Temporal Merge Tree Maps: A Topology-Based Static Visualization for Temporal Scalar Data

Wiebke Köpp and Tino Weinkauf

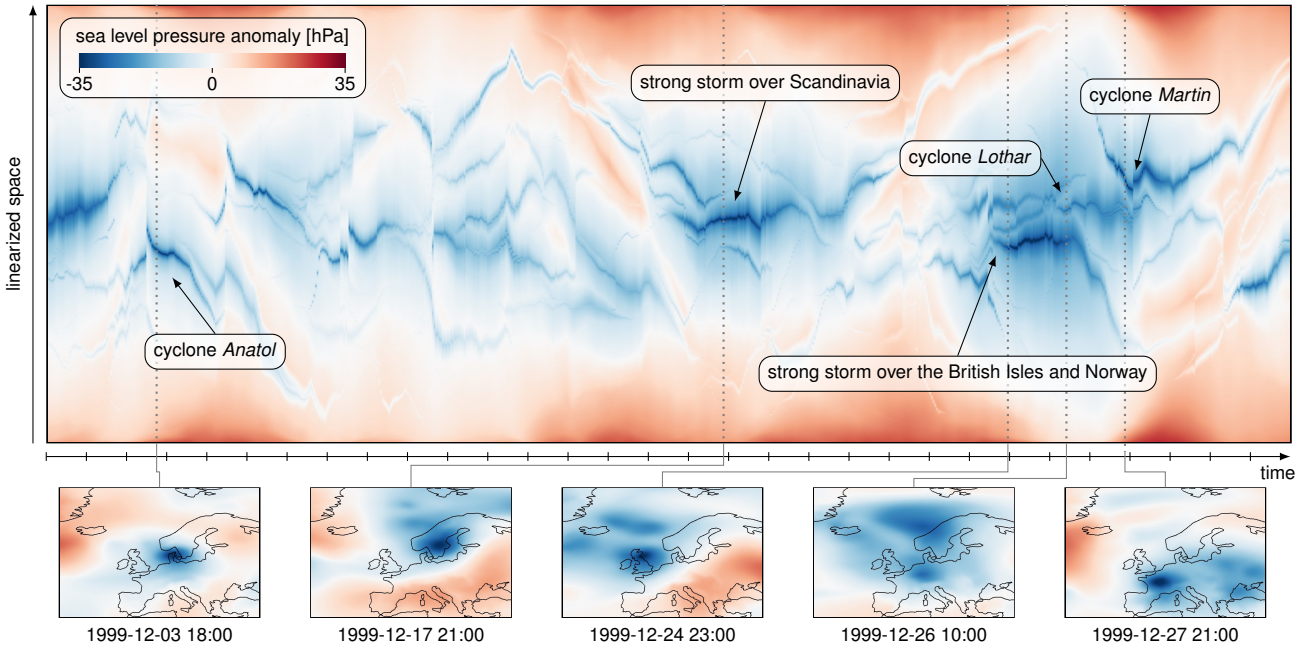


Figure 1. A *temporal merge tree map* (top) is a static visualization of a time-dependent scalar field (bottom). Our method uses augmented merge trees to map the data samples of each time step to a vertical slice. An optimization scheme is employed to achieve a temporally coherent mapping. The shown *Storms* data set represents storm activity over Europe in December 1999 and contains 744 time steps, arranged from left to right. Cyclones can easily be identified as dark blue lines and compared with each other wrt. their strength, lifetime, and footprint, while still being shown in the context of the entire data. Temporal merge tree maps can serve as data analysis tools in their own right, or be used to augment animations and other views. With some parallel processing, the above image can be computed in less than 15 seconds.

Abstract— Creating a static visualization for a time-dependent scalar field is a non-trivial task, yet very insightful as it shows the dynamics in one picture. Existing approaches are based on a linearization of the domain or on feature tracking. Domain linearizations use space-filling curves to place all sample points into a 1D domain, thereby breaking up individual features. Feature tracking methods explicitly respect feature continuity in space and time, but generally neglect the data context in which those features live. We present a feature-based linearization of the spatial domain that keeps features together and preserves their context by involving all data samples. We use augmented merge trees to linearize the domain and show that our linearized function has the same merge tree as the original data. A greedy optimization scheme aligns the trees over time providing temporal continuity. This leads to a static 2D visualization with one temporal dimension, and all spatial dimensions compressed into one. We compare our method against other domain linearizations as well as feature-tracking approaches, and apply it to several real-world data sets.

Index Terms—Scalar field visualization, augmented merge tree, pixel-based visualization

1 Introduction

Essentially all natural phenomena are time-dependent. This includes the weather, the climate, fluid flows, biological processes, chemical reactions, and so on. Understanding the dynamics of these phenomena is a common goal in data analysis.

Time-dependent data can be viewed dynamically or statically. A dynamic visualization employs an animated sequence of images to convey the dynamics of the data in a transient manner. Such animations are commonplace in data visualization and easy to comprehend as the dynamics of the data match the dynamics of the animated visualization. Yet, animations cannot be used in all contexts (e.g., printed on paper) and their transient nature makes it difficult to perceive certain aspects: rapid fluctuations can easily be missed, moving objects are difficult to count, and even the dynamics from different parts of the animation are difficult to compare.

Static visualizations of dynamic data show all time steps at once, either by superimposing them or by assigning a spatial dimension to time. Depending on the data, this can lead to too much information being shown at once and to real estate being taken away from other

• *All authors are with KTH Royal Institute of Technology, Stockholm, Sweden. E-mail: {wiebkek | weinkauf}@kth.se .*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

visualization aspects. Nonetheless, they are a good complement to dynamic visualizations as they are often able to compensate for the issues with animations mentioned above.

Approaches for the static visualization of time-dependent scalar fields fall into two categories: domain linearizations and feature tracking methods. Linearizing a domain means to arrange all sample points of the original 2D/3D data in a 1D domain. The temporal dimension can then be placed orthogonally. Previous work uses space-filling curves [11] to achieve this. Unfortunately, space-filling curves do not keep features intact, which makes certain analysis tasks impossible such as counting the number of features.

Feature tracking methods follow distinct objects of interest over time and create feature paths/surfaces [37,38] or tracking graphs [22,31,45], which can then easily be used for a static visualization. Features are naturally kept intact with these approaches, but information is lost about more general aspects of the data such as how the distribution of the data looks like, i.e., we lose the data context in which these features are living. This happens because (i) data samples not contained in a feature are excluded from further processing, and (ii) data samples within a feature are summarized to a few statistical moments.

We propose a feature-based domain linearization method. It keeps features intact while keeping their data context by arranging all data samples in the 1D domain. We build upon merge trees, which provide a feature-based hierarchical decomposition of the spatial domain, which we use to convert the original 2D/3D scalar field to a 1D function. Notably, we show that under very mild assumptions the original data and the 1D function are topologically equivalent in the sense that they have the same merge tree that was used for the linearization.

We create a static, two-dimensional visualization of the entire time-dependent data set by placing the temporal dimension orthogonally to the linearized spatial domain. We propose an optimization scheme to synchronize the linearizations of neighboring time steps, i.e., to achieve temporal coherence such that users can visually follow features over time.

