Feature Flow Fields in Out-of-Core Settings

Tino Weinkauf[1], Holger Theisel[2], Hans-Christian Hege[3], Hans-Peter Seidel[4]

[1] Zuse Institute Berlin (ZIB), Germany. weinkauf@zib.de
[2] Bielefeld University, Germany. theisel@techfak.uni-bielefeld.de
[3] Zuse Institute Berlin (ZIB), Germany. hege@zib.de
[4] MPI Informatik Saarbrücken, Germany. hpseidel@mpi-inf.mpg.de

Summary: Feature Flow Fields (FFF) are an approach to tracking features in a time-dependent vector field $\mathbf{v}$. The main idea is to introduce an appropriate vector field $\mathbf{f}$ in space-time, such that a feature tracking in $\mathbf{v}$ corresponds to a stream line integration in $\mathbf{f}$. The original approach of feature tracking using FFF requested that the complete vector field $\mathbf{v}$ is kept in main memory. Especially for 3D vector fields this may be a serious restriction, since the size of time-dependent vector fields can exceed the main memory of even high-end workstations. We present a modification of the FFF-based tracking approach which works in an out-of-core manner. For an important subclass of all possible FFF-based tracking algorithms we ensure to analyze the data in one sweep while holding only two consecutive time steps in main memory at once. Similar to the original approach, the new modification guarantees the complete feature skeleton to be found. We apply the approach to tracking of critical points in 2D and 3D time-dependent vector fields.

1 Introduction

The resolution of numerical simulations as well as experimental measurements like PIV have evolved significantly in the last years. The challenge of understanding the intricate flow structures within their massive result data sets has made automatic feature extraction schemes popular. Feature-based analysis can be seen as a kind of data reduction since it brings the raw data mass down to a small number of graphical primitives that ought to give insight into the flow structures. While the outcome of most feature extraction algorithms has a rather small memory footprint, the input often exceeds the main memory of high-end workstations. This is especially true for 3D time-dependent data. Thus, feature extraction algorithms should be compatible to an out-of-core data handling, i.e., treating only a small part of the input at once.

A number of algorithms already work in an out-of-core manner. Tricoche et al. [16] and Garth et.al. [3] show how to track critical points in piecewise linear vector fields by analyzing the data in one sweep and holding only two time slices at once. Their approaches exploit the linearity of non-changing piecewise linear grids and are probably the best way to go for this important class of data.

Another way of tracking features which are defined by the parallel vector operator [8] is introduced in [1]. This approach is based on a 4-dimensional isosurface extraction – and therefore compatible to out-of-core data handling.

Theisel et al. [13] propose a general approach to feature tracking by capturing the temporal evolution of a feature using a stream object[5] integration in a derived vector field – the feature flow field (FFF). This basic idea has been applied not only to tracking critical points [13] and derived applications like simplification [11] and comparison [12], but also to extracting Galilean invariant vortex core lines [10] and tracking closed stream lines [14]. The FFF approach is independent of an underlying grid, i.e., it is entirely based on the description of the data as a continuous field. At first glance, this concept seems to contradict the principle of out-of-core data handling: treating only a small part of the data at once. In this paper we show that those two concepts do *not* contradict. In fact, we show how all FFF-based tracking algorithms can be formulated in an out-of-core manner. This will be used to re-formulate the algorithm for tracking critical points from [13] to make it compatible to out-of-core data handling. The resulting algorithm enables to analyze the data in one sweep while holding only two time slices at once.

The rest of the paper is organized as follows: section 2 surveys the FFF approach as described in [13] and the basics of out-of-core data handling. Section 3 describes the new out-of-core version of the FFF approach. Section 4 applies this knowledge to FFF-based tracking of critical points, while conclusions are drawn in section 5.

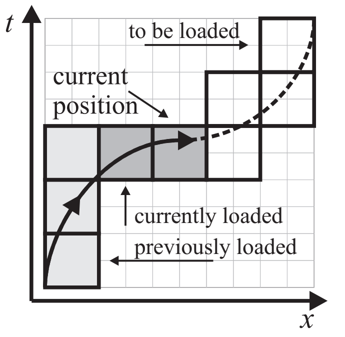
2 Background and Problem

In this section we briefly discuss the basics behind out-of-core data handling and feature flow fields. While their main ideas are not antagonistic, a typical algorithm based on FFF is incompatible with an out-of-core data handling.

