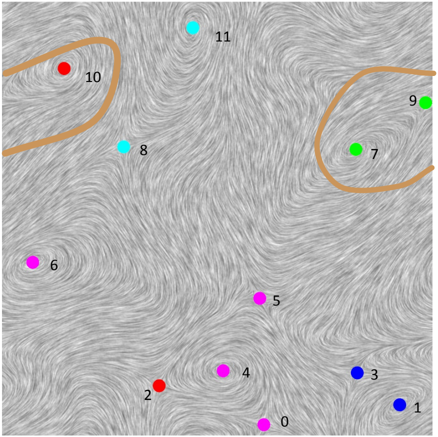


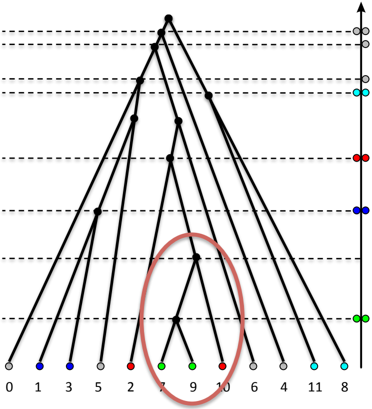
Combustion data



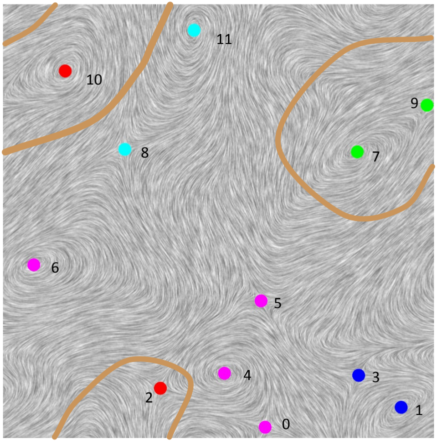


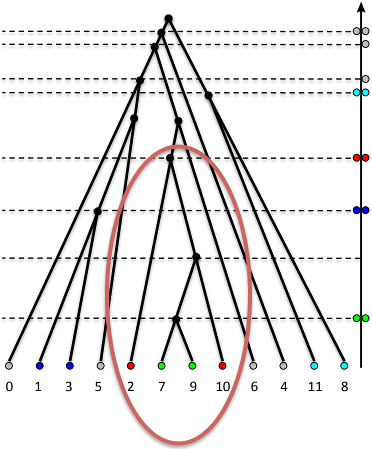
Combustion data



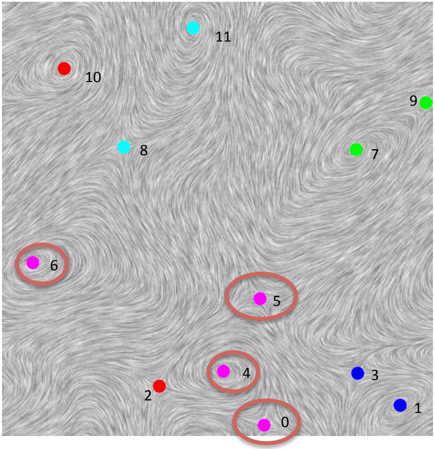


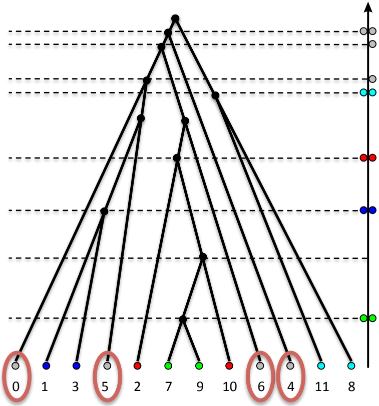
Combustion data



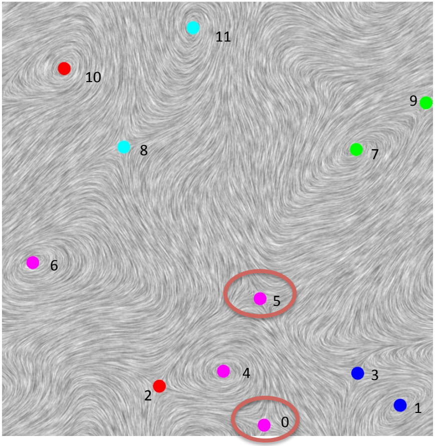


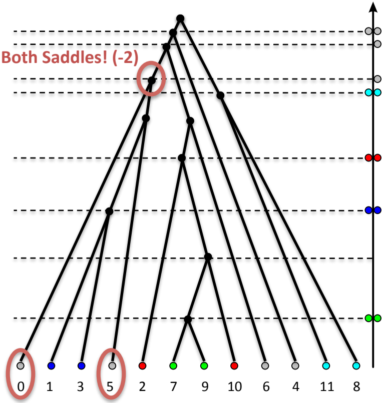
Combustion data



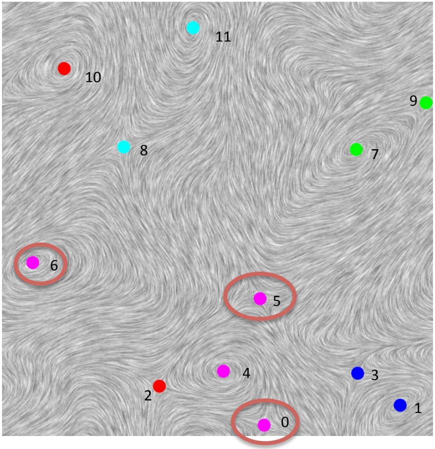


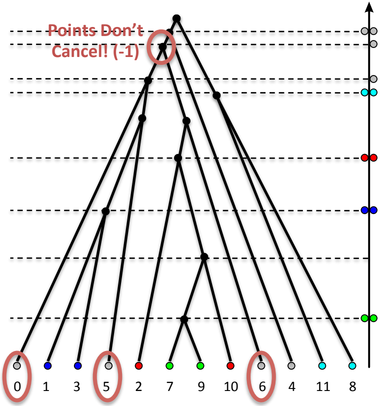
Combustion data



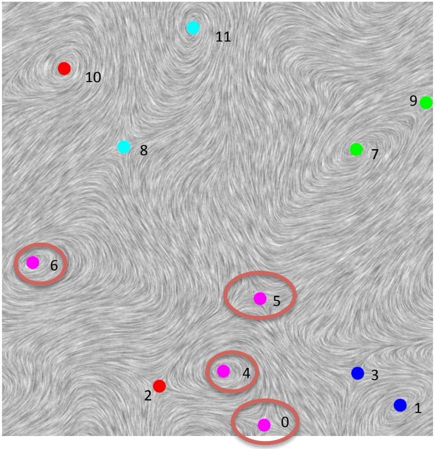

Both Saddles! (-2)

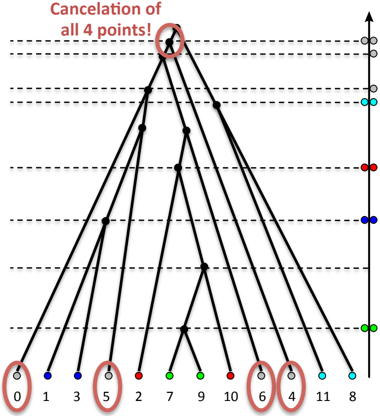
Combustion data





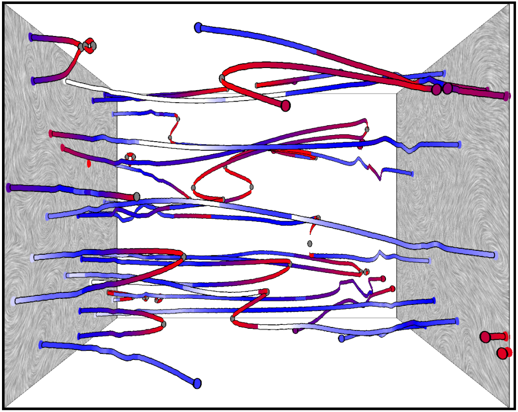
Combustion data




Cancelation of all 4 points!

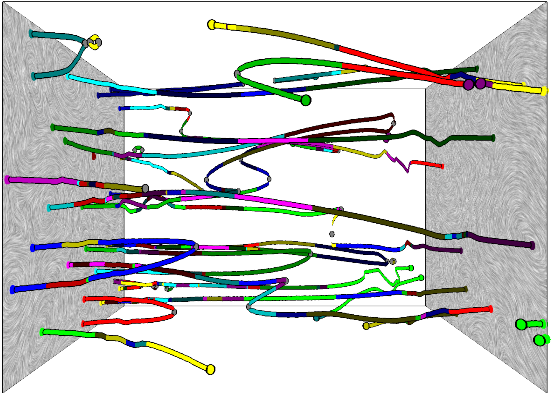
Visualization

- 3D visualization uses time as the 3rd dimension
- Critical points tracked and connected through tubes
- Sequential color map for robustness



Visualization

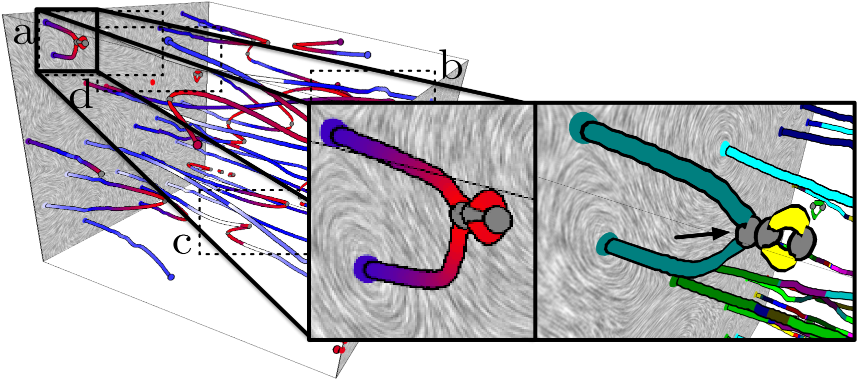
- 3D visualization uses time as the 3rd dimension
- Critical points tracked and connected through tubes
- Sequential color map for robustness
- Categorical color map for pairing information



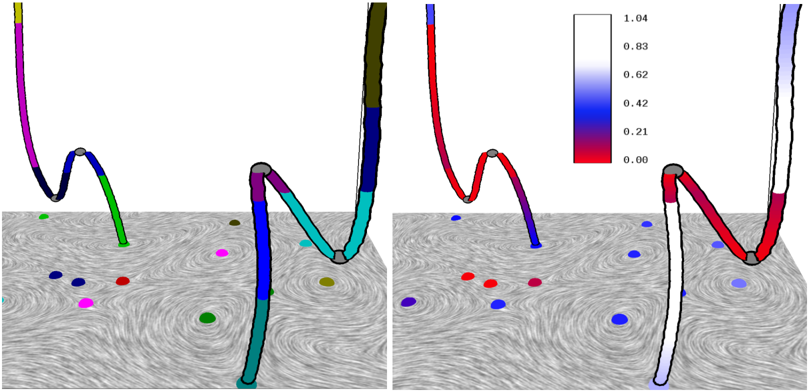
Datasets

- Combustion: homogenous charge compression ignition (HCCI) engine, 640×640 field consisting of 299 time steps [Hawkes, Sankaran, Pebay, Chen 2006]
- Ocean Eddie: Top layer of global oceanic eddy simulation for 2002 [Maltrud, Bryan, Peacock 2010]
  - Central Atlantic Ocean - 60×60, 350 days
  - South Atlantic Ocean - 100×100, 350 days

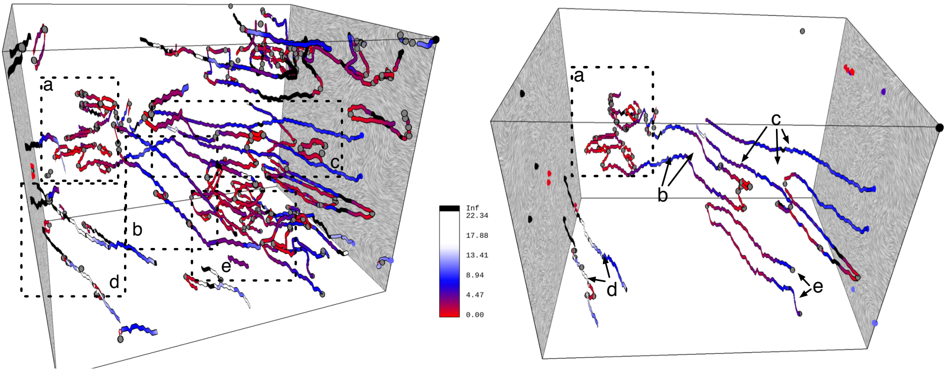
Combustion



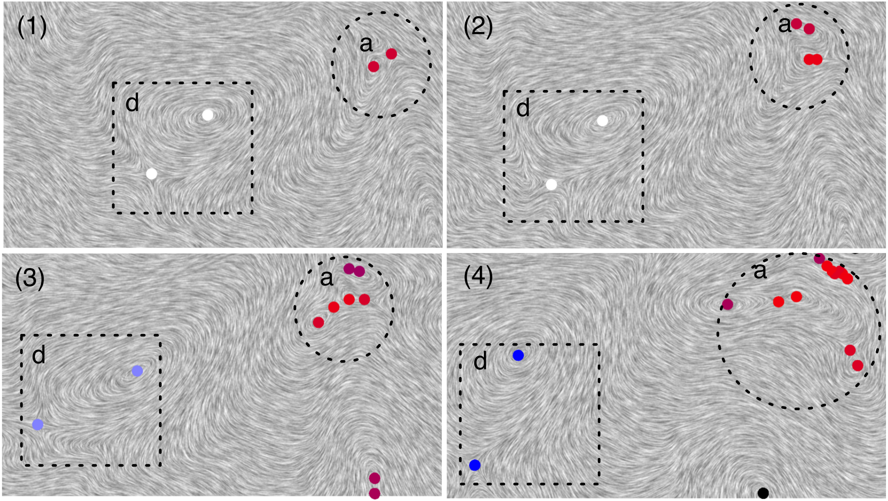
Combustion



Ocean Eddies



Ocean Eddies



Discussions

- Robustness of components in the sublevel sets (over time) [Video]
- Visual clutter
- Better version of robustness-like measure for time-varying VFs

Part 1-B: Robustness

Vector Field Simplification based on Robustness

Joint work: Primoz Skraba, Paul Rosen and Guoning Chen.