We give the following contributions:

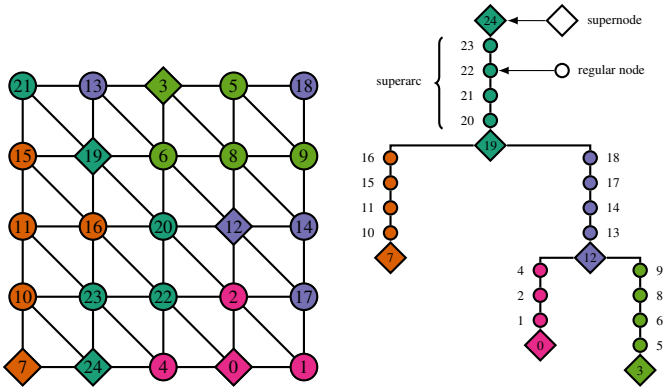
- a feature-based domain linearization that keeps features intact and preserves their context by involving all data samples (Section 3.1),
- a formulation of a discrete optimization problem to address temporal coherence with a practical heuristic to solve this with little computational effort (Section 3.2),
- an evaluation and several comparisons to linearization and feature-based methods (Section 4), and
- the application of our method to 2D and 3D time-dependent data sets (Section 5).

2 Related Work and Background

2.1 Augmented Merge Trees

Consider the sublevel sets $\{\mathbf{x} \in \mathbb{R}^n | s(\mathbf{x}) \leq \alpha\}$ of a scalar field $s : \mathbb{R}^n \to \mathbb{R}$ for an increasing isovalue α: each local minimum gives rise to a connected component, they merge at saddles until only one component remains, which will finally collapse at the global maximum. We can record this behavior in the *join tree* data structure as shown in Figure 2: the minima are the leaves, the saddles are inner parents, and the global maximum is the root. We refer to them as the *supernodes* of the join tree, which are connected to each other via *superarcs.* The superarcs represent the connected components of the sublevel sets. All data samples can be assigned to a superarc. Most data samples are *regular nodes* not giving rise to a topological change. If we opt to store regular nodes in the tree data structure, we call it an *augmented* join tree.

Doing this for a decreasing isovalue gives us the *split tree* for the superlevel sets. We can refer to both of them as *merge trees.* We refer to Hamish Carr's excellent PhD thesis for more details [6]. Figure 2 illustrates these concepts using a small example scalar field, which we continue to use later on for the explanations in Figures 4 and 5. In the remainder of this paper, merge trees are assumed to be binary trees. This means degenerate cases such as plateaus have been dealt with through strategies such as simulation of simplicity [10] and multi-saddles are handled by repeatedly grouping children until a binary tree is obtained. This procedure converts some of the multi-saddle children



(a) Example scalar field given on a triangulated grid.

(b) Augmented merge tree and associated terminology.

Figure 2. Example scalar field and its associated join tree. The different colors represent the superarcs. See also Figures 4 and 5.

from regular nodes to supernodes. The *Cylinder* data set from Figure 14 features on average one multi-saddle per time step.

2.2 Visualization and Tracking of (Augmented) Merge Trees

All tree visualization methods can also be applied to merge trees. A common example is the treemap [34], which finds its application in topological landscapes [43]. Merge trees are an interesting mediator between dimensions: they can be computed for any-dimensional scalar data, and they can be visualized in any-dimensional space. For example, Oesterling et al. [24] propose 1D topological landscape profiles for multi-dimensional point clouds, where each superarc is represented by a hill-like icon parameterized by size and persistence of that superarc. Similar 1D topological landscape profiles have also been introduced for barrier trees [40,41] of optimization landscapes and level set trees [17] of density estimates. Under specific parameter settings and applied to scalar fields, these yield equivalent results to our proposed domain linearization.

Tracking merge trees over time is a computationally very expensive matter, as described by Oesterling et al. [23]. Alternative approaches by Lohfink et al. [19, 20] and Pont et al. [27] achieve much faster computation times by employing heuristics to compute tree alignments for a given similarity metric.

Our approach uses merge trees from every time step as well, but we use a different means to achieve temporal coherence: we formulate it as a discrete optimization problem directly linking the original data and the 1D output data. This serves our visualization purpose more directly and is also very fast to compute. Furthermore, none of the methods from above have been used to create static, two-dimensional visualizations of a time-dependent data set.

2.3 Feature Tracking

A plethora of feature tracking methods exist to accommodate different feature and data types. Point- or line-type features could be tracked by solving an ODE called *Feature Flow Fields* [37,38], or by applying the *Parallel Vectors* operator [2,26]. The results of these methods are often superimposed on the original domain for visualization purposes. Region-based features (same dimensionality as the domain) are often tracked by overlap [18,21,22,35,45] or statistical moments such as their histogram [30]. These result in tracking graphs whose layout in the plane is a topic of research [18,22]. Region-based tracking methods are related to our work in the sense that they often also look at sub/superlevel sets. However, the methods above employ fixed thresholds, while our work uses merge trees describing the topology of sub/superlevel sets over the entire data range without any thresholds.

One goal of feature-based methods is to drastically reduce the amount of information coming with the original data to a small set of features. Visualizations of feature tracks are therefore often abstract

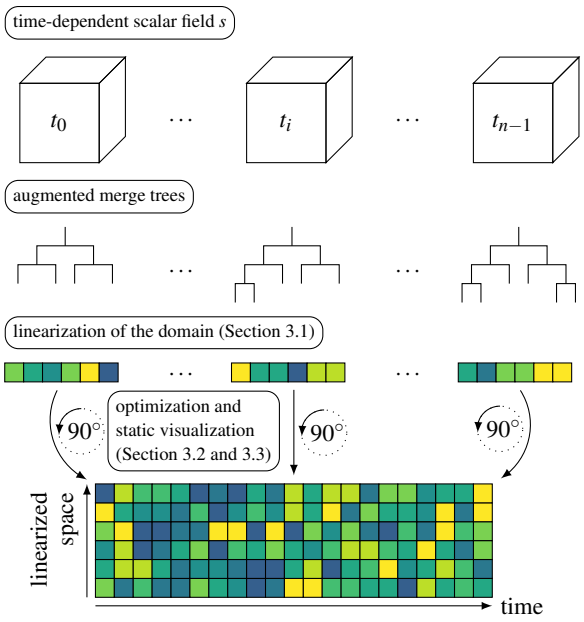


Figure 3. Overview of our method.

– such as the Nested Tracking Graphs [22] – and carry only little information about the data besides the existence and correspondence of features. In contrast, our work aims at showing features together with the entire data context in one image.

2.4 Data Linearizations

The highly astonishing fact that there is a 1-to-1 correspondence between the points on the unit line segment and the points in n-dimensional space was first observed and proven by Georg Cantor [5]. Motivated by this result, Giuseppe Peano developed the first space-filling curve [25], which in turn inspired David Hilbert to come up with one of the most widely used space-filling curves in computer science [15].