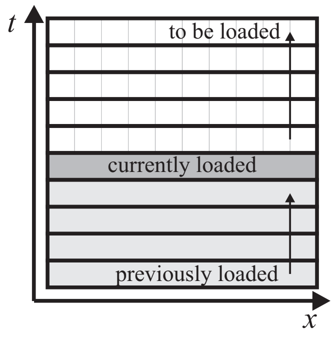
2.1 Out-Of-Core Data Handling

Out-of-core refers to the data handling strategy of algorithms, which process data too large to fit into main memory. Thus, only parts of the data can be loaded at once and acted upon. Since loading the data from a mass storage device is very time-consuming, the number of those operations should be reduced to a minimum. This restriction must already be considered when formulating the algorithm.

[5] This refers to a stream line, stream surface, stream volume, etc. – depending on the dimensionality of the feature.



(a) Block-wise random access.



(b) Slice-wise sequential access.

Fig. 1. Out-of-core data handling strategies.

There are different types of out-of-core data handling strategies. We just want to mention two here:

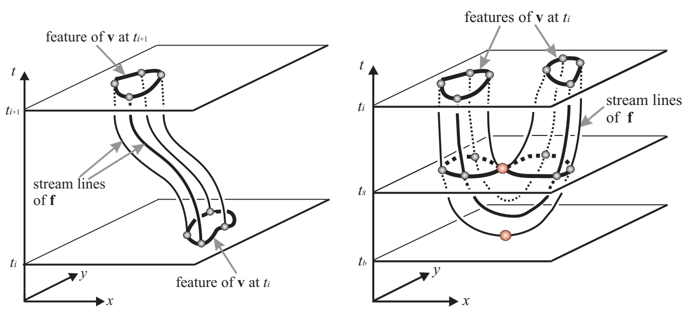
- *Block-wise random access:* Data is loaded in blocks of same size. All dimensions are treated equally. The loaded data block with the oldest access time is subject to be substituted with the next block to be loaded. An application for this access pattern is the integration of a path line, which touches only parts of the domain. Figure 1a gives an illustration.
- *Slice-wise sequential access:* Data is loaded in slices, i.e., one dimension is fixed for every slice. Slices will be loaded as a fixed sequence in ascending or descending order. The procedure to load all slices from first to last is called a *sweep*. A very common approach is the usage of time slices, since a number of data sets are organized such that each time step is given as a separate file. An application for this access pattern is the extraction of fold bifurcations, where all parts of the domain need to be examined. Figure 1b gives an illustration.

Feature extraction algorithms usually do not have a-priori knowledge about the location of the feature and therefore, they need to examine the whole domain. A slice-wise sequential access strategy seems to be even more preferable, if the data is already given in time slices. For the rest of this paper we consider this data handling strategy only. Note, that by loading two consecutive time steps t_i and t_{i+1} and applying a linear interpolation in between them, we obtain the time-dependent field in that time interval.

2.2 Feature Flow Fields

The concept of feature flow fields was first introduced in [13]. It follows a rather generic idea:

Consider an arbitrary point $\mathbf{x}$ known to be part of a feature in a (scalar, vector, tensor) field $\mathbf{v}$. A feature flow field $\mathbf{f}$ is a well-defined vector field at



(a) Tracking features by tracing stream lines.

(b) Events: at the time t_b a new feature is born, at t_s it splits into two features.

Fig. 2. Feature tracking using feature flow fields. Features at t_{i+1} can be observed by intersecting these stream lines with the time plane $t = t_{i+1}$.

$\mathbf{x}$ pointing into the direction where the feature continues. Thus, starting a stream line integration of $\mathbf{f}$ at $\mathbf{x}$ yields a curve where all points on this curve are part of the same feature as $\mathbf{x}$.

FFF have been used for a number of applications, but mainly for tracking features in time-dependent fields. Here, $\mathbf{f}$ describes the dynamic behavior of the features of $\mathbf{v}$: for a time-dependent field $\mathbf{v}$ with n spatial dimensions, $\mathbf{f}$ is a vector field $\mathbb{R}^{n+1} \rightarrow \mathbb{R}^{n+1}$. The temporal evolution of the features of $\mathbf{v}$ is described by the stream lines of $\mathbf{f}$. In fact, tracking features over time is now carried out by tracing stream lines. The location of a feature at a certain time t_i can be obtained by intersecting the stream lines with the time plane t_i. Figure 2a gives an illustration.