[Skraba, Wang, Chen and Rosen 2013 (to appear)]

Motivation

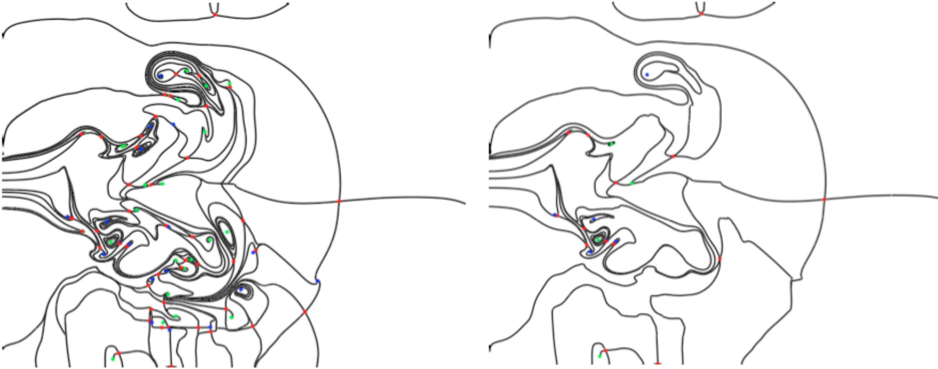
- Gap between increasing size and complexity of VF data and limited bandwidth of our visual perception channel
- Challenge for domain experts: interpret the data (locally or globally)
- e.g. 2D turbulence flows (features are everywhere and feature sizes differ by a few orders of magnitude)

VF simplification:

- Reducing the complexity of the flow by removing features in order of their relevance and importance
- Revealing prominent behavior
- Obtaining a compact representation for interpretation
- Giving a consistent and multi-scale view of the flow dynamics

Topological-skeleton-based simplification

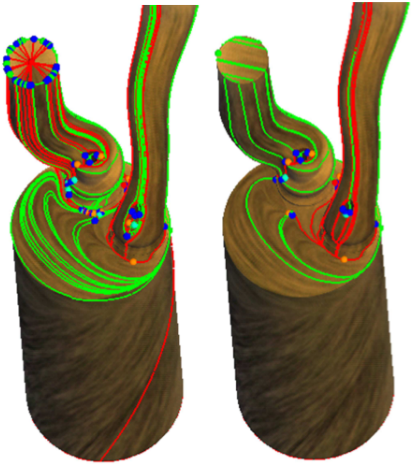
Merging/canceling nearby critical points based on separatrices (dividing domain into regions of uniform flow behavior)



Swirling jet simulation, courtesy of [Tricoche, Scheuermann and Hagen 2001]

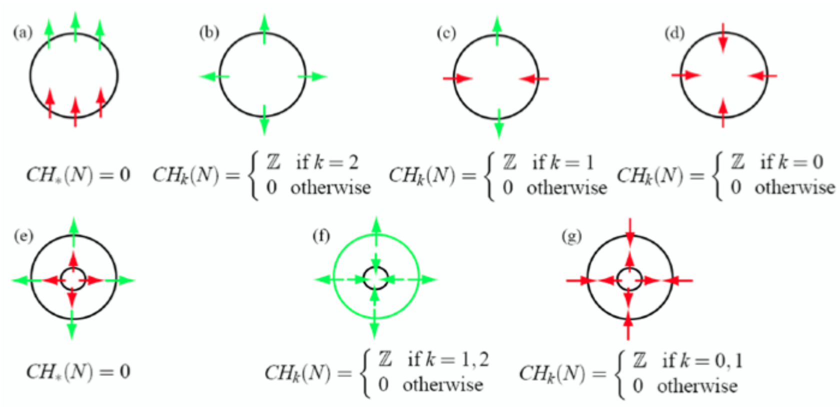
Topological-skeleton-based simplification

Creation and cancellation of fixed points and periodic orbits via Morse decomposition/Conley theory.



Computational fluid dynamics simulation [Chen, Mischaikow, Laramee, Pilarczyk, Zhang 2007]

Conley indices



$CH_*(N)=0$

$CH_k(N)=\begin{cases}\mathbb{Z} & \text{if } k=2\\ 0 & \text{otherwise}\end{cases}$

$CH_k(N)=\begin{cases}\mathbb{Z} & \text{if } k=1\\ 0 & \text{otherwise}\end{cases}$

$CH_k(N)=\begin{cases}\mathbb{Z} & \text{if } k=0\\ 0 & \text{otherwise}\end{cases}$

$CH_*(N)=0$

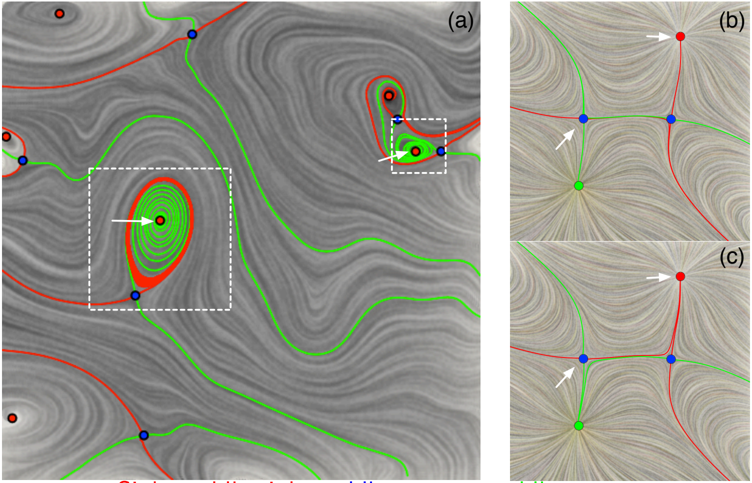
$CH_k(N)=\begin{cases}\mathbb{Z} & \text{if } k=1,2\\ 0 & \text{otherwise}\end{cases}$

$CH_k(N)=\begin{cases}\mathbb{Z} & \text{if } k=0,1\\ 0 & \text{otherwise}\end{cases}$

[Zhang, Mischaikow and Turk 2006]

Topological-skeleton-based simplification

Rely on the stable extraction of the topological skeleton: can be difficult due to instability in numerical integration, especially when processing highly rotational flows. (a) Near Hopf-bifurcations; diff separatrices intersect/switch.



Sink, saddle-sink, saddle, source, saddle-source

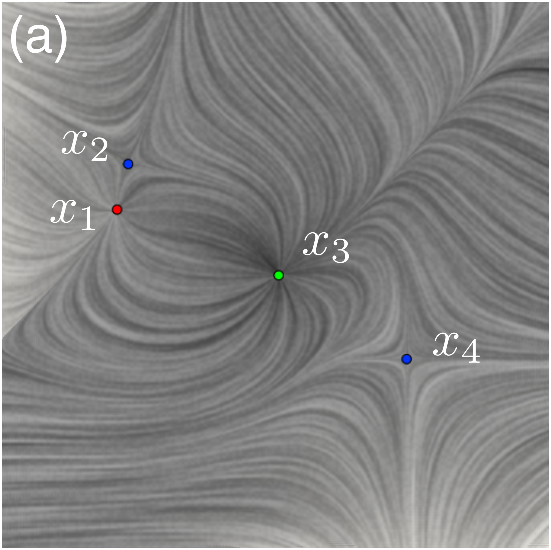
Contributions: Robustness-based simplification

- Pruning critical points based on stability
- Encodes flow magnitude, naturally hierarchical, precise ordering of critical points cancellations, quantifies the amount of perturbation that is needed at each level.
- A complementary view to topological-skeleton-based simplification.
- Efficient computation for large data/when separatrices are difficult to integrate; handle general cases
- A novel simplification algorithm for zero-degree components. Handle more general boundary configurations w/o requirements on the Conley Index. Potential generalization to higher dimensions.

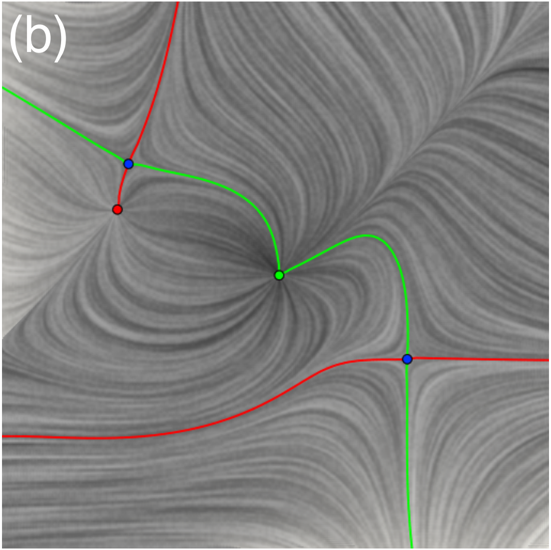
Robustness-based simplification in a nutshell

- Our method finds, in the space of all vector fields, the one that is closest to the original vector field with a particular set of critical points removed, according to a metric based on the L_∞ norm (the maximum point-wise modification to the vector field).
- Our results are optimal in this norm, that is, there exists no simplification with a smaller perturbation.