Space-filling curves also find their applications in the visualization domain to map a higher-dimensional data set to a 1D domain with the goal of comparing data sets [8, 44]. Franke et al. [11] use them for spatio-temporal visualizations similar to ours and compare several curves with each other for their suitability. Most space-filling curves are purely geometrically motivated and data-unaware. Among the few exceptions are Dafner et al. [7] and Zhou et al. [46] who propose data-aware space-filling curves.

None of the above methods keeps the features of a scalar field intact as we show in Section 4.3.

Other methods for deriving a linear order work on point sets and use hierarchical clustering [12] or multi-dimensional scaling [32]. They are not directly applicable to scalar fields.

3 Temporal Merge Tree Maps

We propose a method (Figure 3) to linearize a scalar field based on its merge tree (Section 3.1). We show that this method preserves the merge tree topology and therefore keeps features intact The mapping of consecutive time steps is optimized towards resembling the dynamics of the scalar field as closely as possible (Section 3.2). The optimized one-dimensional mappings are combined into a pixel-based static visualization (Section 3.3) which we call a *temporal merge tree map*. We consider the following to be given:

- A series of n scalar fields $s(\mathbf{x},t)$ with $t \in [t_0,\dots,t_{n-1}]$. Each time step contains m data samples.
- An augmented merge tree $M \in [M_0,\dots,M_{n-1}]$ for every time step. We use readily available algorithms in the open-source library *Topology Toolkit (TTK)* [39] to compute those.

Algorithm 1: Traversal of an augmented merge tree M to derive a mapping of positions in nD to 1D.

Data: An augmented merge tree $M = (N,E)$
Result: A position $x_i \in [0,\dots,m-1]$ for each node in M

Function `ProcTree`(M)**:**
  the root node is placed at $x = 0$
  $x_L, x_R \leftarrow$ `PlaceNode`(*root*, 0, $m-1$, *true*)
  $x_L, x_R \leftarrow$ `ProcSuperArc`(*child superarc of root*, x_L, x_R)
  `ProcSuperNode`(*child supernode of root*, x_L, x_R)

Function `PlaceNode`(*node or supernode*, x_L, x_R, left)**:**
  if left**:**
    $x_i \leftarrow x_L$; $x_L \leftarrow x_L + 1$
  else:
    $x_i \leftarrow x_R$; $x_R \leftarrow x_R - 1$
  return x_L, x_R

Function `ProcSuperArc`(*superarc*, x_L, x_R)**:**
  left $\leftarrow$ false
  for *node in superarc***:**
    $x_L, x_R \leftarrow$ `PlaceNode`(*node*, x_L, x_R, left)
    left $\leftarrow$ not left
    return x_L, x_R

Function `ProcSuperNode`(*supernode*, x_L, x_R)**:**
  $n \leftarrow$ size of first child superarc and subtree below it
  $x_C \leftarrow x_L + n$
  the supernode is placed at x_C
  `PlaceNode`(*supernode*, x_C, x_C, *true*)
  $x_L^1, x_R^1 \leftarrow$ `ProcSuperArc`(*first child arc*, x_L, $x_C - 1$)
  `ProcSuperNode`(*first child super node*, x_L^1, x_R^1)
  $x_L^2, x_R^2 \leftarrow$ `ProcSuperArc`(*second child arc*, $x_C + 1$, x_R)
  `ProcSuperNode`(*second child super node*, x_L^2, x_R^2)

3.1 Mapping a Single Time Step: $\mathbb{R}^d \to \mathbb{R}$

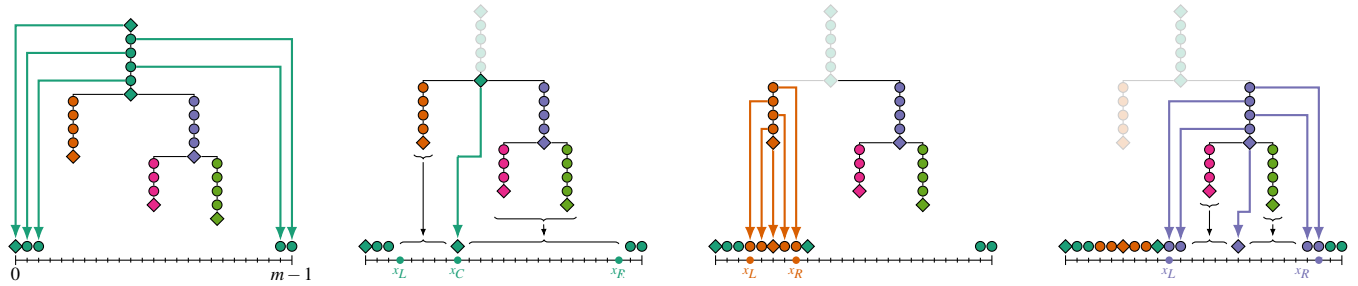
Consider the d-dimensional scalar field s for a given time step with its merge tree M. We want to find a function $g : \mathbb{R}^d \to \mathbb{R}$ that maps each original sample *location* $\mathbf{x} \in \mathbb{R}^d$ of the given scalar field s to a one-dimensional sample *location* $x \in \mathbb{R}$ in our output scalar field f. This is a linearization of the *domain*, while the function values of the samples remain the same. Straightforwardly, we use integer locations for the output, i.e, $g : \mathbb{R}^d \to [0,\dots,m-1]$. This aids the subsequent conversion into pixels.

Under very mild assumptions, our newly constructed output scalar field has the same merge tree M as the input scalar field. This means, we see as many maxima in our output as there are in the split tree of our input (or as many minima as there are in the join tree). This is crucial to retaining features described by minima/maxima (cyclones in climate data, vortices in flow data, etc.) in our final results. We show this *Merge Tree Identity* property in the supplemental material.

We derive the mapping g through a depth-first traversal of M. At each stage of the algorithm, we maintain a contiguous output range $[x_L, x_R]$ of unassigned sample locations to be used for all upcoming nodes in the tree traversal. The following explanations are accompanied by illustrations in Figure 4 and pseudocode is given in Algorithm 1.

Root We start with $x_L = 0$ and $x_R = m-1$. The root is placed at the left side of the output range at x_L (Figure 4(a)). Thereafter, our active range of unassigned sample locations is between $x_L = 1$ and $x_R = m-1$.

Traversal of a Superarc The regular nodes of a superarc are placed in the remaining range $[x_L, x_R]$ from the outside inwards in an *alternating* fashion: the first node is placed on the right at x_R, the second node on the left at x_L, then back to the right side at $x_R = x_R - 1$, and so on. Figure 4(a) shows this. Starting on the right side is an implementation choice: we could just as well start on the left, or even



(a) The root ◆ is placed at the left. The regular nodes ● are placed in an alternating fashion from the outside inwards.

(b) At an inner supernode ◆, the remaining space $[x_L, x_R]$ is distributed between the left ● and the right ● child superarcs, with the supernode in between.

(c) The nodes ● of the left child superarc are placed in an alternating fashion and the recursion ends with the leaf ◆.

(d) The process repeats recursively until all nodes are placed.

Figure 4. We linearize each time step of our input scalar field (2D/3D) with the help of its augmented merge tree. The procedure builds on a depth-first traversal of the merge tree. Importantly, under very mild assumptions, the resulting 1D scalar field has the same merge tree as the original data, i.e., we keep features intact during our linearization process. Algorithm 1 and the explanations in Section 3.1 provide further details. This example uses the input data from Figure 2 and we show the resulting 1D scalar field in Figure 5.