Depending on the dimensionality of the feature at a certain time t_i, the feature tracking corresponds to a stream line, stream surface or even higher-dimensional stream object integration. The stream lines of $\mathbf{f}$ can also be used to detect events of the features:

- A birth event occurs at a time t_b, if the feature at this time is only described by one point of a stream object of $\mathbf{f}$, and all stream lines in the neighborhood of this point are in the half-space $t \geq t_b$.
- A split occurs at a time t_s, if one of the stream lines of $\mathbf{f}$ describing the feature touches the plane $t = t_s$ "from above".
- An exit event occurs if all stream lines of $\mathbf{f}$ describing the feature leave the spatial domain.

The conditions for the reverse events (death, merge, entry) can be formulated in a similar way. Figure 2b illustrates the different events.

Integrating the stream lines of $\mathbf{f}$ in forward direction does not necessarily mean to move forward in time. In general, those directions are unrelated. The direction in time may even change along the same stream line as it is shown in figure 2b. This situation is always linked to either a birth and a split event, or a merge and a death event.

Even though we treated the concept of FFF in a rather abstract way, we can already formulate the basics of an algorithm to track all occurrences of a certain feature in a time-dependent field:

Algorithm 1 *General FFF-based tracking*

```
1. Get seeding points/lines/structures such that the stream object integration of $\mathbf{f}$
   guarantees to cover all paths of all features of $\mathbf{v}$.
2. From the seeding structures: apply a numerical stream object integration of $\mathbf{f}$ in
   both forward and/or backward direction until it leaves the space-time domain.
3. If necessary: remove multiply integrated stream objects.
```

Algorithm 1 is more or less an abstract template for a specific FFF-based tracking algorithm like e.g. tracking of critical points. However, it shows a vital contradiction to out-of-core data handling: it gives no guarantee on how the data is processed. We would end up loading different data slices more than once, since both forward and backward integration of $\mathbf{f}$ are allowed, and as already said, the direction in time may even change along the same stream line.

3 Feature Flow Fields and Out-Of-Core Algorithms

In this section we want to modify algorithm 1 such that it becomes compatible to out-of-core data handling. This will allow to formulate all FFF-based algorithms in an out-of-core manner. Before we formulate the algorithm, we collect some concepts and properties of the FFF integration on which the new algorithm is based upon.

3.1 Direction of Integration Regarding Time

The FFF approach is based on a stream line integration of $\mathbf{f}$. Given a starting point $\mathbf{x}_0 = (\mathbf{x}_0^s, t_0)$, $\mathbf{f}$ can be integrated in forward or backward direction. Assuming an Euler integration[6], the forward integration goes to the next point $\mathbf{x}_1 = \mathbf{x}_0 + \varepsilon\,\mathbf{f}(\mathbf{x}_0)$, while the backward integration gives the next point $\mathbf{x}_1 = \mathbf{x}_0 - \varepsilon\,\mathbf{f}(\mathbf{x}_0)$ for a certain small positive ε. In addition to this distinction of the integration orientation, we can also distinguish a t-forward and t-backward orientation. We call an integration *t-forward* if the next point $\mathbf{x}_1 = (x_1^s, t_1)$ is ahead in time, i.e., if $t_1 > t_0$. For $t_1 < t_0$, we have a *t-backward* integration. This property can be decided locally for a point $\mathbf{x}_0$ by looking at the sign of the t-component of $\mathbf{f}(\mathbf{x}_0)$, or if this component is zero, by looking at the sign of the partial derivative $\mathbf{f}_t(\mathbf{x}_0)$. Figure 3 illustrates some of these cases.

[6] For the actual integration we used a fourth order Runge Kutta method, the Euler integration is only for explaining the concepts.

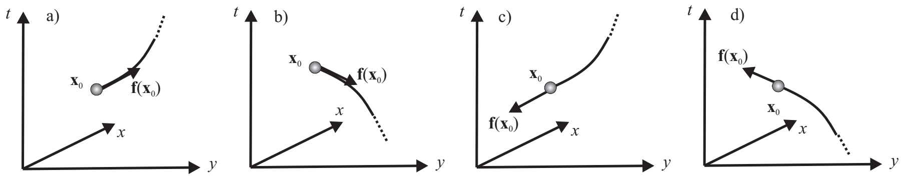


Fig. 3. Orientation of integrating **f**: a) forward and t-forward; b) forward and t-backward; c) backward and t-forward; d) backward and t-backward; the curves are the integrated stream lines starting from $\mathbf{x}_0$.