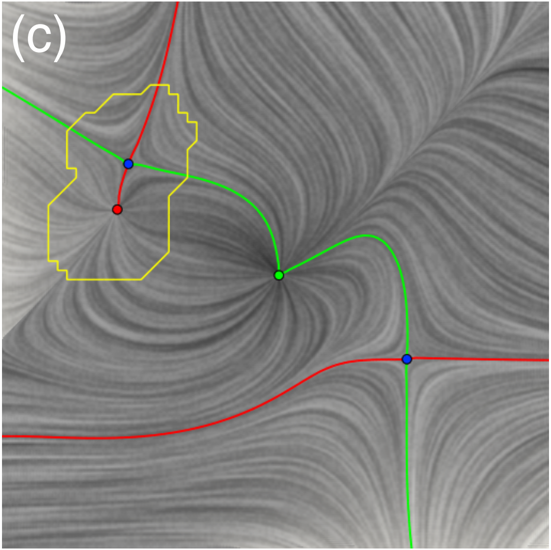
Some teaser results: synthetic A



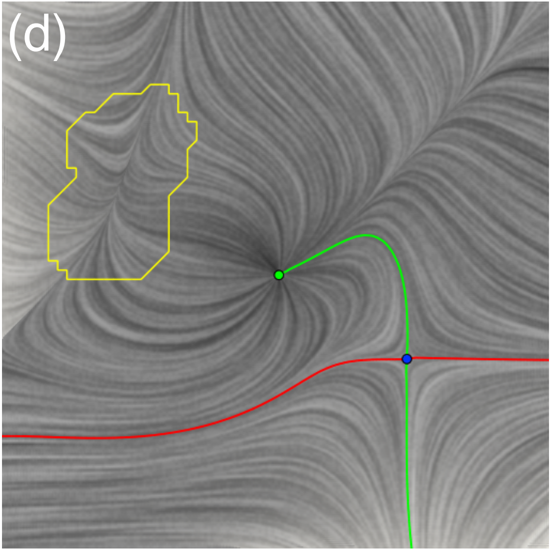
Some teaser results: synthetic A



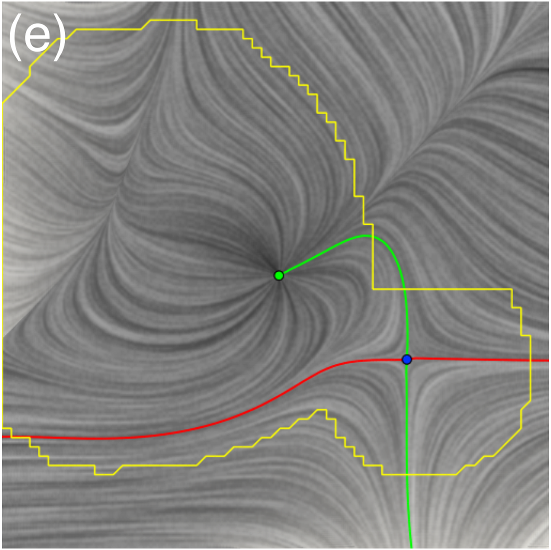
Some teaser results: synthetic A



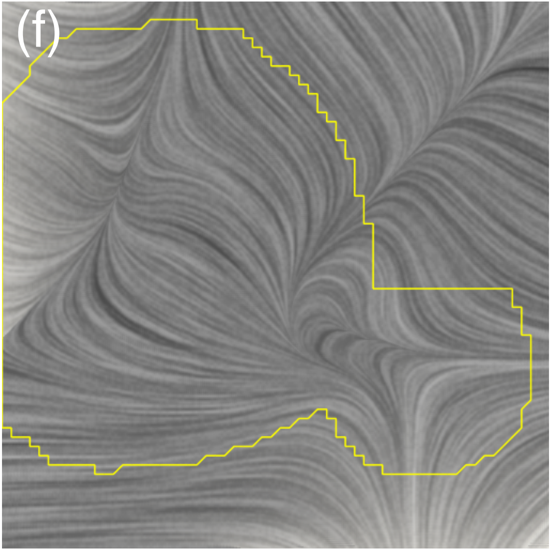
Some teaser results: synthetic A



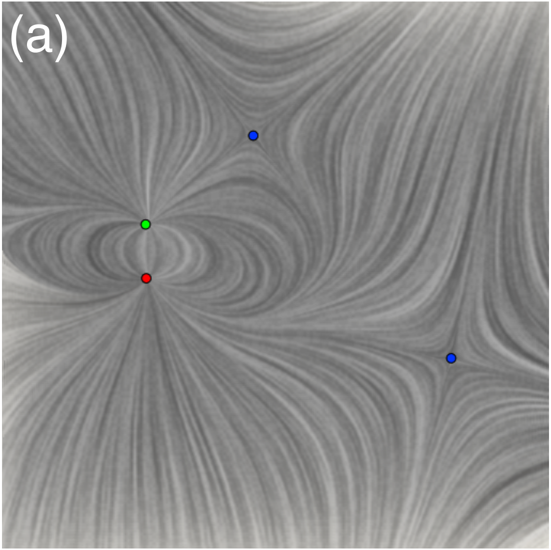
Some teaser results: synthetic A



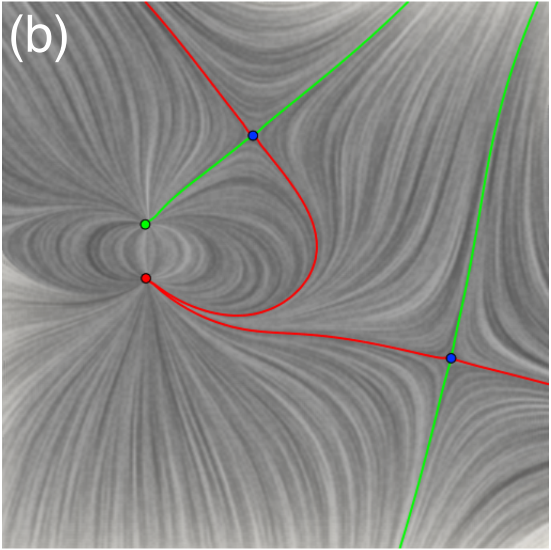
Some teaser results: synthetic A



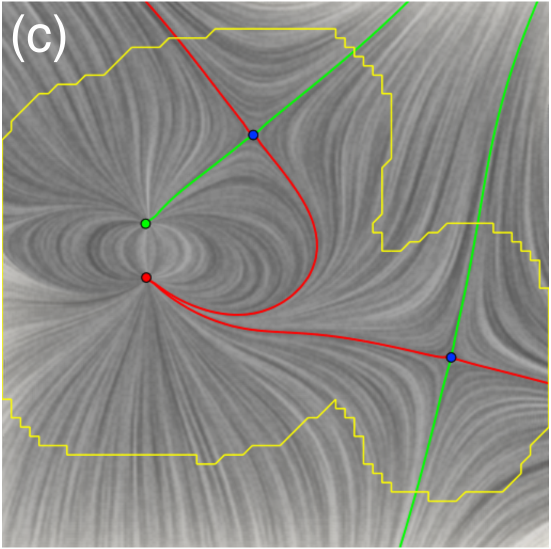
Some teaser results: synthetic B



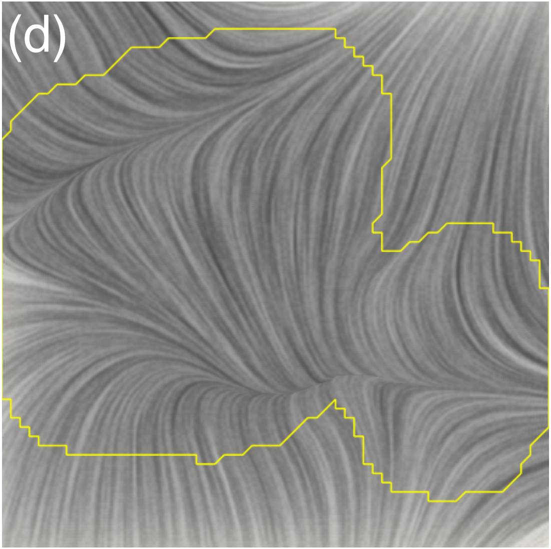
Some teaser results: synthetic B



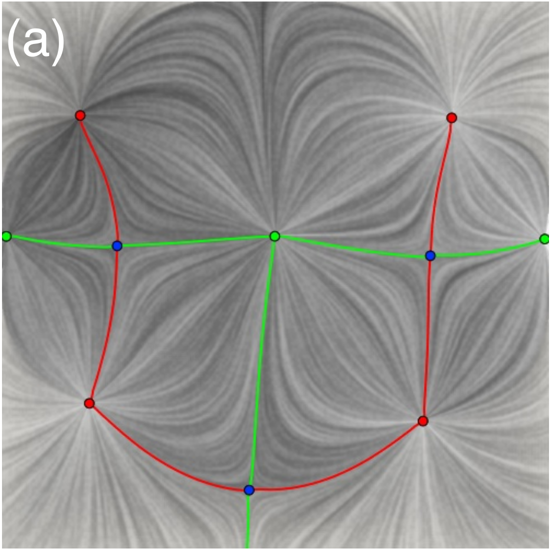
Some teaser results: synthetic B



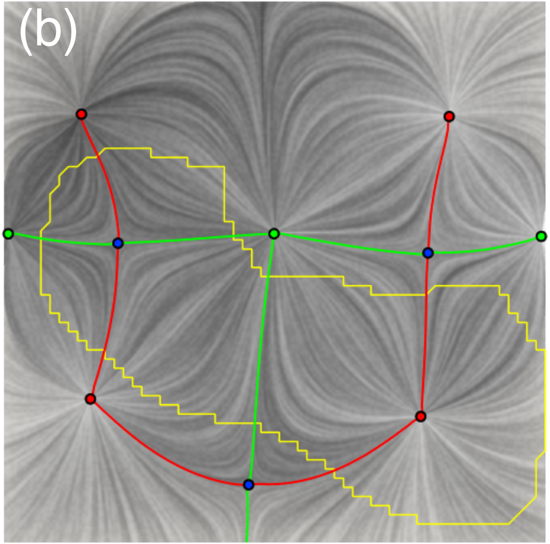
Some teaser results: synthetic B



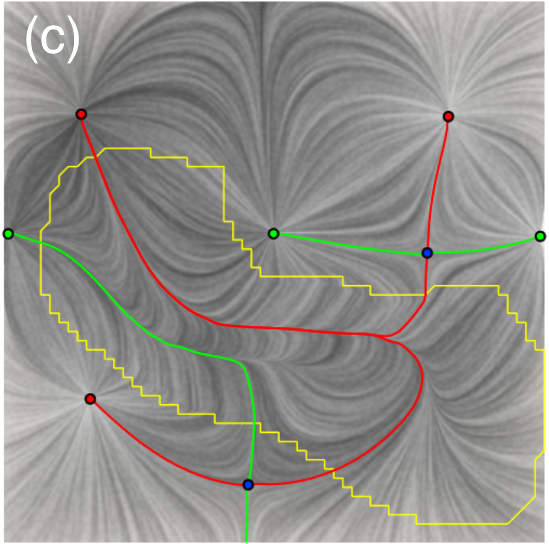
Some teaser results: synthetic C



Some teaser results: synthetic C



Some teaser results: synthetic C



Robustness-Based Simplification Algorithm

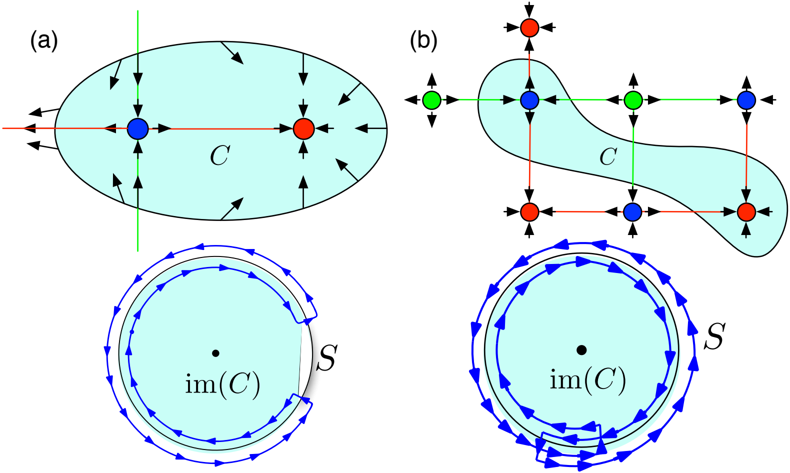
Image space $\mathrm{im}\,(C)$ of a zero-degree component $C \subseteq \mathbb{F}_r$

$f : C \to \mathbb{R}^2$, $\mathrm{im}\,(C) \subset \mathbb{R}^2$ is constructed by mapping each $p \in C$ to its vector coordinates $v_p = f(p)$.

Origin in $\mathrm{im}\,(C)$: critical points (0 vectors) in C.

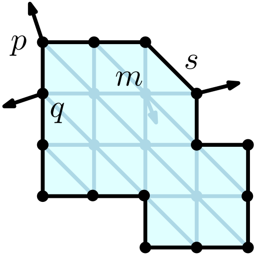
Since $\forall p \in C$, $||v_p||_2 \leq r$, $\mathrm{im}\,(C) \subset$ a disc of radius r (with boundary S).

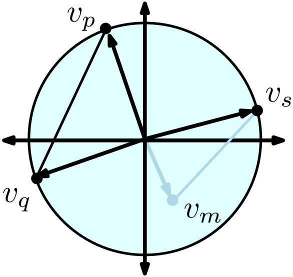
S is uncovered if $\mathrm{im}\,(\partial C) \subset S$, o.w. covered.



PL Image space

$f : K \to \mathbb{R}^2$, K is a triangulation of C
Linear interpolation: edges and triangles in K map to edges and triangles in $\mathrm{im}\,(C)$.





Our simplification algorithm consists of four operations

- $\mathbf{Smoothing}(C)$: Perform Laplacian smoothing on C
- $\mathbf{Cut}(C)$: Deform the vector field in its image space $\mathrm{im}\,(C)$ to remove critical points in C
- $\mathbf{Unwrap}(C)$: Modify the vector field in its image space $\mathrm{im}\,(C)$ so part of its boundary is uncovered
- $\mathbf{Restore}(C)$: Set the boundary to its original value

The Algorithm

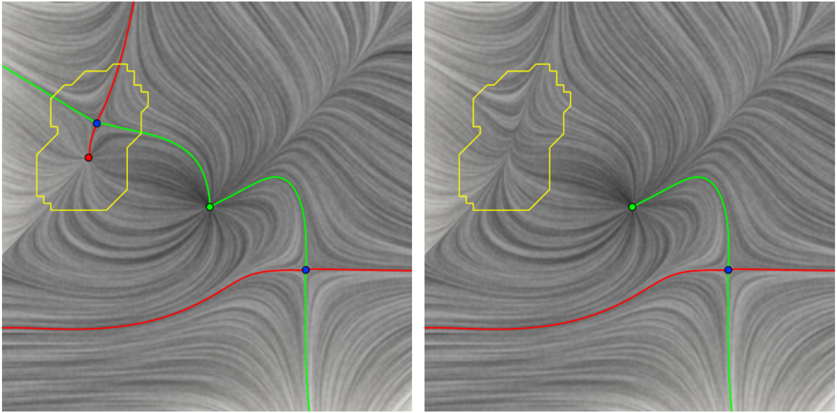
$\mathrm{CH}_*(C)$: Conley index.

- **Case (a)**: If $\mathrm{CH}_*(C)$ is trivial, return $C_1 = \textbf{Smoothing}(C)$.
- **Case (b)**: If $\mathrm{CH}_*(C)$ is non-trivial and the boundary of $\mathrm{im}\,(C)$ is uncovered, then $C_1 = \textbf{Cut}(C)$, and return $C_2 = \textbf{Smoothing}(C_1)$.
- **Case (c)**: If $\mathrm{CH}_*(C)$ is non-trivial and the boundary of $\mathrm{im}\,(C)$ is covered, then $C_1 = \textbf{Unwrap}(C)$, $C_2 = \textbf{Cut}(C_1)$, $C_3 = \textbf{Restore}(C_2)$ and return $C_4 = \textbf{Smoothing}(C_3)$.

(Laplacian) Smoothing

Given a vector field f and an isolating neighborhood C, a modified vector field $\overline{f}$ inside C can be found by solving a constrained optimization problem.

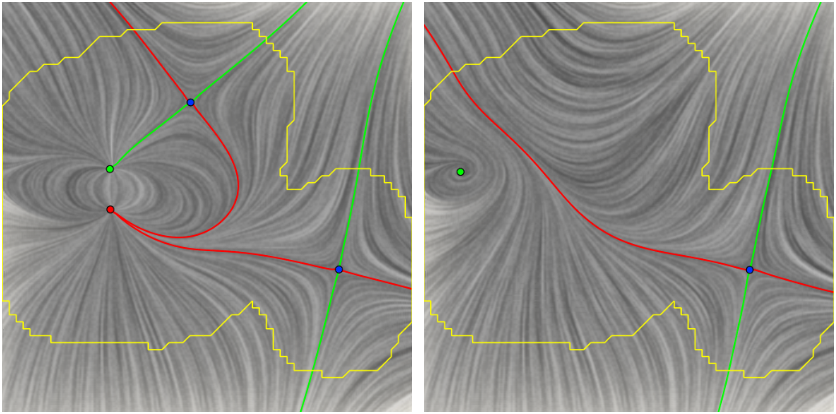
$$\overline{f}(v_i) = \sum_j \omega_{ij} \overline{f}(v_j)$$



(Laplacian) Smoothing

Given a vector field f and an isolating neighborhood C, a modified vector field $\overline{f}$ inside C can be found by solving a constrained optimization problem.

$$\overline{f}(v_i) = \sum_j \omega_{ij} \overline{f}(v_j)$$

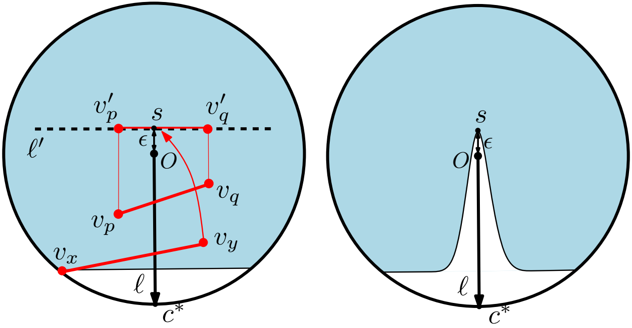


- No guarantee of a critical point free field!

Cut

Deform $\mathrm{im}\,(C)$ s.t. there is a small nbhd surrounding the origin that is not covered by $\mathrm{im}\,(C)$ (i.e. there is no critical point in C after the deformation).

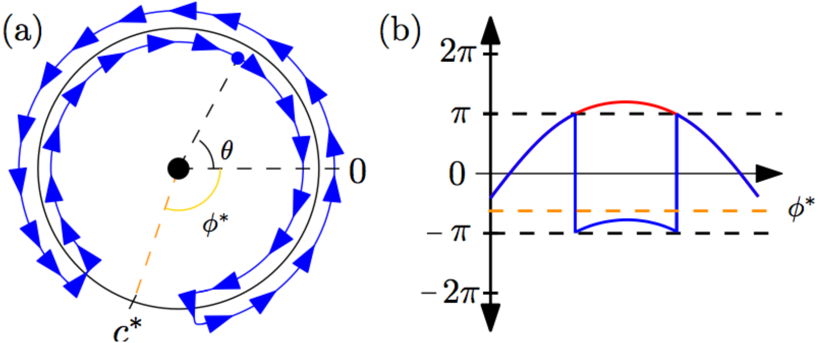
c^*: cut point



By construction: amount of perturbation $< r + \epsilon$

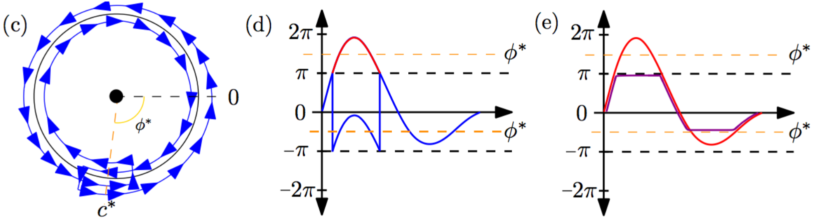
Cut: finding the cut point

Via phase plot (angle-valued function)



Unwrap

c^*: unwrap point

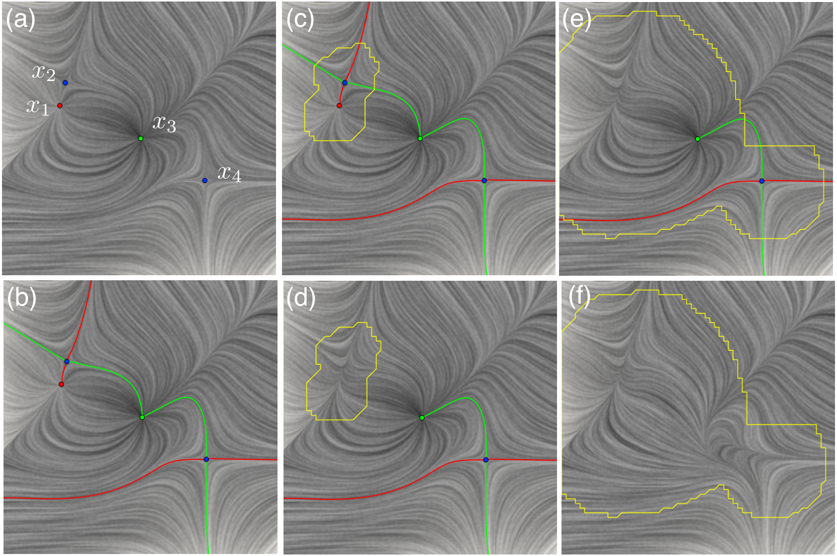


By construction: amount of perturbation $< r + \epsilon$

Restore

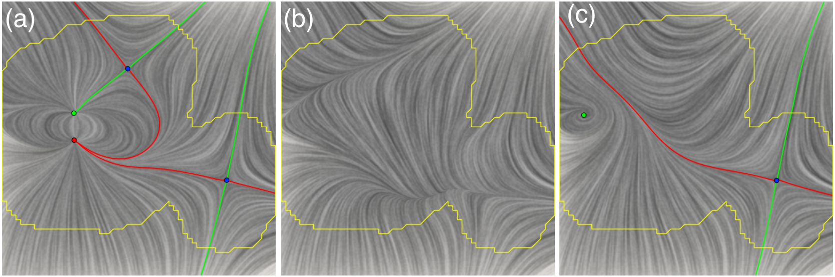
Restore the boundary to its original values, which again covers the boundary but keeps the origin uncovered.
Perturbation $< r + \epsilon$: internal nodes move by less than $r + \epsilon$, while the boundary has the original values.