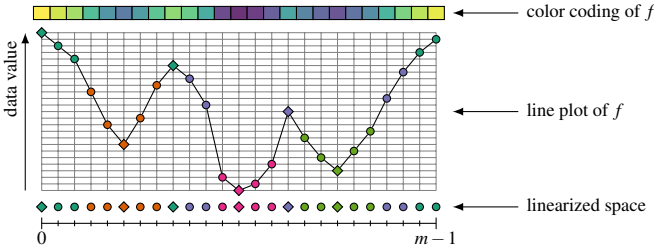


Figure 5. After the domain linearization (Figure 4), we assign the original data values to the samples. The resulting 1D scalar field keeps features intact and contains all original data samples for data context. Compare to the original data in Figure 2.

choose the starting side randomly, without affecting the property of representing the same merge tree.

During the traversal, we keep our range $[x_L, x_R]$ updated (see functions `ProcSuperArc()` and `PlaceNode()` in Algorithm 1). The procedure ends upon reaching the next supernode.

Entering Recursion at a Supernode All inner[1] supernodes of M have two child superarcs. Hence, we have to place three entities in $[x_L, x_R]$: the two child superarcs and the supernode itself. We need to place the supernode between its two child superarcs to maintain the same topology as in the input data. To see this, recall that the child superarcs represent connected components of super- or sub-level sets that merge or split at the supernode. This can only happen if the supernode is adjacent to both of its children, i.e., between them. Assuming the child to be traversed first has a total of n nodes, the supernode will be placed at $x_C = x_L + n$. The child superarcs are then processed recursively: The first child fills the range $[x_L, x_C - 1]$, and the second child fills the range $[x_C + 1, x_R]$. Figure 4(b) illustrates this.

The order of the child superarcs plays an interesting role. Importantly, either order will create a function with the same merge tree as the original data. However, the flexibility in choosing which child to traverse first is what allows for optimizing the mappings between individual time steps with respect to the entire evolution of the data, see Section 3.2.

Ending Recursion at Leaf Nodes A leaf node terminates the recursion. At this point in the algorithm, only a single space at $x_L = x_R$ is left open for the leaf to be assigned to.

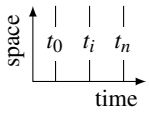
[1]The outer supernodes are the root and the leaves.

Output: 1D Scalar Field The mapping function $g(\mathbf{x}) = x$ assigns each original sample location a 1D location. Its inverse function $g^{-1}(x) = \mathbf{x}$ recovers the original sample location. The output scalar field $f : \{0, \ldots, m-1\} \to \mathbb{R}$ is simply given by $f(x) = s(g^{-1}(x))$, i.e., we assign each mapped position its original data value. We note that $f(x) = f(g^{-1}(x))$ holds, meaning f is a valid linearization of itself.

We can visualize f using a straightforward line plot, or apply color coding to obtain a line of colored pixels, see Figure 5. We will use the latter to assemble many time steps into a static 2D picture in the following.

3.2 Mapping all Time Steps: $\mathbb{R}^{d+1} \to \mathbb{R}^2$

After compressing the spatial dimensions to 1 through linearization, we can use the second dimension of our display to show time. For example, we can orient the time steps vertically and arrange them from left to right. This provides great insight into the dynamics of the data.



Features moving in a scalar field are inherently spatially and temporally coherent, i.e., their value and position changes are smooth. This is true for essentially all natural phenomena such as fluid flows, molecular dynamics, the weather and the climate, and so on. When converting these phenomena to data, the spatial and temporal sampling resolutions play a role: features get only represented appropriately if the Nyquist frequency is respected.

Any dimensionality reduction of the spatial domain restricts the spatial movement of features. This is also the case for our linearization. Furthermore, the following aspects may vary between consecutive time steps: the number and size of features, the hierarchical structure and depth of the merge trees, and features may merge or split. Despite these issues, our goal is to portray the dynamics of the data as accurately as possible. This problem statement is partly related to laying out tracking graphs [18, 45], or any graphs [3], in the plane. Note that we explicitly refrain from using tracking information for the merge trees, since this incurs very high computational costs [23].

Luckily, our linearization algorithm from Section 3.1 gives us some freedom in how to arrange the features such that we can maximize spatio-temporal feature coherence: when encountering an inner supernode, we can decide the order in which we are processing its two child superarcs. Figure 6 illustrates this with a data set containing two subtree regions a and b in t_i as child superarcs of the root. The regions slightly move and change their geometry between the time steps t_i and t_{i+1}. Let us denote them as a' and b' in t_{i+1}. Without loss of generality, we can fix the traversal of the tree at t_i such that a comes before b. Now we have two options for traversing the tree at t_{i+1}: either b' before a' (mapping 1), or the other way around (mapping 2). It is crucial to

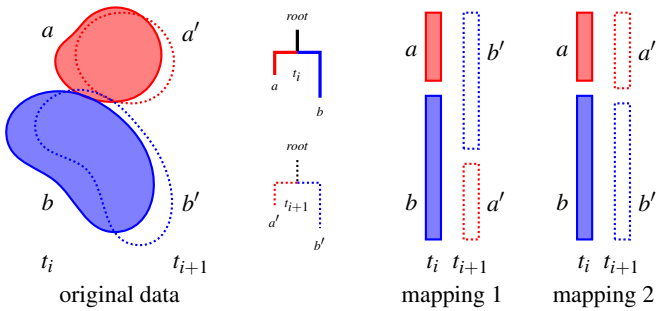


Figure 6. The traversal order during linearization of two consecutive time steps is chosen such that the overlap in the linearized domain resembles the overlap in the original data. In this example, mapping 2 is preferred over mapping 1.

understand that we do not actually know the "names" of these regions; each time step has its own merge tree and is linearized independently. However, we do know the amount of overlap that we get with each mapping in relation to the overlap of the regions in the original data. In the original data, the regions a and a' as well as b and b' overlap substantially. For the linearized data, mapping 2 is the preferred option, since its overlap resembles the overlap in the original data much better than mapping 1.

3.2.1 Discrete Optimization Problem

Let $S \in M_t$ and $T \in M_{t+1}$ be subtrees in the merge trees of two consecutive time steps. We measure their overlap (cf. Silver and Wang [35])

$$p(S,T) = |S \cap T| \tag{1}$$

as the number of nodes (sample points in the data) belonging to both subtrees. We note that this is computable in the original and in the linearized domain independently, and we denote these measures as p^{nD} and p^{1D}, respectively.

We seek to capture the *difference* in subtree overlap

$$d(S,T) = \left(p^{nD}(S,T) - p^{1D}(S,T)\right)^2 \tag{2}$$

between the original data and our mapping. Taking all time steps and all subtrees into account, we measure the sum of overlap differences over all pairs of subtrees in all pairs of consecutive time steps:

$$E = \sum_{t=t_0}^{t_{n-2}} \sum_{S_j \in M_t} \sum_{T_k \in M_{t+1}} d(S_j, T_k)\ . \tag{3}$$

This brings us to a discrete optimization problem where E is the objective function to be minimized, and the search space is formed by all possible merge tree traversal orders.