3.2 Classification of Seeding Structures

The FFF approach is also based on finding appropriate starting structures for the integration. The definition of a complete set of seeding structures is up to the specific FFF-based application. However, we can give the following classification of those structures:

- **t-forward structures:** all integrations started here are t-forward only.
- **t-backward structures:** all integrations started here are t-backward only.
- **intermediate structures:** a t-forward and a t-backward integration will be started here.

This classification is independent of a specific FFF-based application, though it might be that in certain cases a class of structures is empty, e.g. there are only t-forward and intermediate structures but no t-backward structures.

As already discussed in section 2.2, a t-forward integration may change to a t-backward integration even along the same stream line. This situation is always linked to either a birth or a death event, which perfectly fit into the classification: a birth event is a t-forward point, and a death event a t-backward point.

3.3 Out-Of-Core FFF-based Tracking Algorithm

The split of the integration into different directions regarding the time is the conceptual key to an out-of-core version of algorithm 1:

Algorithm 2 *Out-of-core FFF-based tracking*

```
1. Load the data in a slice-wise sequential manner from $t_{min}$ to $t_{max}$. For each time interval between the time slices $t_i$ and $t_{i+1}$:
    a) Get seeding structures such that the stream object integration of $\mathbf{f}$ guarantees to cover all paths of all features of $\mathbf{v}$.
    b) Classify the seeding structures into t-forward, t-backward and intermediate structures.
    c) Start a t-forward integration at all t-forward and intermediate structures. Stop the integration, if
        i. the spatial domain was left.
```

```
      ii. the temporal domain of the time interval was left, i.e., the integration
          reached $t_{i+1}$. Add the result of this integration to the list of t-forward
          structures for the next time interval.
     iii. a death event was reached. If this point will not be reached by any other
          t-forward integration, add the result to the list of t-backward structures.
2. If the list of t-backward structures is non-empty: load the data in a slice-wise
   sequential manner from $t_{max}$ to $t_{min}$. For each time interval between the time
   slices $t_i$ and $t_{i-1}$ start a t-backward integration at all t-backward and interme-
   diate structures similar as above.
3. Repeat the stream object integrations of steps 1 and 2 until the lists of seeding
   structures are empty.
```

The basic idea of this algorithm template is to ensure that the direction of loading the data coincides with the direction of integration, i.e., if we are loading the data from t_{min} to t_{max} then we are integrating t-forward only, and the other way around. This alone does not sound very effective, since the data might been loaded more than once, possibly even an unknown number of times.

This changes, if we take a closer look at the t-forward structures, i.e., all points where *only* a t-forward integration is intended. At those points the features appear for the first time. Examples are entry points, birth events or all occurrences of the feature at t_{min}. If we can find all t-forward structures while doing the first sweep through the data, then the whole feature skeleton can be extracted with this one sweep. This is always fulfilled, if all types of t-forward structures are locally defined, i.e., they can be extracted by a local analysis. Under these prerequisites, we can reformulate algorithm 2 and obtain the following algorithmic template:

Algorithm 3 *One-sweep Out-of-core FFF-based tracking*

```
1. For each time interval $[t_i, t_{i+1}]$ from $t_{min}$ to $t_{max}$:
   a) Extract all t-forward seeding structures needed to cover all paths of all fea-
      tures of $\mathbf{v}$.
   b) Apply a t-forward integration starting at those structures until
        i. the spatial domain was left.
       ii. the temporal domain of the time interval was left, i.e., the integration
           reached $t_{i+1}$. Add the result of this integration to the list of t-forward
           structures for the next time interval.
      iii. a death event was reached.
```

Algorithm 3 ensures that every path of a feature is integrated only once. Thus, a removal of multiply integrated stream objects is not needed anymore. In comparison to algorithms 1 and 2 it is perfectly fitted for large data sets: it reads only parts of the data and each part only once.

4 Application to Tracking of Critical Points

Critical points, i.e. isolated points with a vanishing flow, are perhaps the most important topological feature of vector fields. For static fields, their extraction and classification is well-understood both in the 2D [6] and the 3D case [17]. Critical points also serve as the starting points of certain separatrices, i.e. stream lines/surfaces which divide the field into areas of different flow behavior. Topological methods have not only been developed for visualization purposes [4, 5], but have also been applied to simplify [2, 15], smooth [18], and compress [11, 7] vector fields. A thorough overview can be found in [9].