Example revisited: Synthetic A trivial Conley index



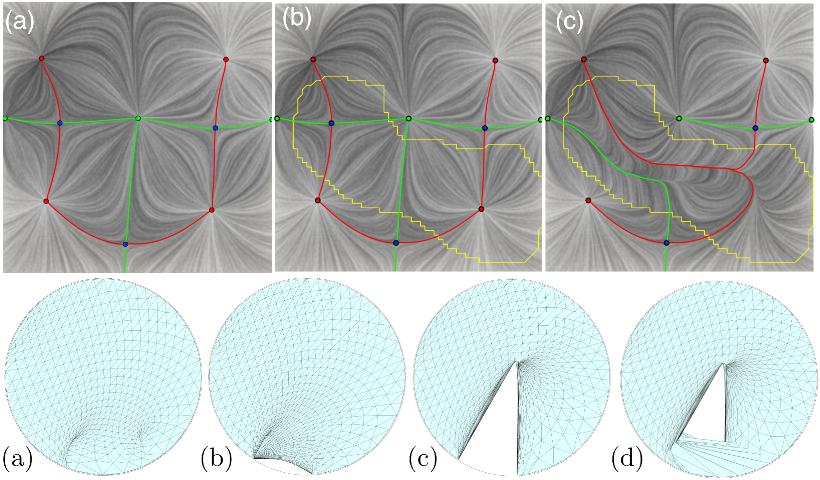
(a)-(b) original, (c)-(d) (e)-(f) 1st and 2nd level simplification after **Smoothing**

Example revisited: Synthetic B non-trivial Conley index



(a) original, (b) after **Cut** and **Smoothing**, (c) only **Smoothing** does not give critical point free field

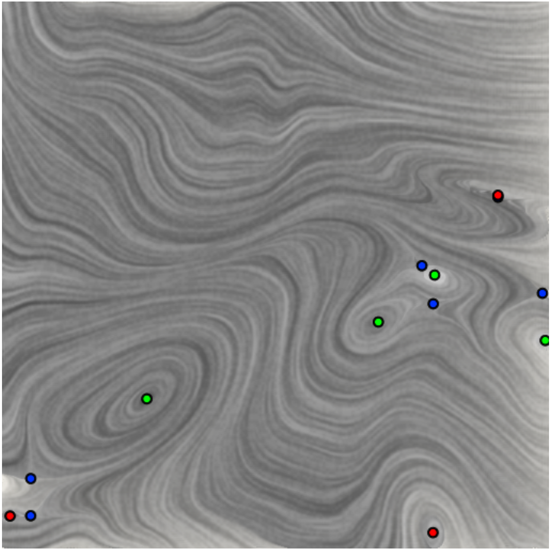
Example revisited: Synthetic C



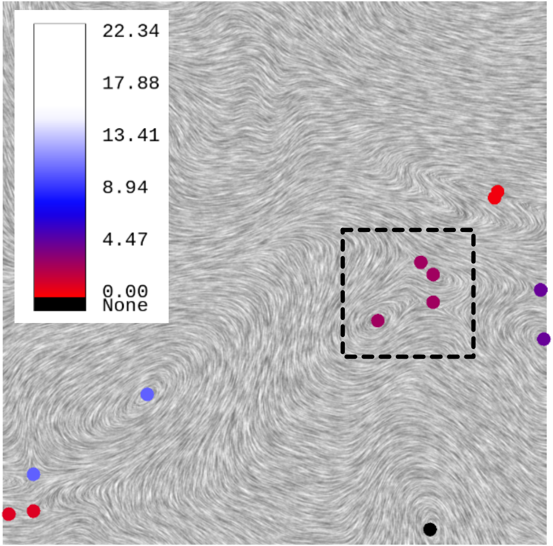
(a) original, (b) after **Unwrap**, (c) after **Cut** and (d) final output after **Restore**

Results

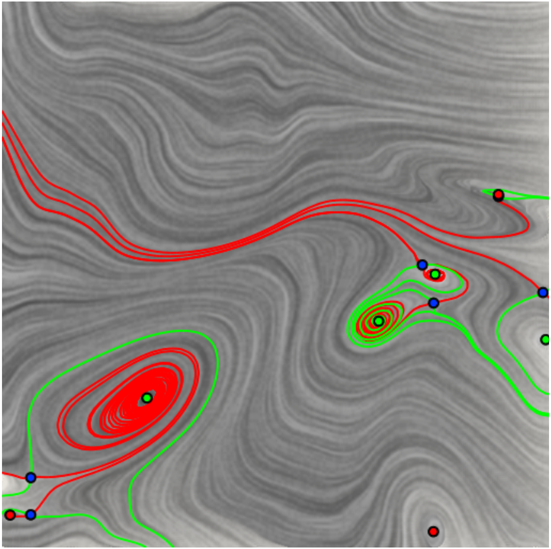
Ocean Eddie dataset A



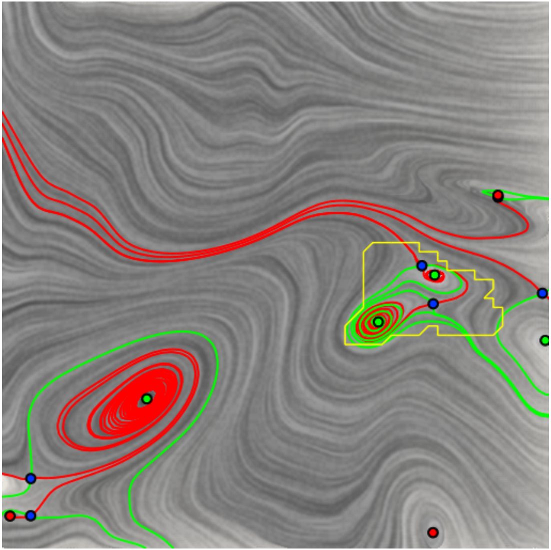
Ocean Eddie dataset A



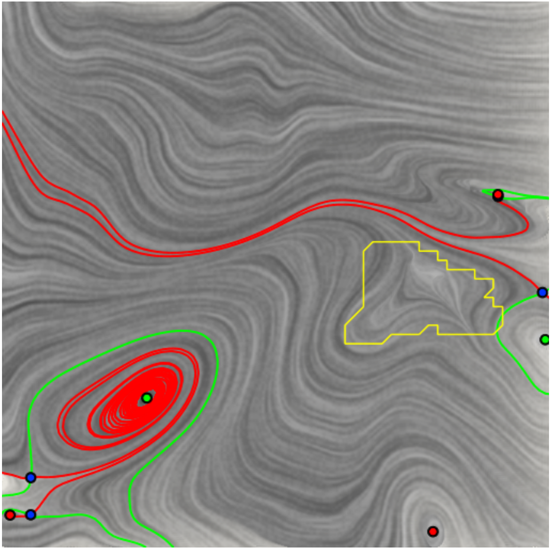
Ocean Eddie dataset A



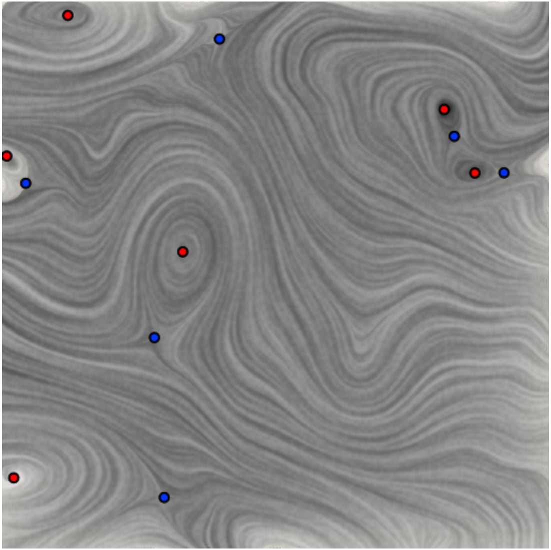
Ocean Eddie dataset A



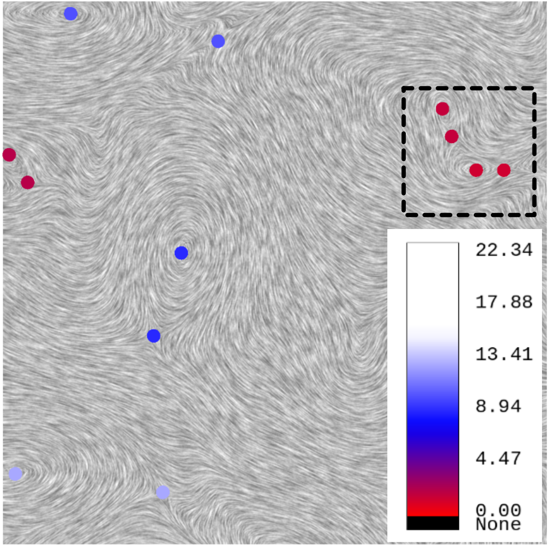
Ocean Eddie dataset A



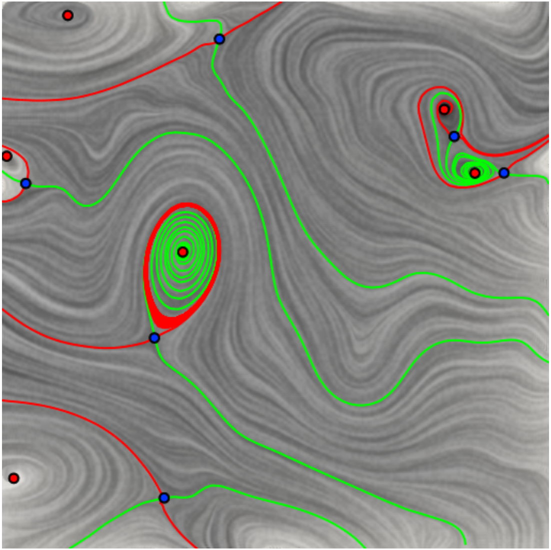
Ocean Eddie dataset B



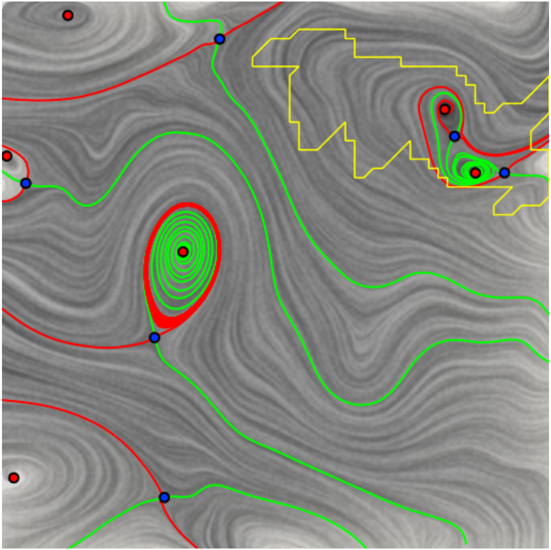
Ocean Eddie dataset B



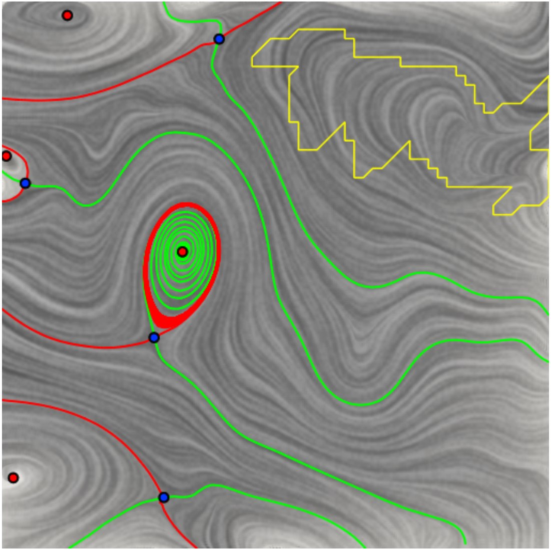
Ocean Eddie dataset B



Ocean Eddie dataset B

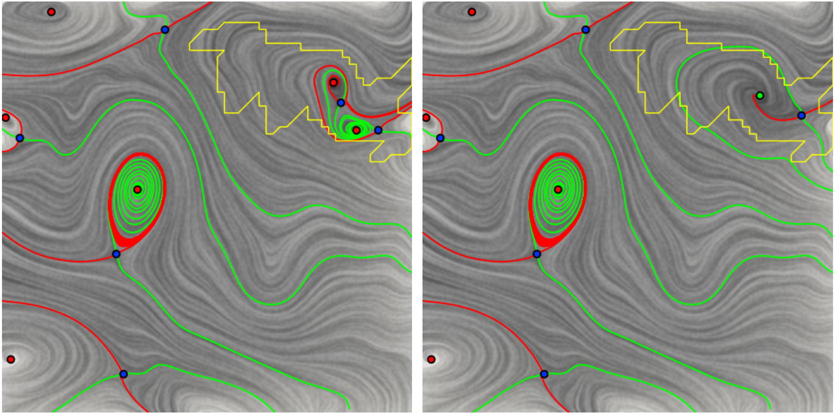


Ocean Eddie dataset B

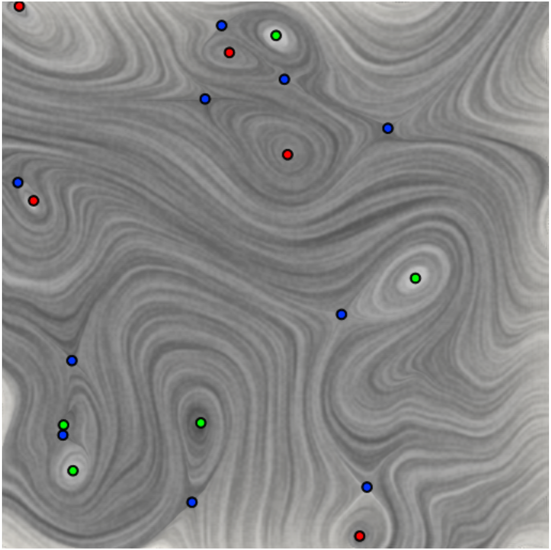


Ocean Eddie dataset B: Challenging Scenarios

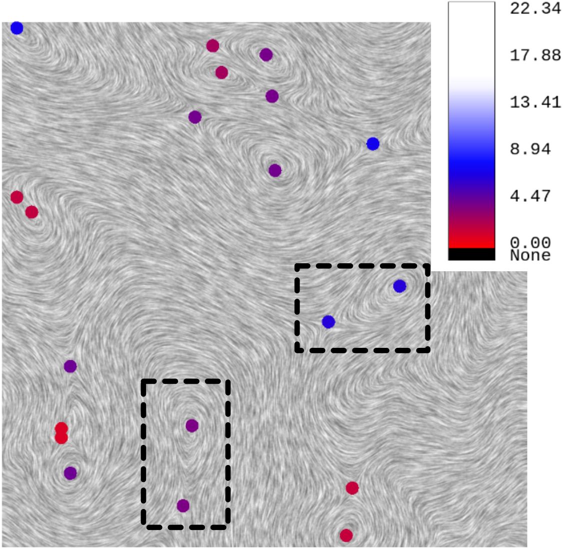
A region with non-trivial Conley index and uncovered boundary, where direct smoothing does not remove its critical points.