3.2.2 Heuristic Approach to the Discrete Optimization Problem

Traversal orders are determined by a series of binary decisions: At each inner supernode, we have to decide which child superarc to process first. This means our search space contains a total of 2^N possible configurations, where N is the total number of inner supernodes over all time steps. Such a large search space is impossible to fully enumerate even for moderately sized data sets.

Local, greedy heuristics give great results for many complex discrete optimization problems. We propose such a heuristic for the above optimization problem. In particular, we exploit the fact that mismatches higher up in a merge tree cause a larger error compared to lower levels just due to their larger size. Intuitively, a misalignment early in the traversal cannot be rectified by changing traversal decisions in the lower parts of the tree.

We propose the following procedure: We regard the traversal order for one initial merge tree M_t as fixed, e.g., using the order in which the arcs are stored in the data structure. Any time step could be chosen here. The traversal of the next merge tree M_{t+1} is then optimized by starting at its root and establishing a traversal order at each inner supernode. To do so, we compute the contribution of the two children to the overall objective from Equation (3) for both possible traversal orders. The traversal order with the smaller contribution wins. We continue with the inner supernodes at the deeper levels of the tree.

After determining the traversal order for M_{t+1}, we continue with M_{t+2} and so on. Similarly, we can go back in time from M_t to M_{t-1} until the merge trees in all time steps have a fixed traversal order.

An implementation of this benefits from the following observations. First, we can pre-compute subtree overlaps p^{nD} in the original scalar field by iterating over the data once. Second, as we navigate the search space of traversal orders, overlaps in the linearized domain have to be calculated frequently. We can utilize interval arithmetic to compute this directly without iterating over the data. For this, we use the minimum and maximum position assigned to each subtree and calculate the amount of overlap between S_j and T_k as the difference between the smaller maximum and the larger minimum.

$$\begin{aligned} \min_{x_{max}} &= \min(x_{max}(S_j), x_{max}(T_k)) \\ \max_{x_{min}} &= \max(x_{min}(S_j), x_{min}(T_k)) \\ p^{1D}(S_j,T_k) &= \max(0, \min_{x_{max}} - \max_{x_{min}} + 1) \end{aligned} \tag{4}$$

Lastly, note that the subtree below the root, i.e., the entire tree, can be omitted from these computations. Any other subtree always overlaps with the full domain in its entirety. This naturally holds for both original and linearized data, and the difference makes no additional contribution to the objective function.

3.3 Assembling the Final Image: Pixel-Based Visualization

Our final visualization consists of a 2D image representing linearized space in one direction (usually vertical), and time in the other direction (usually horizontal). We use color coding to show the corresponding data values and call this image the *temporal merge tree map*. See the bottom of Figure 3 for an illustration.

If we were to represent every sample point of the original data as a single pixel, the final resolution of the image would be $m \times n$, with m being the number of data points in a time step, and n being the number of time steps. However, m often significantly exceeds the dimensions of a display or a GPU texture size limit, while n is often below those numbers. Along the dimension of linearized space, we use subsampling to reduce the number of pixels to below the GPU texture size limit, which is 4096 on our hardware. Along the dimension of time, if necessary, we use linear interpolation to fill the display.

4 Evaluation and Discussion

We start our evaluation by introducing all data sets used in this paper and providing the respective running times of our algorithm (Section 4.1). We then analyze the quantitative and qualitative performance of the optimization scheme (Section 4.2). We compare our algorithm with other linearization methods (Section 4.3) and with feature tracking methods (Section 4.4). Finally, we discuss the limitations of our algorithm regarding data size and complexity (Section 4.5).

4.1 Data Sets and Runtime Performance

We use the following data sets in this paper:

Nucleon Padded slice from the nucleon data set used for evaluation by Zhou et al. [46]. The data set is publicly available in the *Open Scientific Visualization Datasets* collection [16] and courtesy of SFB 382 of the German Research Council (DFG).

Ring Analytical data set created and used for evaluation by Franke et al. [11]. Parametrized by peak value and standard deviation to generate a Gaussian bell-curve along a circle given by center position and radius. All parameters except the center position vary linearly between start and end.

Table 1. Overview of our data sets with their dimensions ($x \times y \times z$), time steps (n), persistence threshold p in % of the data range, and number of supernodes in the simplified merge trees (# snodes). The runtimes for the pre-processing (extracting and simplifying the merge trees with TTK [39]) are summarized as t_{topo}. The runtimes for our method are given in detail for each individual stage: the optimization (t_{opt}), the linearization (t_{map}), and the image creation (t_{img}). All timings are given in seconds and measured single-threaded unless indicated otherwise; times with * are from parallel runs with 18 threads. We use a workstation with two 18-core 2.3GHz Intel Xeon E5-2697 v4 processors and 256GB main memory. The shorter runtimes are obtained by averaging 10 runs.

data set	$x \times y \times z$	n	p	# snodes	t_{topo}	our method			t_{total}
						t_{opt}	t_{map}	t_{img}	
Nucleon	64×64	1	–	28	0.10	–	0.01	–	0.1
Ring	14×14	40	–	4 - 14	0.28	0.01	0.03	0.09	0.4
Benzene	61×61	21	0.05	14 - 68	0.73	0.02	0.02	0.23	1.0
Storms	282×181	744	0.015	15 - 79	67.71	0.50	2.24	1.22	71.7
Cylinder	$135 \times 64 \times 48$	508	2	2 - 375	641.32	7.13	14.00	4.20	666.7
Tangaroa	$300 \times 180 \times 120$	201	0.15	418 - 2811	20min*	10.4h	56.50*	15.97	10.7h
			0.5	392 - 1974	96min	147min	115.37	14.96	150min
			2	322 - 794	94min	9min	116.87	13.02	104min
			10	50 - 186	101min	15.67	123.53	14.13	103min

Benzene The electrostatic field around a benzene molecule was calculated using the fractional charges method described in [36]. The gradient of this field describes the force upon a positive point charge given in a certain location.

Storms 1-hourly mean sea level pressure anomaly for December 1999. Instantaneous data is obtained from the ERA5 reanalysis data set available at the *Copernicus Climate Change Service (C3S) Climate Data Store* [14]. We subtract the mean over 8 days following the data processing procedure used for detection of cyclones by Deroche et al. [9] and apply a light Gaussian smoothing to the result.

Cylinder The flow behind a square cylinder has been obtained from a direct numerical Navier Stokes simulation by Camarri et al. [4]. A uniformly resampled version of this flow from von Funck et al. [42] has been used to compute the Okubo-Weiss criterion $Q = 1/2(\|\Omega\|^2 - \|\mathbf{S}\|^2)$, which is a time-dependent scalar field indicating vortex activity for regions with $Q > 0$ and used in this paper.

Tangaroa The flow behind a model of the research vessel *Tangaroa* has been simulated with the Gerris flow solver [28] by Popinet et al. [29]. We use the velocity magnitude of a resampled version available from the visualization data set collection at the ETH Computer Graphics Laboratory [13].