Considering a stream line oriented topology of time-dependent vector fields, critical points smoothly change their location and orientation over time. In addition, certain bifurcations of critical points may occur. To capture the topological behavior of time-dependent vector fields, it is necessary to capture the temporal behavior of the critical points.

Theisel et al. introduced in [13] a FFF-based approach to track critical points, which matches algorithm 1. In order to track critical points in an effective out-of-core manner using algorithm 3 we need to find the complete set of t-forward points, i.e., all points in space-time where a critical point appears for the first time. This will be done in section 4.2. But before that we discuss the definition of the feature flow field $\mathbf{f}$ itself.

4.1 FFF for Tracking Critical Points

We first consider tracking critical points in a 2D time-dependent vector field, which is given as

$$\mathbf{v}(x,y,t) = \begin{pmatrix} u(x,y,t) \\ v(x,y,t) \end{pmatrix} \tag{1}$$

in the 3D space-time domain $D = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [t_{min}, t_{max}]$. We can construct a 3D vector field $\mathbf{f}$ in D with the following properties: for any two points $\mathbf{x}_0$ and $\mathbf{x}_1$ on a stream line of $\mathbf{f}$, it holds $\mathbf{v}(\mathbf{x}_0) = \mathbf{v}(\mathbf{x}_1)$. This means that a stream line of $\mathbf{f}$ connects locations with the same values of $\mathbf{v}$. Figure 4 gives an illustration. In particular, if $\mathbf{x}_0$ is a critical point in $\mathbf{v}$, then the stream line of $\mathbf{f}$ describes the path of the critical point over time. To get $\mathbf{f}$, we search for the direction in space-time in which both components of $\mathbf{v}$ locally remain constant. This is the direction perpendicular to the gradients of the two components of $\mathbf{v}$. We get

$$\mathbf{f}(x,y,t) = \operatorname{grad}(u) \times \operatorname{grad}(v) = \begin{pmatrix} u_x \\ u_y \\ u_t \end{pmatrix} \times \begin{pmatrix} v_x \\ v_y \\ v_t \end{pmatrix} = \begin{pmatrix} \det(\mathbf{v}_y, \mathbf{v}_t) \\ \det(\mathbf{v}_t, \mathbf{v}_x) \\ \det(\mathbf{v}_x, \mathbf{v}_y) \end{pmatrix}. \tag{2}$$

The FFF approach for 3D vector fields is a straightforward extension of the 2D case. Given the 3D time-dependent vector field

$$\mathbf{v}(x,y,z,t) = \begin{pmatrix} u(x,y,z,t) \\ v(x,y,z,t) \\ w(x,y,z,t) \end{pmatrix} \tag{3}$$

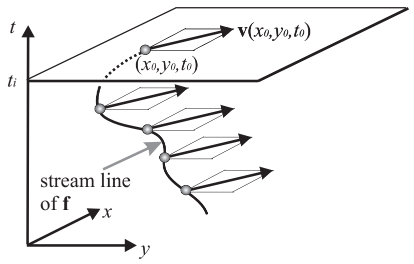


Fig. 4. Tracking 2D critical points: all points on a stream line of **f** have the same value for **v**.

in the 4D space-time domain $D = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}] \times [t_{min}, t_{max}]$, the 4D FFF **f** is defined by the conditions

$$\mathbf{f} \perp \operatorname{grad}(u) = (u_x, u_y, u_z, u_t)^T \ \ , \ \ \mathbf{f} \perp \operatorname{grad}(v) \ \ , \ \ \mathbf{f} \perp \operatorname{grad}(w).$$

This gives a unique solution for **f** (except for scaling)[7]

$$\mathbf{f}(x,y,z,t) = \begin{pmatrix} +\det(\mathbf{v}_y, \mathbf{v}_z, \mathbf{v}_t) \\ -\det(\mathbf{v}_z, \mathbf{v}_t, \mathbf{v}_x) \\ +\det(\mathbf{v}_t, \mathbf{v}_x, \mathbf{v}_y) \\ -\det(\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z) \end{pmatrix} . \tag{4}$$

4.2 Complete Set of *t*-forward Points

Theisel and Weinkauf showed in [14] that two classes of seeding points guarantee that all paths of critical points are captured: the intersections of the paths with the domain boundaries, and fold bifurcations. However, they did not distinguish between t-forward and t-backward points. We are going to do this here in order to find the complete set of t-forward points.