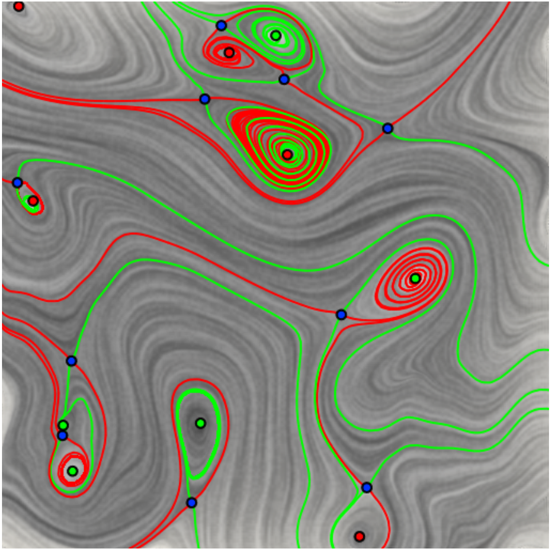
Ocean Eddie dataset C



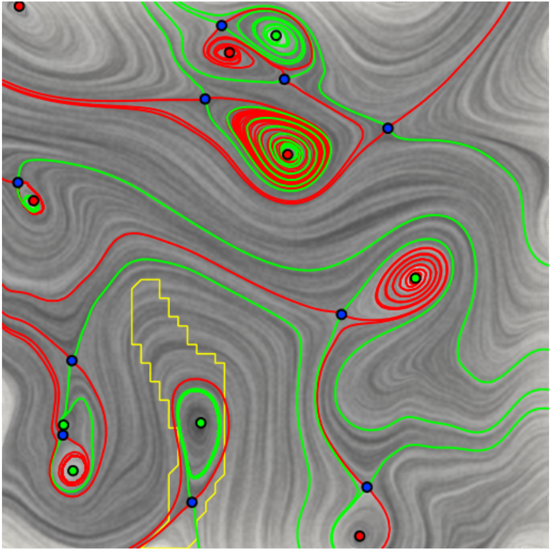
Ocean Eddie dataset C



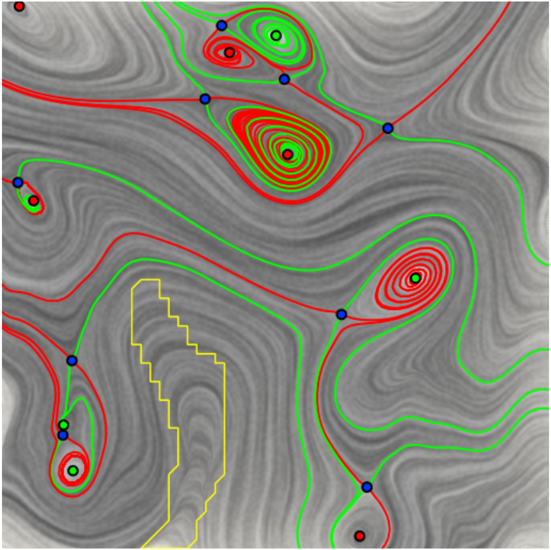
Ocean Eddie dataset C



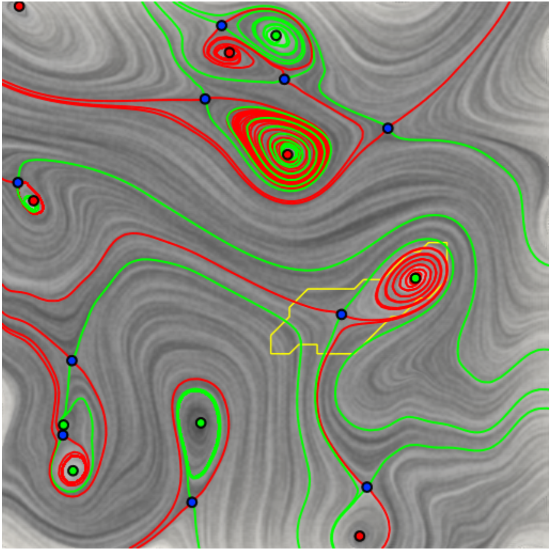
Ocean Eddie dataset C



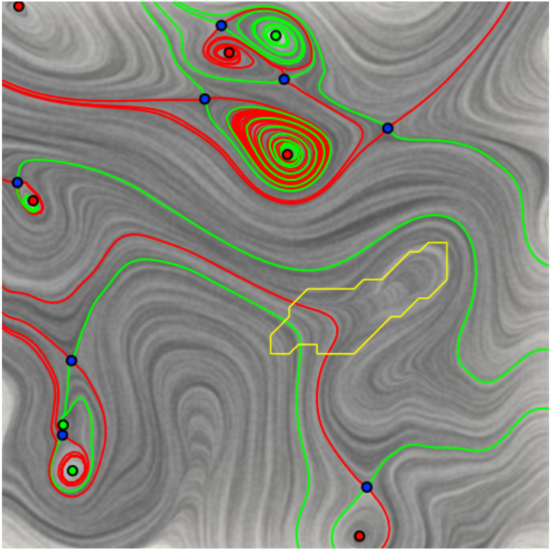
Ocean Eddie dataset C



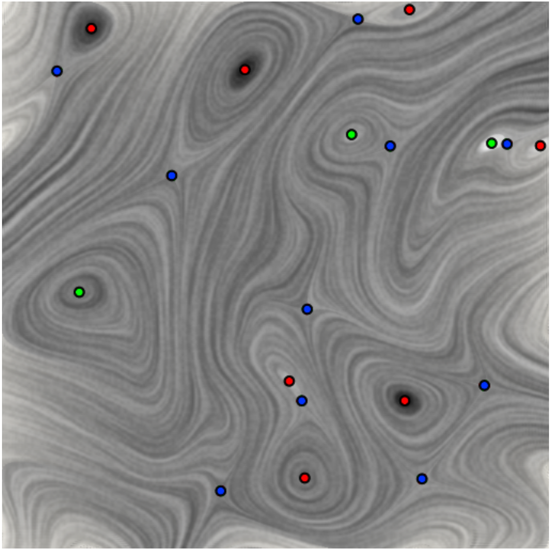
Ocean Eddie dataset C



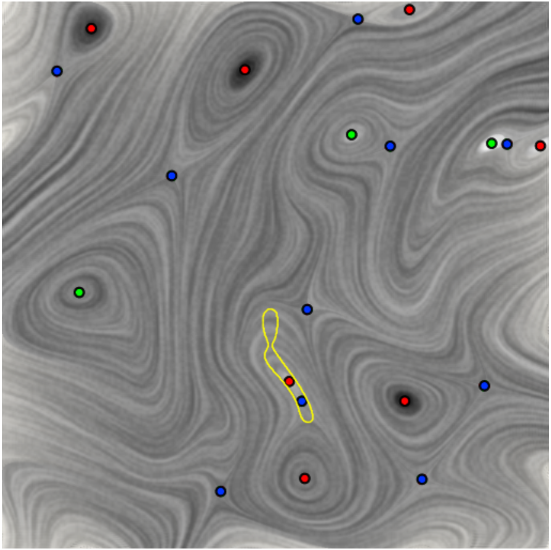
Ocean Eddie dataset C



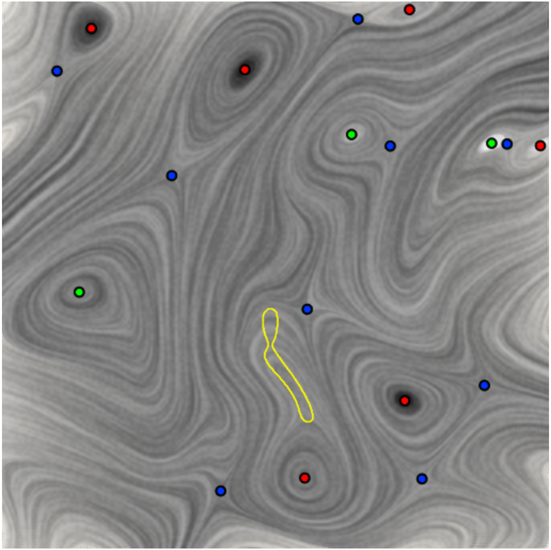
Combustion dataset



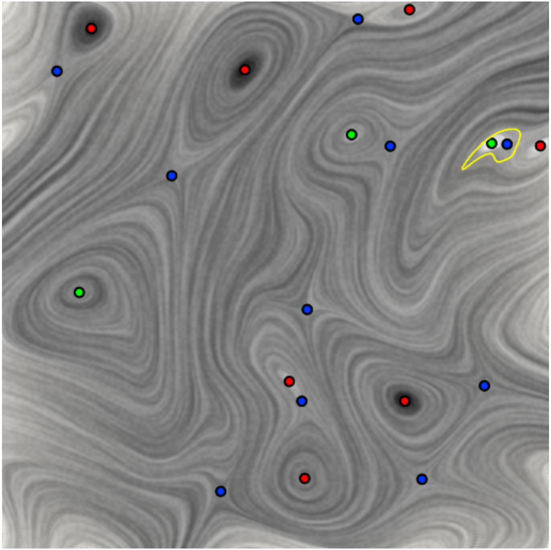
Combustion dataset



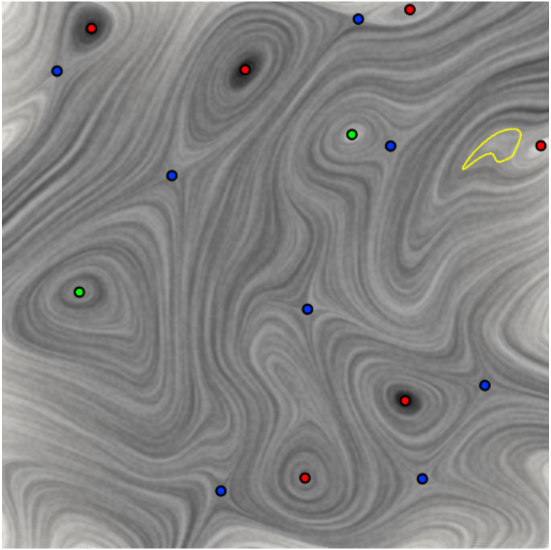
Combustion dataset



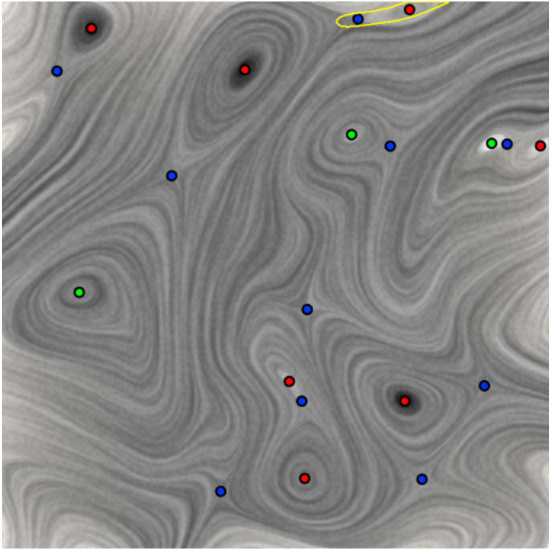
Combustion dataset



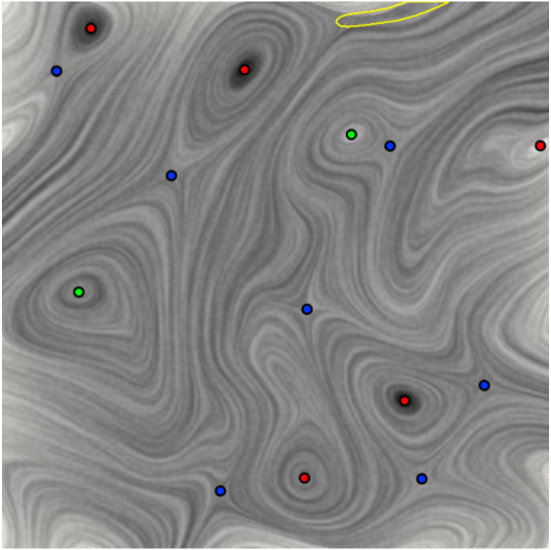
Combustion dataset



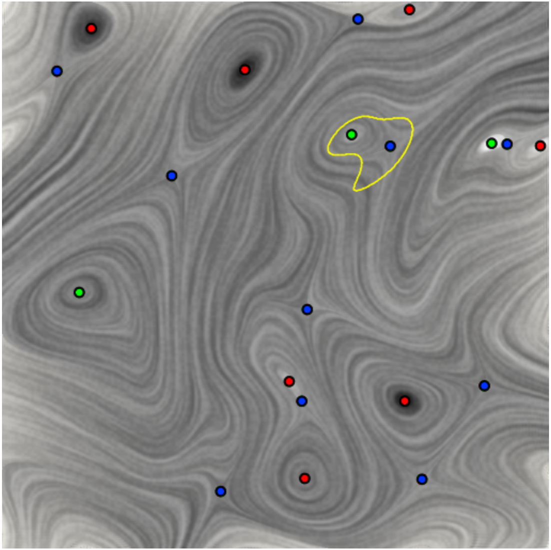
Combustion dataset



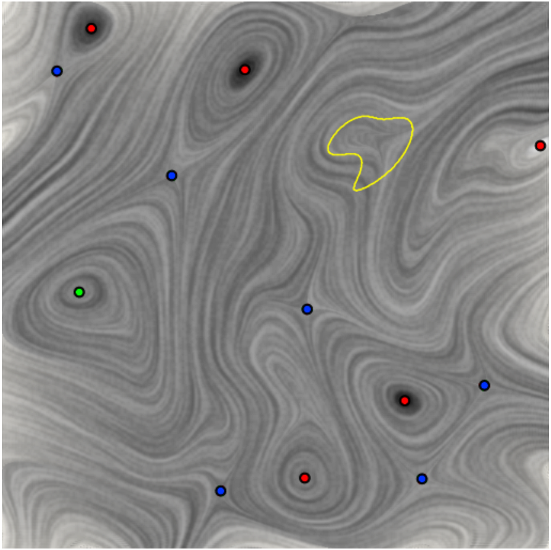
Combustion dataset



Combustion dataset



Combustion dataset



Combustion dataset discussed

- Incompressible flow and has high resolution.
- Topological-skeleton-based method: potentially difficult, as separatrices either do not exist or require many integration steps before reaching the sink/source-like critical points (due to numerical instabilities). This makes the pairing of the critical points challenging.
- One standard solution in 2D: compute topological-skeleton of the dual VF(rotation by 90°). Not straight forward in 3D and computationally expensive for large data.
- Robustness-based method does not require the computation of topology, and its sublevel set computation is fast and can be augmented using parallel computation. Maybe more practical for processing large datasets.

Discussions

- Scalability: large datasets; merge tree (near linear time) encodes all information needed for simplification; cut/unwrap point detection occur on the boundary
- Generality: require only degree-zero component; apply to highly rotational data (e.g. centers) as well as cases where critical points are not connected by separatrices
- Other metrics for merge tree / robustness computation
- Time-varying: consistency simplification across time-slices
- 3D challenge: few existing simplification techniques; finding an uncovered boundary point and computing an unwrapping

Part 1-C: Robustness

Interpreting Feature Tracking through the Lens of Robustness

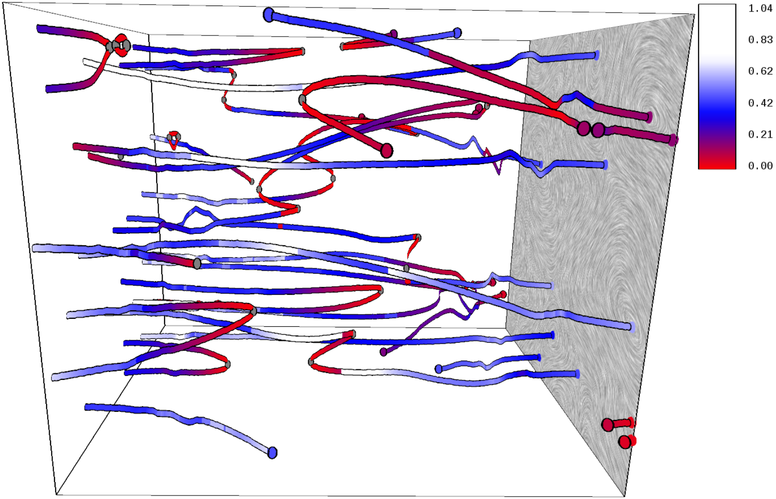
Joint work: Primoz Skraba

[Skraba, Wang 2013]

Feature tracking from vector field data

Feature tracking: resolve correspondences between features in successive time steps and to analyze the dynamic behaviors of such features.

Commonly tracked features: volume, areas, contours, boundaries, vortices, shock waves, critical points and sublevel sets...