Resolutions, simplification thresholds, topological complexity, and runtimes are given in Table 1. We note that the majority of the runtime is spent in TTK [39] on the extraction and simplification of the merge trees, which can be sped up significantly by parallelizing over the time steps as we have done for the *Tangaroa* data set. We obtain a speed-up factor of 8 by running with 18 parallel threads.

4.2 Analysis of the Objective Function

To gain insight into our optimization scheme, we choose a non-trivial data set for which we can expect a certain output. The *Benzene* data set represents the electrostatic potential around the benzene molecule and exhibits the well-known 6-fold symmetry in the xy-plane. It is a static 3D scalar field and we choose the z-dimension to slice through the data over "time," making this a 2D time-dependent field. Five representative z-slices show the setup in Figure 7. At $z = 10$ we see the strongest values for the electrostatic potential, since this slice cuts directly through the molecule itself. With increasing distance from $z = 10$, the field becomes weaker in both directions, i.e., we have a "temporal" symmetry besides the spatial 6-fold symmetry.

We used the join tree to analyze this data, i.e., the leaves of the join tree are the minima of the electrostatic potential.

The temporal merge tree map in Figure 7 shows the optimized output of our algorithm and reveals both temporal and spatial symmetries very nicely. The spatial 6-fold symmetry is easily discernible in slices $z \leq 4$

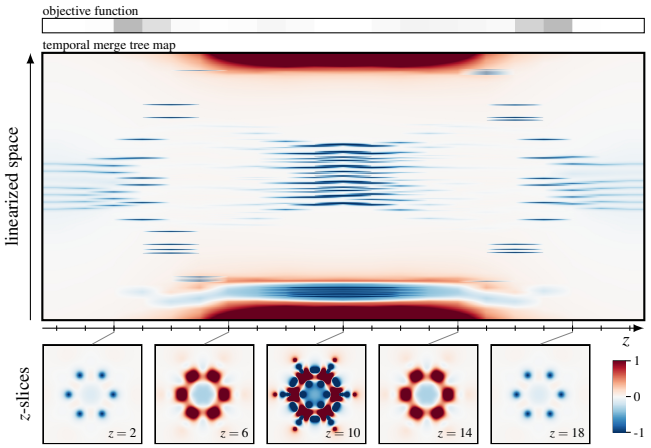


Figure 7. Temporal merge tree map for the electrostatic potential of the *Benzene* molecule when slicing through the z-dimension. The spatial 6-fold symmetry, as well as the temporal symmetry, are well captured. Compare this optimized version with the other ones in Figure 9.

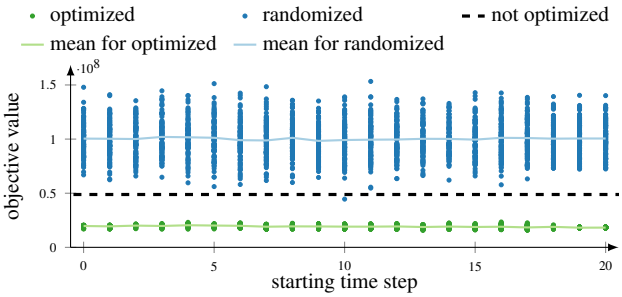


Figure 8. Our optimization scheme results in consistently better objective values, compared with not optimizing or using completely random traversal orders. This holds irrespective of the time step at which we initialize the optimization. See the text for more details. This experiment has been conducted with the *Benzene* data set. The visual result for the best optimized objective is shown in Figure 7, and the non-optimized and worst randomized results are shown in Figure 9.

where six horizontal blue lines represent the minima that we also see as six distinct blue areas in the $z = 2$ slice. We see the same behavior in slices $z \geq 16$, which successfully reveals the temporal symmetry in the data. It also shows that our optimization scheme handles the near-identical data at either end of the z-dimension in a very consistent manner.

To quantitatively evaluate the optimization scheme, we set up the following experiment: for each time step, we start the optimization scheme 100 times; each run is initialized with a different starting condition by randomizing the traversal order of the merge tree in that time step. We recorded the objective function value E from Equation (3) for each run and plotted the results in green color in Figure 8. We compare this with two other conditions: the dashed line represents the objective function value for the unoptimized version, and the blue dots represent runs where we randomized the traversal order of *all* merge trees. As we can see, our optimization achieves significantly better objective function values in a consistent manner. This also translates to a higher visual quality: Figure 9, shows the temporal merge tree maps for the non-optimized and randomized versions, which exhibit less symmetry and more distortions than the optimized version in Figure 7.

4.3 Comparison with other Linearization Methods

Franke et al. [11] use space-filling curves to linearize their data for a spatio-temporal summary view. The layout is very much like ours: linearized space and time are the two dimensions of this view. Many different space-filling curves are evaluated for this purpose, particularly

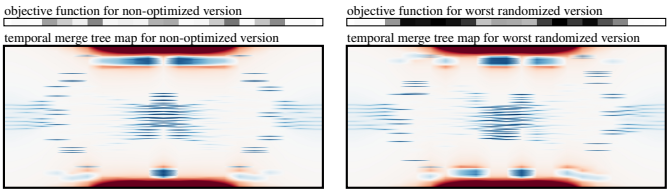


Figure 9. Non-optimized version (left) and worst randomized version (right) of the temporal merge tree map for the *Benzene* data set.

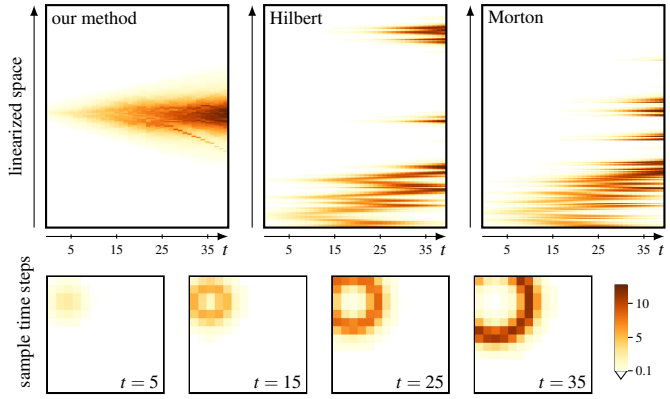


Figure 10. Result for our method and two different space-filling curves for the *Ring* data set by Franke et al. [11]. The data contains a single feature (the ring) growing in size and intensity. Our method captures this well, whereas the Hilbert and Morton curves do not keep the feature intact.

using the artificial *Ring* data set. We re-created the experiment in Figure 10 for the Hilbert and Morton curves using the code of the authors (see Figure 5a in Franke et al. [11]) and compare it to our result. This data set has a single feature, namely a ring becoming larger over time, but the space-filling curves fail to keep this feature intact and scatter its footprint over the spatial axis: the coherence of the original data is not communicated by the Hilbert and Morton space-filling curves. On the other hand, our method keeps the feature intact and its growth in intensity and footprint is easy to read from our result.