To find all intersections with the boundaries, we have to solve

$$\begin{aligned} &\mathbf{v}(x,y,t_{min}) = (0,0)^T \text{ and } \mathbf{v}(x,y,t_{max}) = (0,0)^T \text{ for the unknowns } x, y, \\ &\mathbf{v}(x,y_{min},t) = (0,0)^T \text{ and } \mathbf{v}(x,y_{max},t) = (0,0)^T \text{ for the unknowns } x, t, \\ &\mathbf{v}(x_{min},y,t) = (0,0)^T \text{ and } \mathbf{v}(x_{max},y,t) = (0,0)^T \text{ for the unknowns } y, t. \end{aligned}$$

Each of the 6 solutions turns out to be a simple extraction of critical points of a 2D (steady) vector field. We can make the following distinction:

- *Bottom intersection points* are intersections with the plane $t = t_{min}$
- *Top intersection points* are intersections with the plane $t = t_{max}$
- *Side intersection points* are intersections with the plane $x = x_{min}$, $x = x_{max}$, $y = y_{min}$, or $y = y_{max}$ respectively.

Side intersection points can be further classified into entry and exit points. At an entry point, a t-forward integration of **f** goes into D, while at an exit point

[7] Note that the formulation of $\mathbf{f}(x,y,z,t)$ in [13] contains an error: the alternating signs of the components are missing.

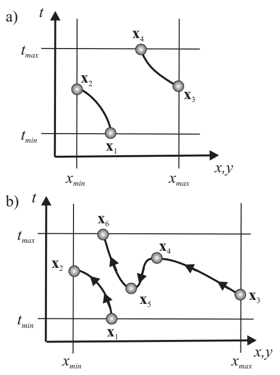


Fig. 5. a) intersections of the paths of critical points with the domain of D: bottom intersection ($\mathbf{x}_1$), exit ($\mathbf{x}_2$) and entry ($\mathbf{x}_3$) side intersection, top intersection ($\mathbf{x}_2$); b) example consisting of a bottom intersection ($\mathbf{x}_1$), exit ($\mathbf{x}_2$) and entry ($\mathbf{x}_3$) side intersection, death ($\mathbf{x}_4$) and birth ($\mathbf{x}_5$) fold bifurcation, top intersection ($\mathbf{x}_6$): the paths of critical points consist of 4 segments which are integrated t-forward from→to: $\mathbf{x}_1 \rightarrow \mathbf{x}_2$, $\mathbf{x}_3 \rightarrow \mathbf{x}_4$, $\mathbf{x}_5 \rightarrow \mathbf{x}_4$, $\mathbf{x}_5 \rightarrow \mathbf{x}_6$.

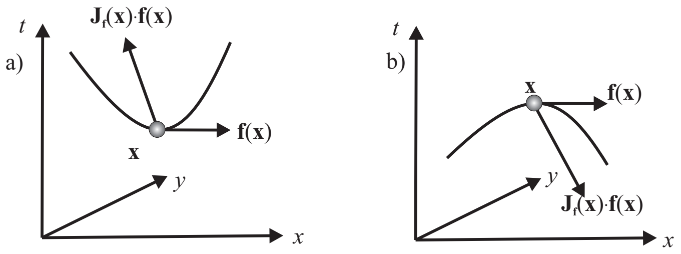


Fig. 6. Classifying fold bifurcations by the last component of $\mathbf{J_f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x})$: a) birth event; b) death event.

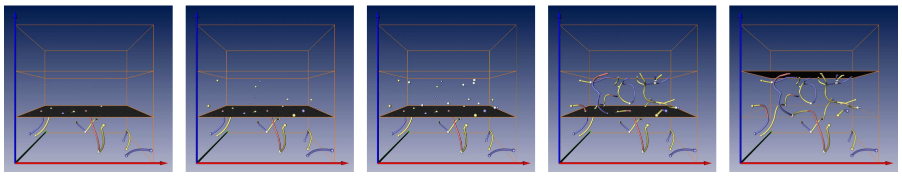
a t-forward integration leaves D. Figure 5a illustrates the different kinds of intersection points with the boundary of D. It is easy to see that only bottom and entry side intersections are t-forward points.