Feature tracking from vector field data: an incomplete view

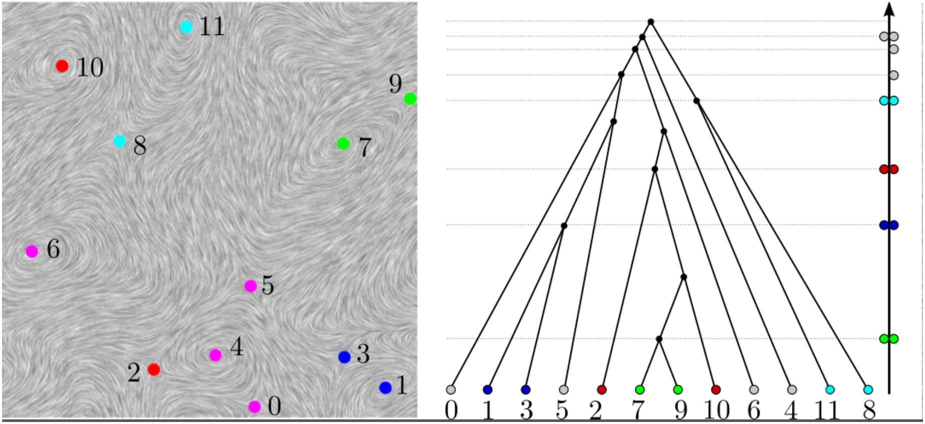
- Feature extractions at individual time slices and feature matching via region/attribute correspondence [Post et al. 03] correspondence based on distance proximity / spatial overlap [Samtaney et al. 94] [Silver et al. 99].
- Temporal interpolation, time as the 4th dim, iso-surface extraction in 4D space-time [Weigle et al. 98] vortex tracking [Bauer et al. 02]. For for critical points, Reeb graphs [Edelsbrunner et al. 02] & Jacobi sets; 2D/3D linear interpolation [Tricoche et al. 02][Garth et al. 04].
- Feature flow fields, integrated persistence [Theisel el al. 03] [King et al. 08]] [Reininghaus et al. 11] [Weinkauf et al. 11]

Take home message

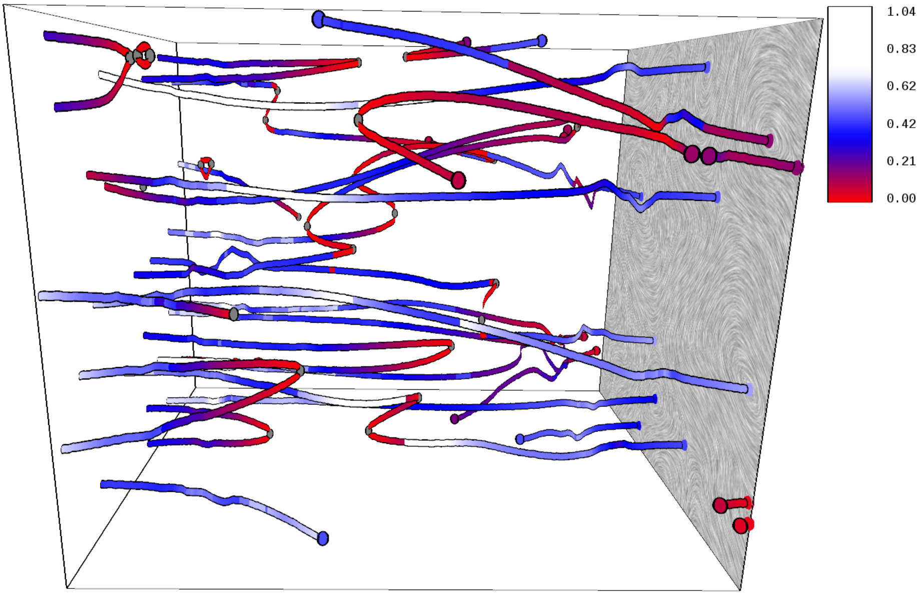
- Fresh interpretation of the notion of critical point tracking, through the lens of robustness. (Cute :-)
- Relate critical points tracking with their stability: stable critical points could be tracked more easily and more accurately.
- Robustness help us understand the sufficient sampling conditions under which we can resolve correspondences based on region overlap.
- Theoretical basis for visualizing PL critical paths.

Note: results in $\mathbb{R}^n$, use $\mathbb{R}^2$ for illustration/explanations.

Robustness: combustion dataset



Robustness Vis Revisited



An Illustrative Guide to Main Results: Correspondence Under Robustness Setting

Assumptions

Time-varying vector field is c-Lipschitz.
We have an ϵ-sample in space and time.

$f : \mathbb{X} \times \mathbb{R} \to \mathbb{R}^2$, $\mathbb{X} \subseteq \mathbb{R}^2$, $f_t(x) = f(x, t) : \mathbb{X} \to \mathbb{R}^2$ for $t \in \mathbb{R}$.

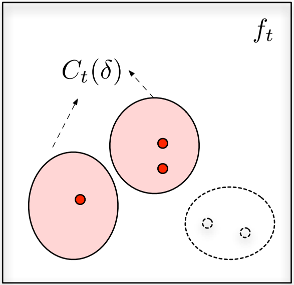
$$||f_t(x) - f_{t+\epsilon}(x)||_2 \leq c\epsilon, \quad \forall x \in \mathbb{X}$$

Better error bound for approximating the underlying vector fields (function), better interpolation, and better constants in the theoretical guarantees.

Definition: Sublevel set of Euclidean norm of a time slice

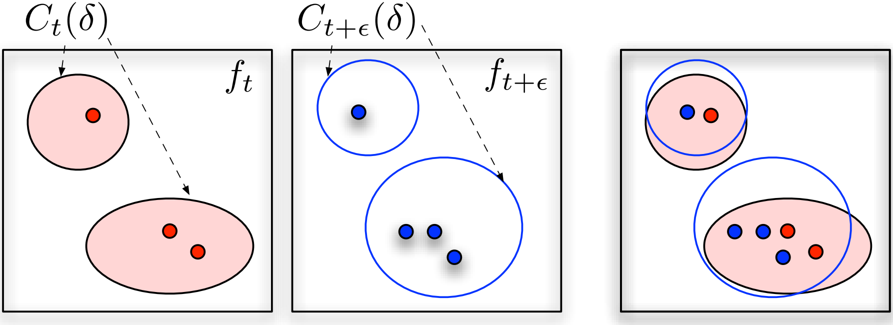
Sublevel set of $||f_t(x)||_2$ for any $\delta > 0$ whose degree is non-zero:

$$C_t(\delta) = \{x \in \mathbb{X} \mid ||f_t(x)||_2 \leq \delta\}$$



Lemma 1: Critical Points Containment

For two adjacent time steps of the vector field $f_t, f_{t+\epsilon} : \mathbb{X} \rightarrow \mathbb{R}^2$, the critical points in both time steps belong to $\operatorname{int}\left(C_t(\delta) \cap C_{t+\epsilon}(\delta)\right)$ for all $\delta > c\epsilon$.

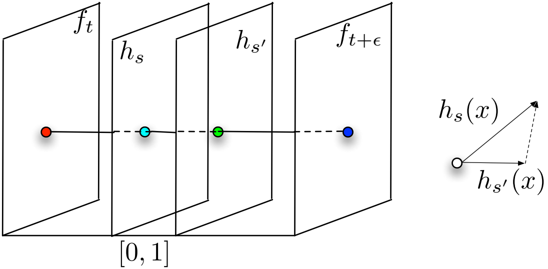


Intuition: Critical points are contained in intersections across time steps, although this does not imply correspondence.

Definition: δ-bounded homotopy between f_t and f_ϵ

$H : \mathbb{X} \times [0,1] \to \mathbb{R}^2$, s.t $\forall x \in \mathbb{X}$, $H(x,0) = f_t(x)$, $H(x,1) = f_{t+\epsilon}(x)$.
Intermediate time slices: $h_s(x) = H(x,s) : \mathbb{X} \to \mathbb{R}^2$ $(s \in [0,1])$,
based on straight-line homotopy.
H is δ-bounded, if $\forall x \in \mathbb{X}$ and $\forall s, s' \in [0,1]$,

$$||h_s(x) - h_{s'}(x)||_2 \leq \delta$$



Definition: δ-correspondence

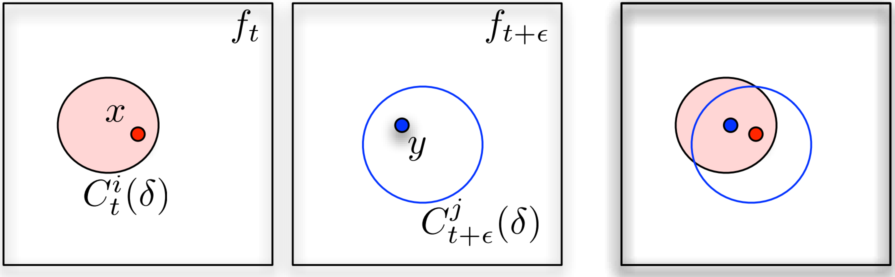
A δ-correspondence between a pair of critical points is defined s.t. there exists a δ-bounded homotopy which maps the points to each other.

There is a δ-correspondence between critical points x at t and y at $t + \epsilon$ if there exists a δ-bounded homotopy H between f_t and $f_{t+\epsilon}$, s.t. $H^{-1}(0)$ contains a continuous path embedded in $\mathbb{X} \times [0, 1] \subset \mathbb{R}^3$ that connects x with y.

We refer to such a path as the critical path.

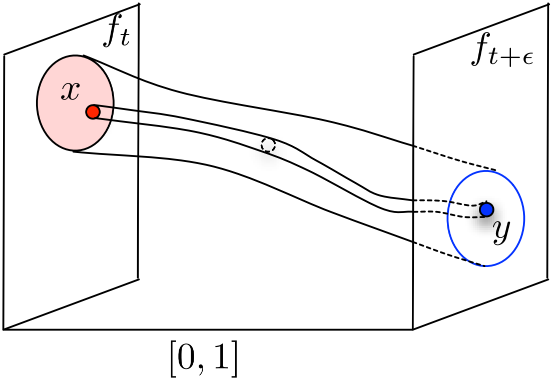
Lemma 2: Critical Points Correspondence Under Unique Intersection

For $\delta > c\epsilon$, let $C_t^i(\delta)$ and $C_{t+\epsilon}^j(\delta)$ be components of the δ-sublevel sets with a unique intersection. If there exists a unique δ-robust critical point in each component, then they are in correspondence.



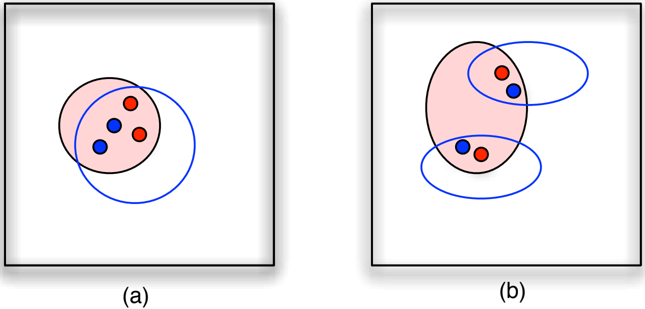
Lemma 2 Cont.

Main idea: Construct a $(\delta + c\epsilon)$-bounded homotopy that maps x to y. Consider tubular neighborhood surrounding the parametrized curve connecting x and y



Lemma 3: Robust Critical Points Correspondence

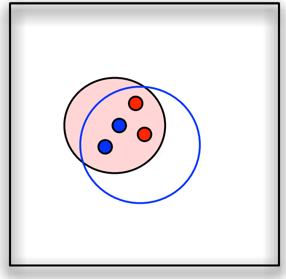
There exists a $(\delta + c\epsilon)$-bounded homotopy between δ-robust critical points between time slices, from which a correspondence could be obtained.



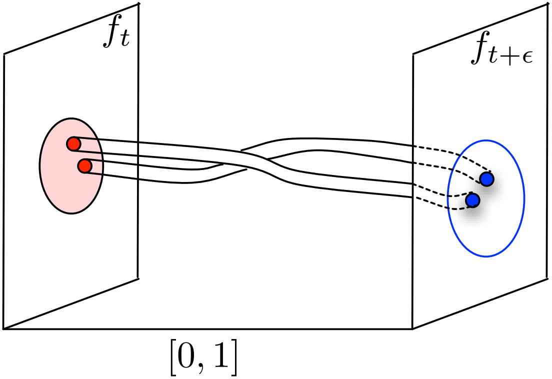
Main idea: Consider (a) with unique intersection condition, however there may be multiple δ-robust critical points in each component; (b) without unique intersection condition.

Lemma 3 Cont.

Case (a): choose the desired correspondence by constructing critical paths (which are no longer unique).



(a)

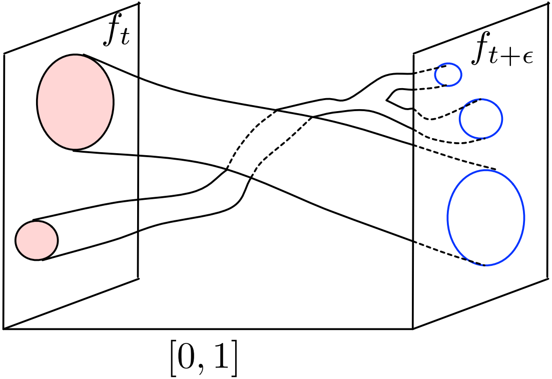


Case (b): Require a path exists between critical points. Many choices of correspondences, homotopy construction works for any of them (under the restriction that the corresponding points are distinct).

Definition: δ-tube

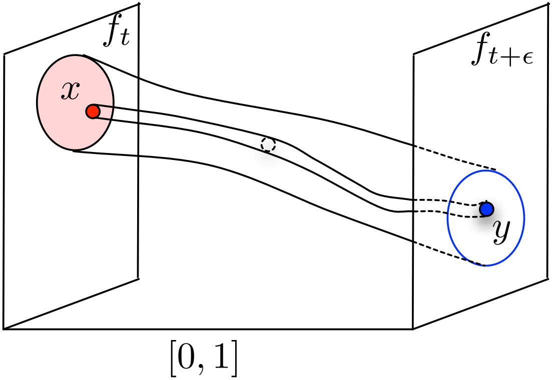
A δ-tube is defined as a component in $\bigcup_{s\in[0,1]} C_s(\delta)$.

(also defined for PL version)



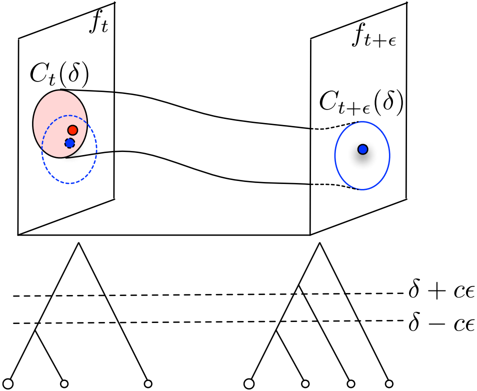
Lemma 4: Critical Paths Containment

For any $\delta > c\epsilon$, if a critical path between two δ-robust critical points exists, it will be completely contained within a δ-tube between the two time slices f_t and $f_{t+\epsilon}$.



Lemma 5: Sublevel Set Unique Correspondence

For $\delta > c\epsilon$, if there are no merge events in $[\delta - c\epsilon, \delta + c\epsilon]$ (within the merge trees) between times t and $t + \epsilon$, the correspondences between c.c. in $C_t(\delta)$ and $C_{t+\epsilon}(\delta)$ whose intersections contain critical points are unique.



Conclusion and Discussion

- Infer correspondence between critical points based on their closeness in stability, measured by robustness, instead of just distance proximities within the domain.
- Robustness framework gives correspondence a natural sense of scale: if we allow larger perturbations, more correspondences are possible.
- How to visualize possible correspondences?
- How to choose the best correspondence between a set of possible correspondences?
- Construct a δ-bounded homotopy with a controlled Lipschitz constant?

Part 2-A: Branching and Circular Structure in Data

Branching and Circular Structure Detection Joint work: Valerio

Pascucci, Brian Summa, Mikael Vejdemo-Johansson

[Wang, Summa, Pascucci, Bhatia and Vejdemo-Johansson 2011]

Background

Persistent cohomology and circular coordinates [de Silva, Morozov, Vejdemo-Johansson 2009]

Circular structure detection

- For topological spaces with the homotopy type of a cell complex:

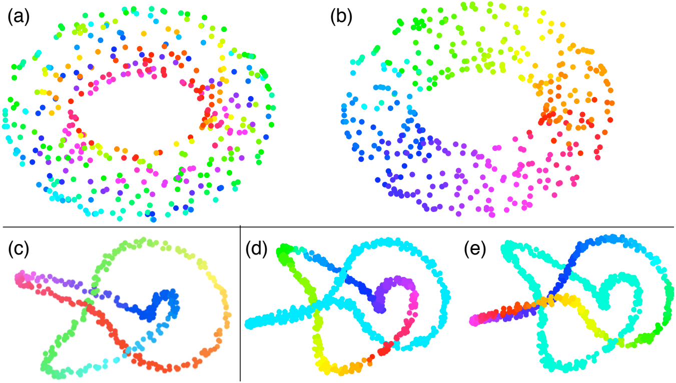
$$H^1(\mathbb{X};\mathbb{Z}) \cong [\mathbb{X},\mathbb{S}^1]$$

.