Visualization and graphics applications benefit from a form of "data awareness" for domain linearizations. Zhou et al. [46] propose data-driven space-filling curves aiming to minimize the similarity of data values and location coherency in a neighborhood. Dafner et al. [7] propose context-based space-filling curves to improve autocorrelation in 2D image and video encoding. We compare them with our method in Figure 11, where we essentially recreate Figure 6 from Zhou et al. [46] using the code of the authors. The lineplots for the linearizations in the top row show that the Hilbert and context-based curves do not retain feature coherence. At first glance, the straightforward scanline method seems to keep the feature intact, but this is circumstantial and on close inspection one can see individual spikes. The data-driven curve of Zhou et al. [46] manages to keep most larger data values close together, but also exhibits distinct separate peaks. However, the Nucleon data set consists of one large feature with two small maxima on top, which is faithfully reproduced by our method.

The bottom row of Figure 11 visualizes the traversal order for the different methods using a color coding scheme: blue points are visited first, yellow points are visited last. This reveals very interesting access patterns for the space-filling curves and it also highlights again that our linearization method does *not* use a space-filling curve: the linearization index jumps through the original data domain. Because of that, applications like image and video encoding are not likely to benefit from our method. Instead, we target applications where feature coherence is important such as the spatio-temporal summary views presented in this paper.

We provide additional comparisons using other data sets in the supplemental material.

4.4 Comparison with other Feature-based Methods

One of the goals of feature tracking methods [18, 22, 33, 45] is to reduce the amount of data as much as possible; hence, features are presented without their data context. We elaborate on this difference to our method in Figure 12 where we use 4 different functions that all exhibit the same data range $-0.3 \leq s(\mathbf{x}) \leq 1$, but with varying distributions and spatial patterns. Specifically, we compare our method to approaches that use sub/superlevel sets such as Nested Tracking Graphs of Lukasczyk et al. [22] or Temporal Treemaps of Köpp and Weinkauf [18]. We use $s_a = 0.06$ and $s_b = 0.9$ to extract superlevel sets in all four examples: the sets are identical in size, i.e., they cover the same area in the data. Hence, methods solely focusing on these features represent them indistinguishably. On the other hand, our method uses merge trees instead of fixed thresholds and maps all data samples to the linearized domain. Hence, the different data distributions (Figure 12(a) vs. Figure 12(b)) or the different number of features (Figure 12(a) vs. Figure 12(c)) can be distinguished with our method. However, our method is blind to the different spatial distribution of the data between Figures 12(a) and 12(d).

Our method can be parametrized to emulate a Nested Tracking Graph [22] by using a discrete colormap, i.e., a few discrete colors are distributed over the data range. They correspond to the fixed thresholds defining the layers of a Nested Tracking Graph. Figure 13 shows this for the *Storms* data set: both methods create a similar first-glance impression, but our method clearly shows a better temporal coherence. Methods for increasing temporal coherence of Nested Tracking Graphs exist [18], but their computationally more demanding optimization method is unable to deal with the complexity of this data. Our heuristic maintains a satisfactory temporal coherence with a computational effort of less than a second.

4.5 Discussion of Data Size and Topological Complexity

The following aspects play a role in how well our method works for a data set: the spatial size of the data, its temporal size, and its topological complexity. These aspects may also be intertwined, e.g., spatially large data sets tend to be topologically complex as well, but not necessarily.

All spatial dimensions are compressed into one and displayed vertically. As mentioned earlier (Section 3.3), this often leads to subsampling. Depending on the spatial size of the data and the available vertical space of the display, features with a small spatial footprint may get lost. It is an interesting avenue for future research to investigate adaptive sampling methods, which would need to be synchronized with the adjacent time steps.

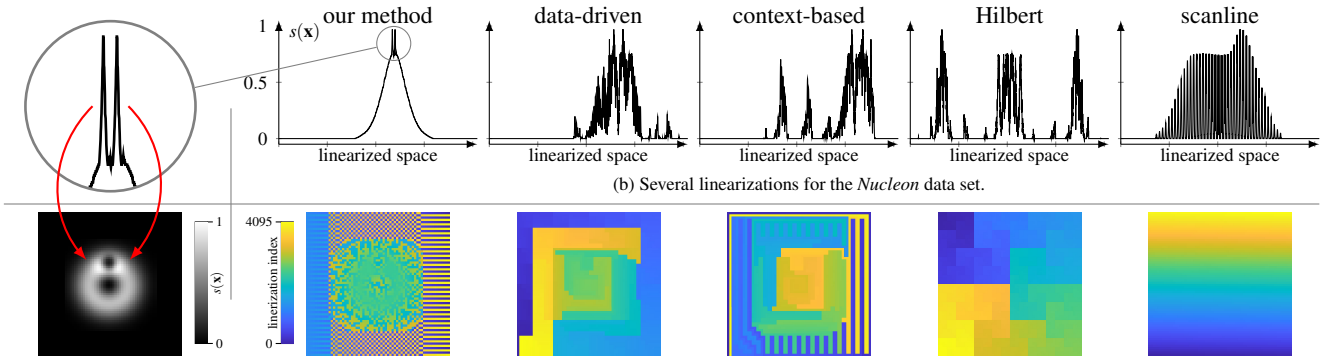
Temporally large data sets do not present a major issue for our method. Since time gets its dedicated dimension, we can accommodate many time steps before running out of horizontal space on a modern display. Simple *zooming & panning* may then mitigate the issue in most cases.

Topologically complex data sets pose a perceptual challenge for those feature-based visualization methods aiming to show features as distinguishable entities to the user: we can only distinguish a limited number of features in any given visualization. This applies to our method and the *Cylinder* data set (see Figure 14 and Section 5) is an example where the topological complexity is too high to reliably distinguish features in the later time steps of the simulation.

We study the effects of topological simplification using the *Tangaroa* data set. The results of four different simplification thresholds are shown in Figure 15: increased simplification leads to less noisy and more distinct structures in the output images, while the overall Gestalt of the data is retained. The unsimplified merge trees of this data set have up to 3655 supernodes, which leads to an unreasonably long running time for the optimization stage. We can see from Table 1 that topological simplification significantly reduces the running times.

5 Results

The *Storms* data set visualized in Figure 1 and Figure 13 is a time-dependent scalar field derived from atmospheric pressure, which gives us cyclones as low-pressure regions moving over time. It describes the

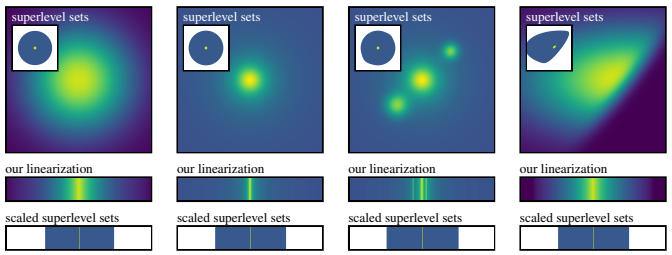


(a) *Nucleon* data set.

(b) Several linearizations for the *Nucleon* data set.

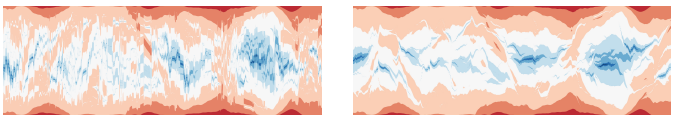
(c) Colormapped traversal orders.