To detect fold bifurcations inside D, we search for locations $\mathbf{x}$ with

$$[\ \mathbf{v}(\mathbf{x}) = (0,0)^T \ \ , \ \ \det(\mathbf{J_v}(\mathbf{x})) = 0\ ] \tag{5}$$

where $\mathbf{J_v}$ is the Jabcobian matrix of $\mathbf{v}$. (2) shows that the second condition of (5) ensures that the last component of $\mathbf{f}$ vanishes, i.e., that $\mathbf{f}$ is parallel to the t-axis. To solve (5), we use a numerical approach similar to extracting isolated critical points in 3D vector fields. There are two kinds of fold bifurcations: birth and death events. To distinguish them, we consider the last component of $\mathbf{J_f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x})$ at the fold bifurcation. If this component is positive, we have a birth bifurcation; if it is negative, a death bifurcation is present. Figure 6 illustrates this. It is easy to see that only birth bifurcations are t-forward points.

For 3D time-dependent vector fields, the extraction of the seeding points follows the same ideas. Boundary intersections are found as isolated critical points of the 3D (steady) vector fields at the space-time domain boundary. Again, only bottom and entry side intersections are of interest. Fold bifurcations are the solutions of



(a) At t_i. (b) Entries. (c) Births. (d) Integration. (e) At t_{i+1}.

Fig. 7. Application of algorithm 3: critical points tracked in one sweep.

$$[\ \mathbf{v}(\mathbf{x}) = (0,0,0)^T \ \ , \ \ \det(\mathbf{J}_\mathbf{v}(\mathbf{x})) = 0\ ] \tag{6}$$

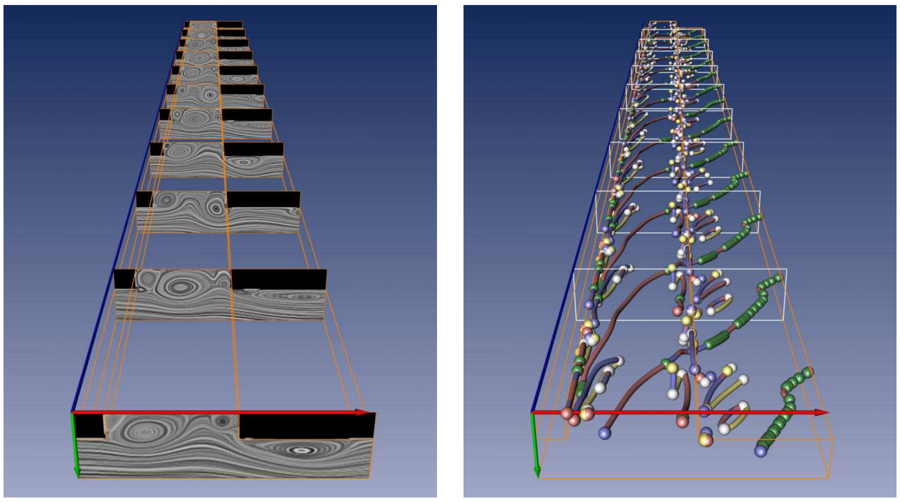
which corresponds to numerically finding isolated critical points in 4D vector fields. The distinction between births and deaths follows the 2D case.

We have found all points in space-time where a critical point can appear for the first time: bottom and entry side intersections as well as birth bifurcations. They can all be extracted using a local analysis. All prerequisites for algorithm 3 are fulfilled. Thus, we are now able to track critical points in 2D and 3D time-dependent vector fields in an effective out-of-core manner: in one sweep and by loading only two slices at once. We applied this algorithm to a random 2D time-dependent data set. Random vector fields are useful tools for a proof-of-concept of topological methods, since they contain a maximal amount of topological information. Figure 7 shows the execution of algorithm 3 between two consecutive time steps t_i and t_{i+1}.