- If $\mathbb{X}$ has a non-trivial 1-dimensional cohomology class $[\alpha] \in H^1(\mathbb{X};\mathbb{Z})$, we can construct a continuous function $\theta : X \to \mathbb{S}^1$ from a representative α.

Circular structure detection

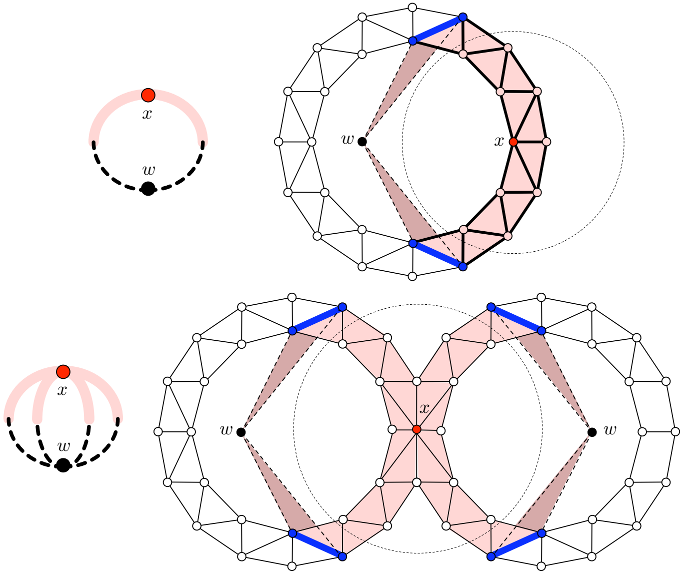
- Describe a topological procedure that enlarges the class of coordinate functions for dimensionality reduction to include global circle-valued functions, mapping the point cloud to a closed circle.



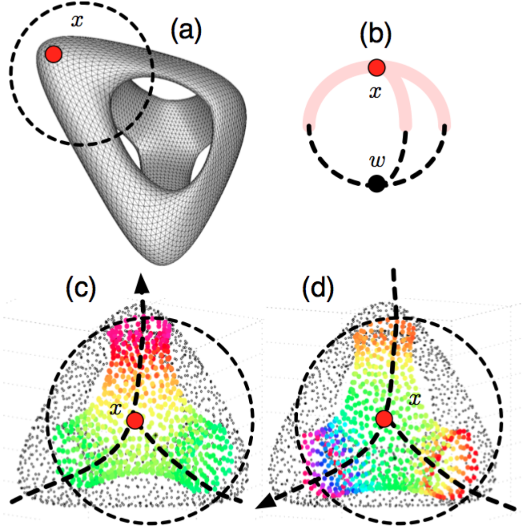
Courtesy [de Silva, Morozov, Vejdemo-Johansson 2009]

Branching Structure Detection

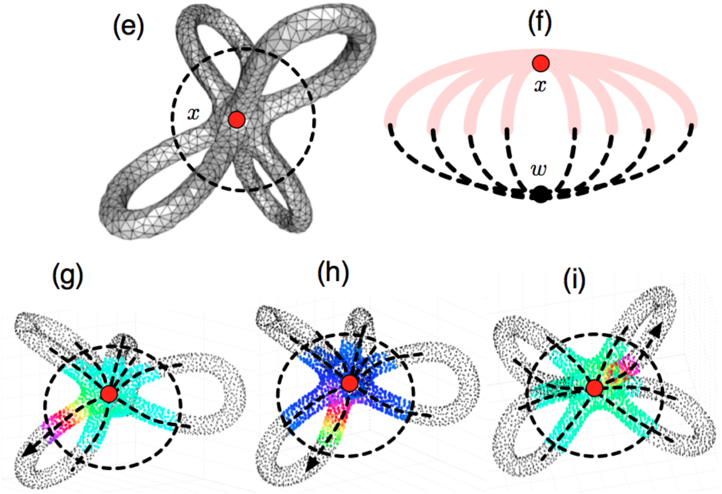
Local version of persistent cohomology



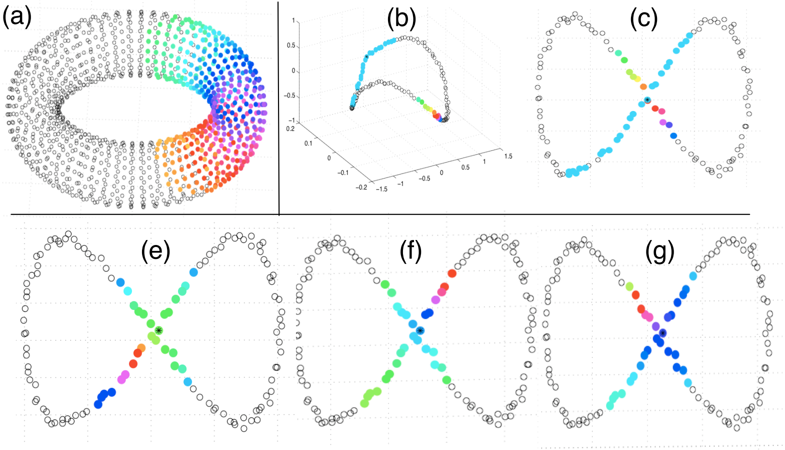
Local parametrization: branching detection



Local parametrization: branching detection

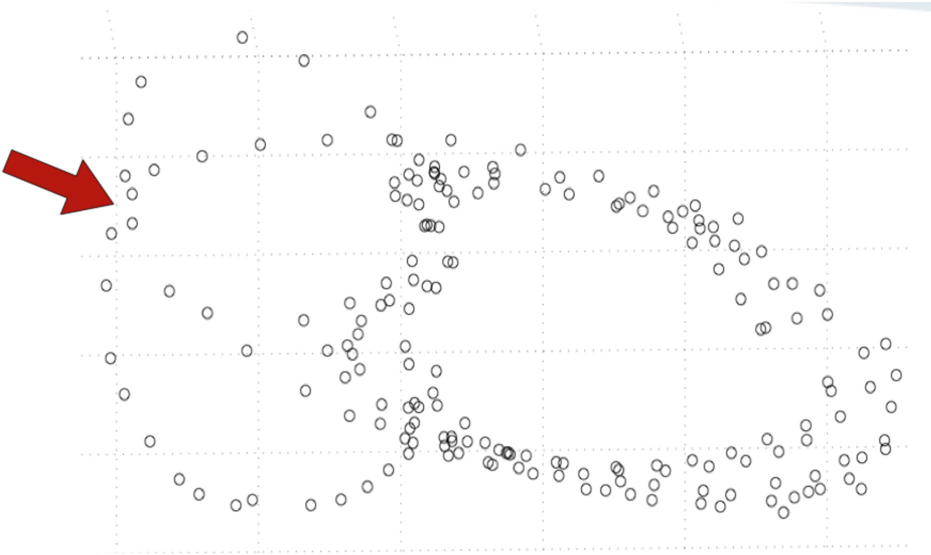


Local parametrization: fake branching detection



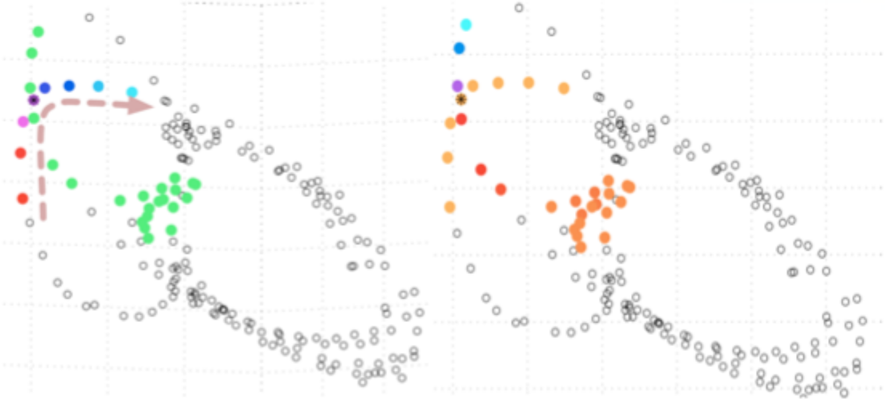
An Example

Motion capture data: Walk/Hop/Walk



[Video]

Motion capture data: Walk/Hop/Walk



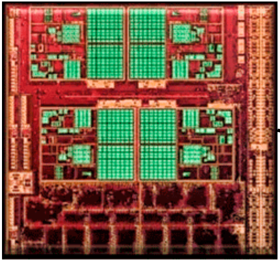
Part 2-B: Branching and Circular Structure in Data

Circular Structures in Memory Reference Traces

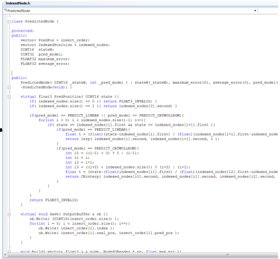
Joint work: Paul Rosen, Valerio Pascucci, A.N.M. Imroz Choudhury

[Choudhury, Wang, Rosen, Valerio 2012]

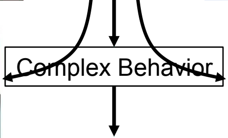
Overview



Phenom II CPU combined with DirectX 11 GPU



```
class PredictedNode {

protected:
public:
    vector< PredPos > insert_order;
    vector< IndexedPosition > indexed_nodes;
    UINT16  stateN;
    UINT32  pred_model;
    FLOAT32 maximum_error;
    FLOAT32 average_error;


public:
    PredictedNode( UINT16 _stateN, int _pred_model ) : stateN(_stateN), maximum_error(0), average_error(0), pred_model(
    ~PredictedNode(void){ }

    virtual float3 PredPosition( UINT16 state ){
        if( indexed_nodes.size() == 0 ){ return FLOAT3_INVALID; }
        if( indexed_nodes.size() == 1 ){ return indexed_nodes[0].second; }

        if(pred_model == PREDICT_LINEAR || pred_model == PREDICT_CATMULLROM){
            for(int i = 0; i < indexed_nodes.size()-1; i++){
                if( state >= indexed_nodes[i].first && state <= indexed_nodes[i+1].first ){
                    if(pred_model == PREDICT_LINEAR){
                        float t = (float)(state-indexed_nodes[i].first) / (float)(indexed_nodes[i+1].first-indexed_node
                        return lerp( indexed_nodes[i].second, indexed_nodes[i+1].second, t );
                    }
                    if(pred_model == PREDICT_CATMULLROM){
                        int i0 = ((i-1) < 0) ? 0 : (i-1);
                        int i1 = i;
                        int i2 = i+1;
                        int i3 = ((i+2) < indexed_nodes.size()) ? (i+2) : (i+1);
                        float t = (state-(float)indexed_nodes[i1].first) / (float)(indexed_nodes[i2].first-indexed_node
                        return CRinterp( indexed_nodes[i0].second, indexed_nodes[i1].second, indexed_nodes[i2].second,
                    }
                }
            }
        }
        return FLOAT3_INVALID;
    }

    virtual void Save( OutputBuffer & ob ){
        ob.Write( (UINT16)insert_order.size() );
        for(int i = 0; i < insert_order.size(); i++){
            ob.Write( insert_order[i].index );
            ob.Write( insert_order[i].real_pos, insert_order[i].pred_pos );
        }
    }

    void Build( vector< float3 > & node, NodePOReader * po, float max_err ){
```



Complex Behavior

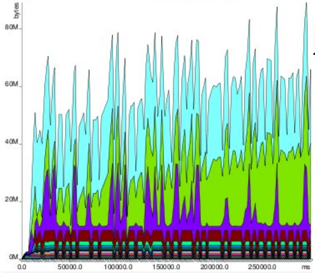


Image courtesy of http://wingolog.org

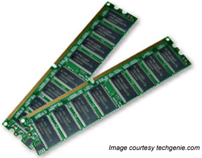
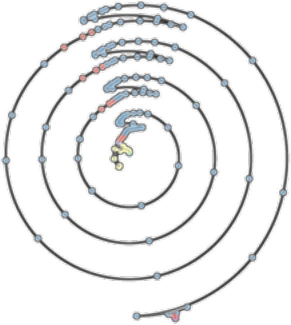
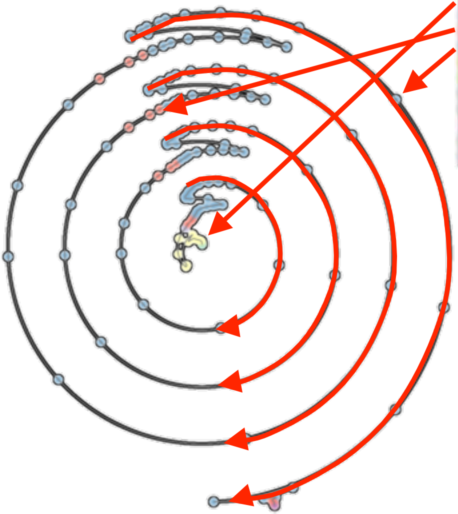
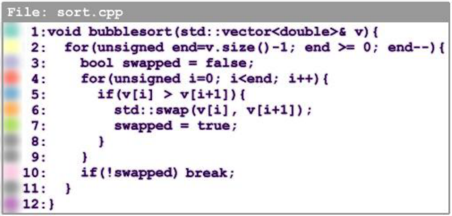


Image courtesy techgenie.com



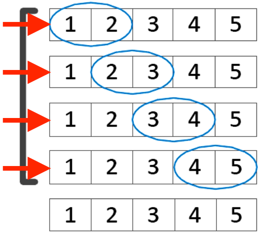
Example





File: sort.cpp

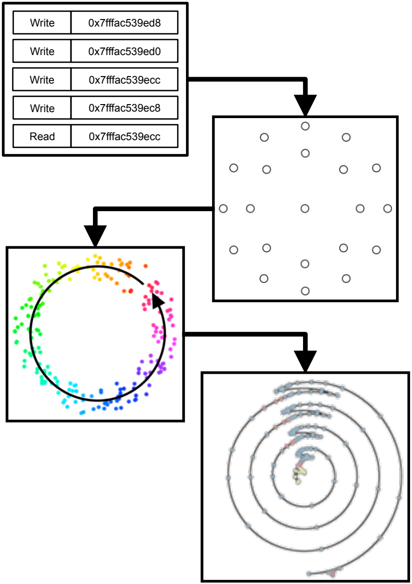
```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```



1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

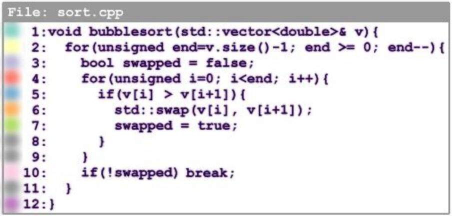
System pipeline

Exe. app. to generate memory reference trace → PCD → TDA → Visualization



Write	0x7fffac539ed8
Write	0x7fffac539ed0
Write	0x7fffac539ecc
Write	0x7fffac539ec8
Read	0x7fffac539ecc

Capture a memory reference trace



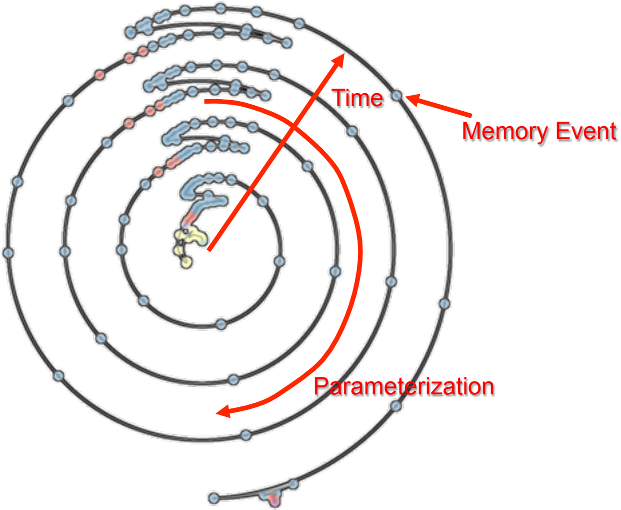
File: sort.cpp

```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```

Write	0x7fffac539ed8
Write	0x7fffac539ed0
Write	0x7fffac539ecc
Write	0x7fffac539ec8
Read	0x7fffac539ecc
Read	0x7fffac539ec8
Write	0x7fffac539eb8
Write	0x7fffac539eb0

Results

Visualization



Datasets

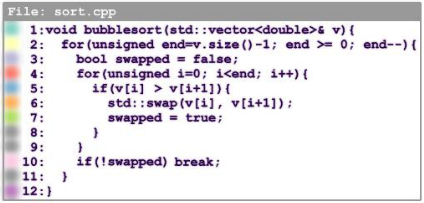
Four algorithms:

- bubble sort
- naïve and blocked matrix multiplication
- material point method

Identified 2 major classes of structure:

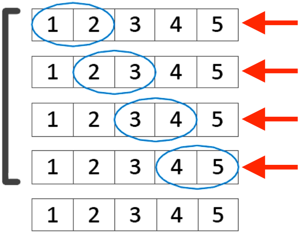
- Data dependent vs. algorithm dependent
- Loop-based vs. non loop-based

Data dependent structure

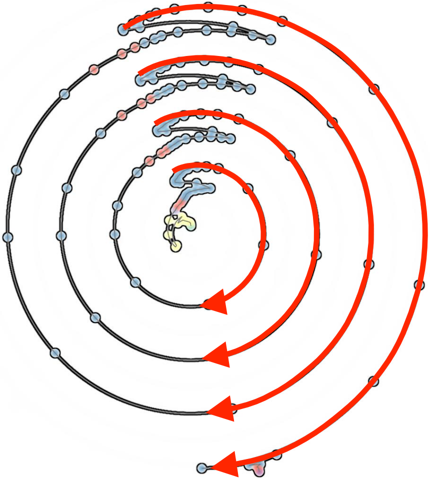


File: sort.cpp

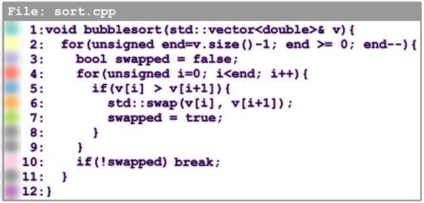
```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```



1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

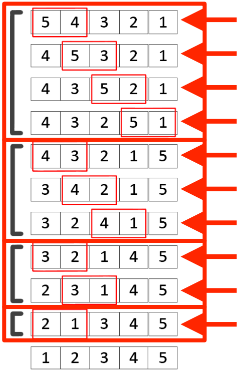


Data dependent structure

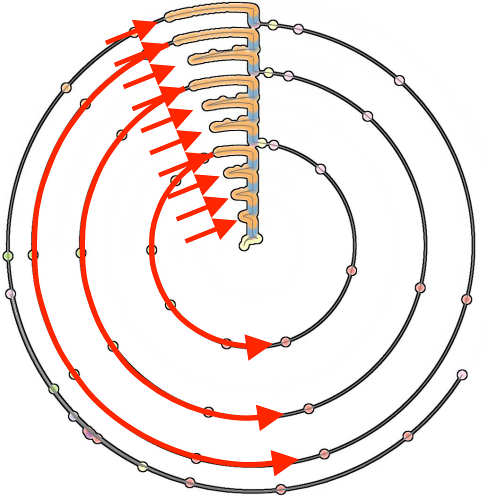


File: sort.cpp

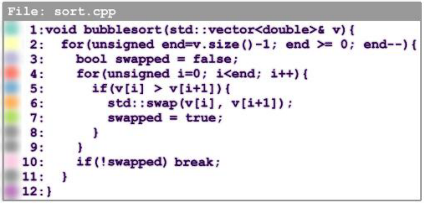
```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```



5	4	3	2	1
4	5	3	2	1
4	3	5	2	1
4	3	2	5	1
4	3	2	1	5
3	4	2	1	5
3	2	4	1	5
3	2	1	4	5
2	3	1	4	5
2	1	3	4	5
1	2	3	4	5

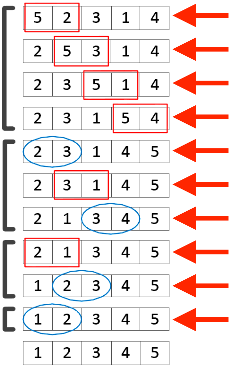


Data dependent structure

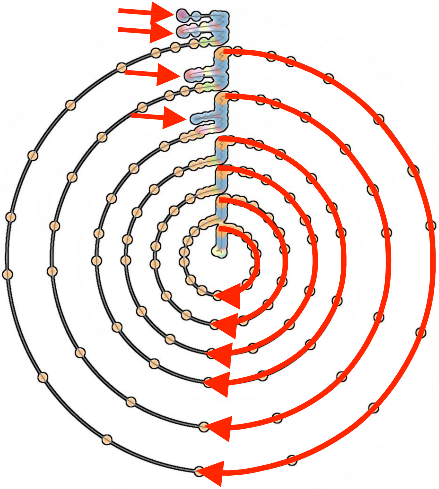


File: sort.cpp

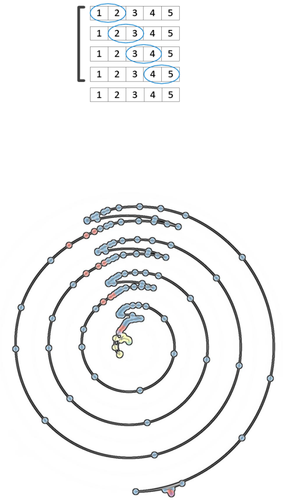
```
void bubblesort(std::vector<double>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```



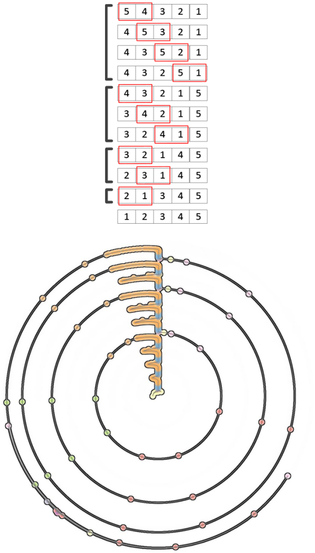
5	2	3	1	4
2	5	3	1	4
2	3	5	1	4
2	3	1	5	4
2	3	1	4	5
2	3	1	4	5
2	1	3	4	5
2	1	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5



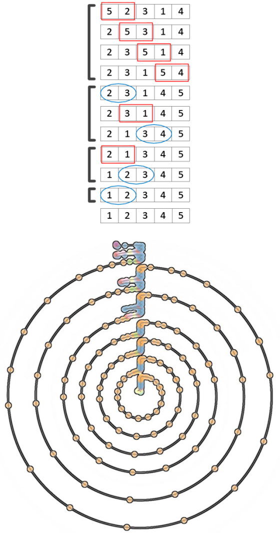
Data dependent structure



1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

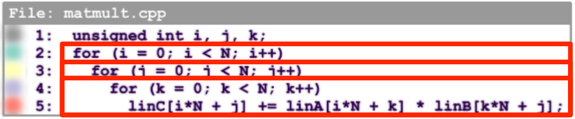


5	4	3	2	1
4	5	3	2	1
4	3	5	2	1
4	3	2	5	1
4	3	2	1	5
3	4	2	1	5
3	2	4	1	5
3	2	1	4	5
2	3	1	4	5
2	1	3	4	5
1	2	3	4	5



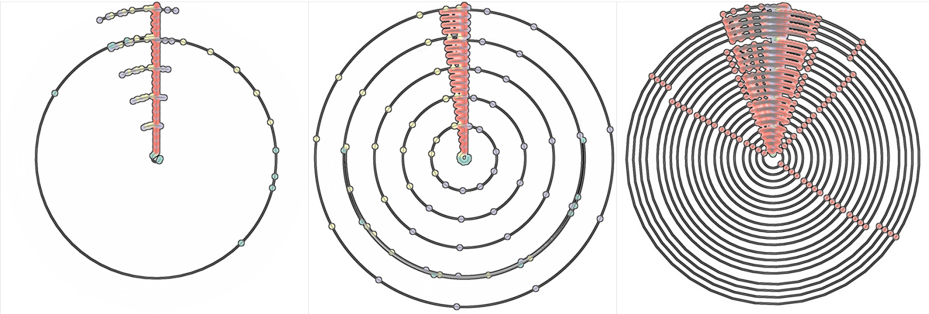
5	2	3	1	4
2	5	3	1	4
2	3	5	1	4
2	3	1	5	4
2	3	1	4	5
2	3	1	4	5
2	1	3	4	5
2	1	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Algorithm dependent structure

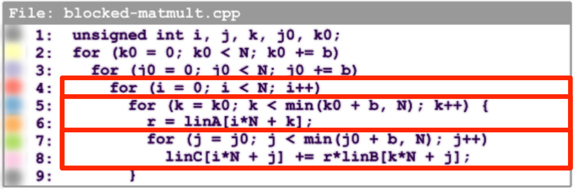


File: matmult.cpp

```
unsigned int i, j, k;
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < N; k++)
      linC[i*N + j] += linA[i*N + k] * linB[k*N + j];
```

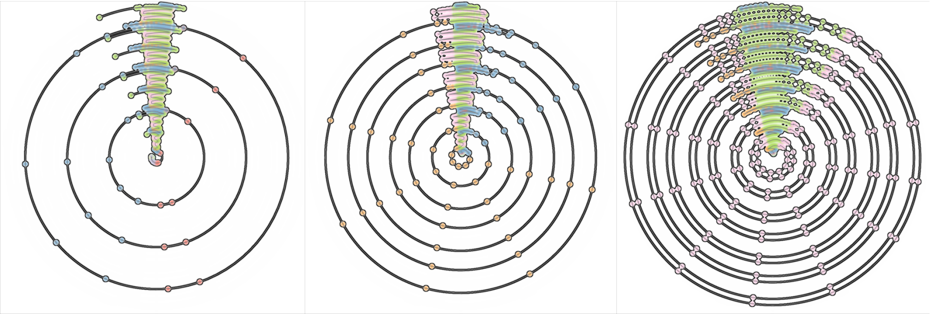


Algorithm dependent structure

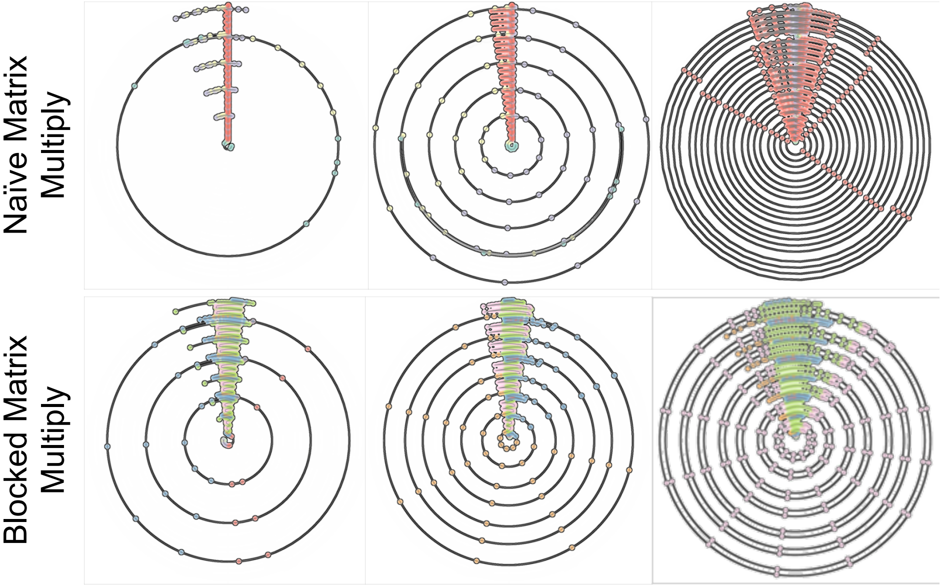


File: blocked-matmult.cpp

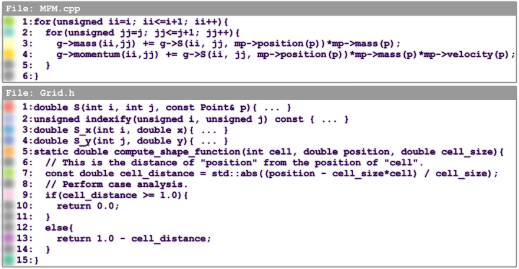
```
unsigned int i, j, k, j0, k0;
for (k0 = 0; k0 < N; k0 += b)
  for (j0 = 0; j0 < N; j0 += b)
    for (i = 0; i < N; i++)
      for (k = k0; k < min(k0 + b, N); k++) {
        r = linA[i*N + k];
        for (j = j0; j < min(j0 + b, N); j++)
          linC[i*N + j] += r*linB[k*N + j];
      }
```



Algorithm dependent structure



Non loop-based structure

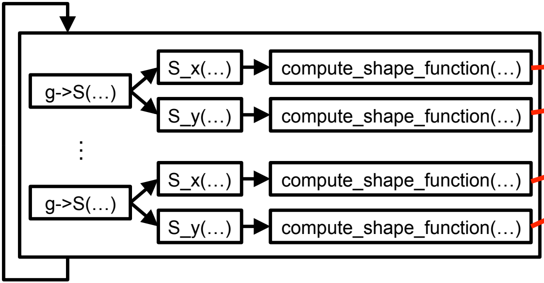


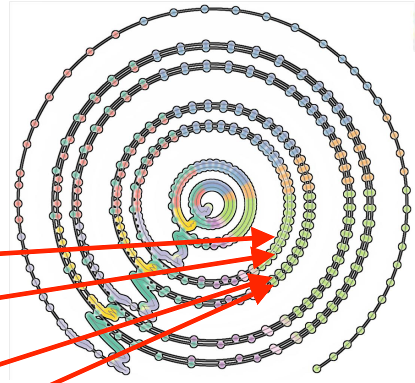
File: MPM.cpp

```
for(unsigned ii=i; ii<=i+1; ii++){
  for(unsigned jj=j; jj<=j+1; jj++){
    g->mass(ii,jj) += g->S(ii, jj, mp->position(p))*mp->mass(p);
    g->momentum(ii,jj) += g->S(ii, jj, mp->position(p))*mp->mass(p)*mp->velocity(p);
  }
}
```

File: Grid.h

```
double S(int i, int j, const Point& p){ ... }
unsigned indexify(unsigned i, unsigned j) const { ... }
double S_x(int i, double x){ ... }
double S_y(int j, double y){ ... }
static double compute_shape_function(int cell, double position, double cell_size){
  // This is the distance of "position" from the position of "cell".
  const double cell_distance = std::abs((position - cell_size*cell) / cell_size);
  // Perform case analysis.
  if(cell_distance >= 1.0){
    return 0.0;
  }
  else{
    return 1.0 - cell_distance;
  }
}
```


g->S(...)
S_x(...)
compute_shape_function(...)
S_y(...)
compute_shape_function(...)
⋮
g->S(...)
S_x(...)
compute_shape_function(...)
S_y(...)
compute_shape_function(...)



Discussions

- Introduced a new domain for exploring software behavior using topological analysis
- Identified some basic classes of structure that appear in software
- Performance analysis? The current connection to performance is loose, but we hope to strengthen that connection through additional study
- Branching structures? Branching in code

Thank you!
beiwang@sci.utah.edu
www.sci.utah.edu/∼beiwang