Figure 8 shows the visualization of a vector field describing the flow over a 2D cavity. This data set was kindly provided by Mo Samimy and Edgar Caraballo (both Ohio State University) as well as Bernd R. Noack and Ivanka Pelivan (both TU Berlin). 1000 time steps have been simulated using the *compressible* Navier-Stokes equations; it exhibits a non-zero divergence inside the cavity, while outside the cavity the flow tends to have a quasi-divergence-free behavior. Instead of loading only two time steps at once, the data set has been divided into 10 sections each consisting of 100 time steps (Figure 8a). Each section fits easily into main memory and has been treated according to algorithm 3. This approach reduced the overhead introduced by the out-of-core handling. The computation time for this data set was 20 minutes. The topological structures visualized in Figure 8b elucidate the quasi-periodic nature of the flow. The most dominating topological structures originate in or near the boundaries of the cavity itself. The quasi-divergence-free behavior outside the cavity is affirmed by the fact that a high number of Hopf bifurcations has been found in this area.

5 Conclusions

In this paper we showed how all FFF-based tracking algorithms can be formulated in an out-of-core manner. This has been used to re-formulate the



(a) LIC planes: 10 sections. (b) Tracked critical points.

Fig. 8. Cavity data set consisting of 1000 time steps. Algorithm 3 has been applied onto the 10 depicted sections consisting of 100 time steps each.

algorithm for tracking critical points from [13] to make it compatible to out-of-core data handling. The resulting algorithm enables to analyze the data in one sweep while holding only two time slices at once. For future work, we intend to apply algorithm 3 to other types of features, especially to vortex core lines.

References

1. D. Bauer and R. Peikert. Vortex tracking in scale space. In *Data Visualization 2002. Proc. VisSym 02*, pages 233–240, 2002.
2. W. de Leeuw and R. van Liere. Collapsing flow topology using area metrics. In *Proc. IEEE Visualization '99*, pages 149–354, 1999.
3. C. Garth, X. Tricoche, and G. Scheuermann. Tracking of vector field singularities in unstructured 3D time-dependent datasets. In *Proc. IEEE Visualization 2004*, pages 329–336, 2004.
4. A. Globus, C. Levit, and T. Lasinski. A tool for visualizing the topology of three-dimensional vector fields. In *Proc. IEEE Visualization '91*, pages 33–40, 1991.
5. H. Hauser and E. Gröller. Thorough insights by enhanced visualization of flow topology. In *9th international symposium on flow visualization*, 2000.
6. J. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer*, 22(8):27–36, 1989.
7. S. Lodha, N. Faaland, and J. Renteria. Topology preserving top-down compression of 2d vector fields using bintree and triangular quadtrees. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):433–442, 2003.

8. R. Peikert and M. Roth. The parallel vectors operator - a vector field visualization primitive. In *Proc. IEEE Visualization 99*, pages 263–270, 1999.
9. F.H. Post, B. Vrolijk, H. Hauser, R.S. Laramee, and H. Doleisch. Feature extraction and visualisation of flow fields. In *Proc. Eurographics 2002, State of the Art Reports*, pages 69–100, 2002.
10. J. Sahner, T. Weinkauf, and H.-C. Hege. Galilean invariant extraction and iconic representation of vortex core lines. In *Proc. EuroVis 2005*, pages 151–160, 2005.
11. H. Theisel, Ch. Rössl, and H.-P. Seidel. Combining topological simplification and topology preserving compression for 2d vector fields. In *Proc. Pacific Graphics 2003*, pages 419–423, 2003.
12. H. Theisel, Ch. Rössl, and H.-P. Seidel. Using feature flow fields for topological comparison of vector fields. In *Proc. Vision, Modeling and Visualization 2003*, pages 521–528, 2003.
13. H. Theisel and H.-P. Seidel. Feature flow fields. In *Data Visualization 2003. Proc. VisSym 03*, pages 141–148, 2003.
14. H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Topological methods for 2D time-dependent vector fields based on stream lines and path lines. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):383–394, 2005.
15. X. Tricoche, G. Scheuermann, and H. Hagen. Continuous topology simplification of planar vector fields. In *Proc. Visualization 01*, pages 159 – 166, 2001.
16. X. Tricoche, T. Wischgoll, G. Scheuermann, and H. Hagen. Topology tracking for the visualization of time-dependent two-dimensional flows. *Computers & Graphics*, 26:249–257, 2002.
17. T. Weinkauf, H. Theisel, H.-C. Hege, and H.-P. Seidel. Boundary switch connectors for topological visualization of complex 3D vector fields. In *Proc. VisSym 04*, pages 183–192, 2004.
18. R. Westermann, C. Johnson, and T. Ertl. Topology-preserving smoothing of vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):222–229, 2001.