Figure 11. Comparison of our method against several space-filling curves in terms of linearization and traversal order for the *Nucleon* data set inspired by a similar comparison in Zhou et al. [46]. The Nucleon data set consists of one large feature with two small maxima on top. The space-filling curves cannot keep these features intact, but our linearization can. As the traversal orders show, our method is *not* a space-filling curve, but jumps through the original domain.



(a) Normal distribution. (b) Delta sequence. (c) Three Features. (d) Skew normal distribution.

Figure 12. Some feature tracking methods use fixed thresholds to extract sub/superlevel sets and depict them in a tracking graph scaled by size. These four different scalar fields cannot be distinguished by those methods. Since our method uses merge trees to map *all* data samples, it can distinguish between different data distributions (a vs. b) or a different number of features (a vs. c). It is however blind to changes in the spatial distribution (a vs. d).



(a) Nested Tracking Graph with 9 layers.

(b) Temporal Merge Tree Map with a 9-step discrete colormap.

Figure 13. Using a discrete colormap with our method is similar to defining thresholds for Nested Tracking Graphs [22]. The results bear a resemblance, but our method shows better temporal coherence due to our explicit temporal optimization. Compare these results to our original result for the *Storms* data set from Figure 1.

storm activity over Europe for the entire month of December 1999 in 1-hourly intervals. This is our data set with the most time steps.

December 1999 was a disastrous month for Europe: cyclone *Lothar* killed 110 people and caused the highest storm damages in recent European history (11 billion Euro), overshadowing the other violent storms in that month such as *Anatol* and *Martin*.

Our temporal merge tree map in Figure 1 shows low pressure systems with blue colors. This reveals the individual storms as dark blue curves very well thanks to our feature-preserving linearization and the optimization of the temporal coherence. The temporal merge tree map facilitates data analysis tasks such as counting storms, comparing their life time or the size of their footprint, and so on.

We ask the reader to pay attention to December 26 where almost all of the visualization turns blue, indicating a dominance of low-pressure systems at that moment. Similarly, almost the entire map turns red for December 23. These fleeting moments in time are easy to miss in an animation, but are prominently revealed through our visualization. A mere feature tracking would also not be able to reveal these moments as the data context would be discarded.

The 3D flow around a *Cylinder* shown in Figure 14 exhibits periodic vortex shedding leading to the well known von Kármán vortex street. The simulation is initiated from an impulsive start-up and the periodic vortex shedding develops with time. This means, the flow becomes increasingly unsteady over time. This is our topologically most complex data set.

Our temporal merge tree map is able to reveal the different phases of the simulation. The fluid is at rest at the beginning of the simulation and a recirculation region slowly builds behind the cylinder. This is the startup phase in which no vortex shedding occurs. Once the recirculation region is large enough, vortices separate from it and are transported downstream. We only see primary vortex structures with almost 2D behavior in this second phase, i.e., their profile remains constant in spanwise direction and they are almost straight tubes parallel to the z-axis. The von Kármán vortex street is fully developed in the third phase. Primary and secondary vortex structures with varying profiles and geometries appear in this phase.

Note that we are transforming the 4D space-time domain of this data set to a 2D domain for the visualization. There is no such thing as a free lunch. This example reveals that our ability to see the evolution of features is closely correlated to the topological complexity of the data. The more superarcs we have in the merge trees, the harder it is to identify these individual regions, establish their temporal coherence, and follow them over time. The average amount of superarcs for the three phases is 14, 37, and 241, respectively. Nonetheless, having a static visualization of this data set is highly informative and a very good companion to a spatial volume rendering.

The 3D flow behind the research vessel *Tangaroa* shown in Figure 15 captures how a side-on airflow is affected by the vessels' geometry. This is of interest, as the several instruments mounted on the *Tangaroa* take meteorological measurements which can be influenced by airflow distortions. Our temporal merge tree map reveals one large-scale structure persisting throughout the entire time range. This is the structure right behind and around the vessel. Several structures of various sizes split of and eventually disappear. The largest of these is a feature group detaching from the main one around $t = 25$ and disappearing through leaving the domain around $t = 145$. Since our method provides a static overview of all time steps, we can confidently state that this process happens only once and is not akin to a periodic vortex shedding, but rather most likely caused by the initial flow conditions.

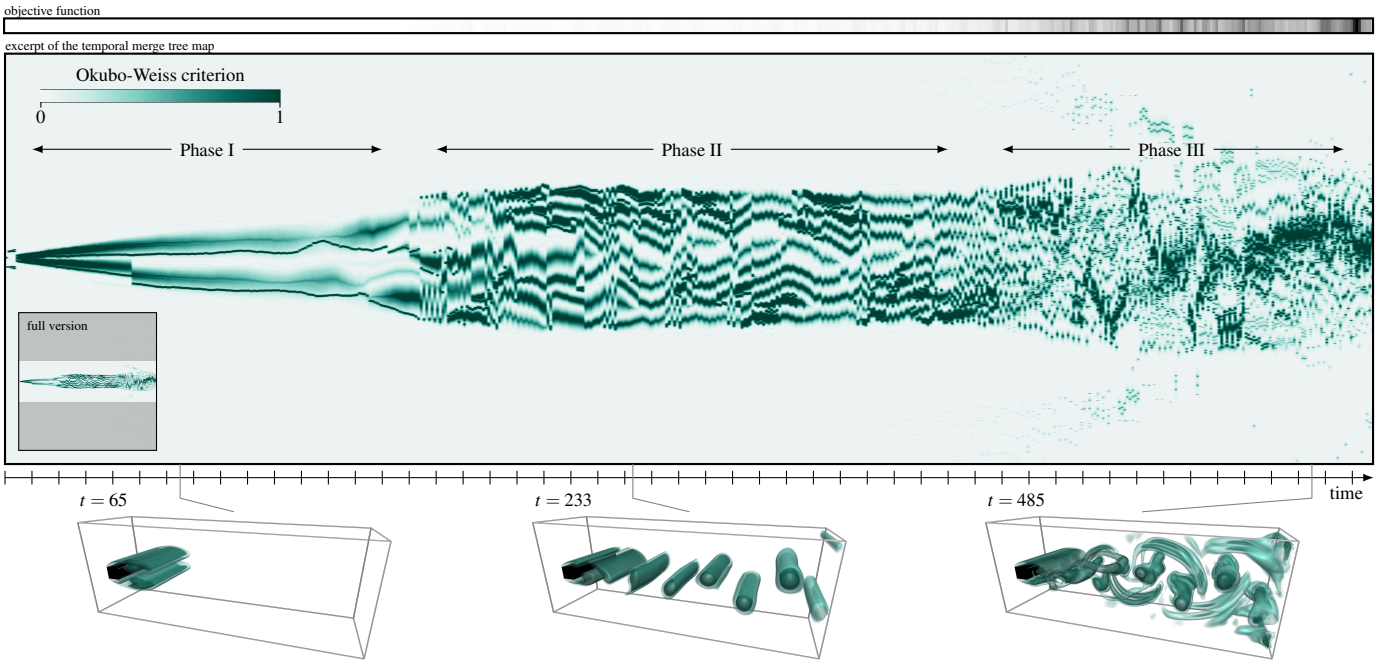
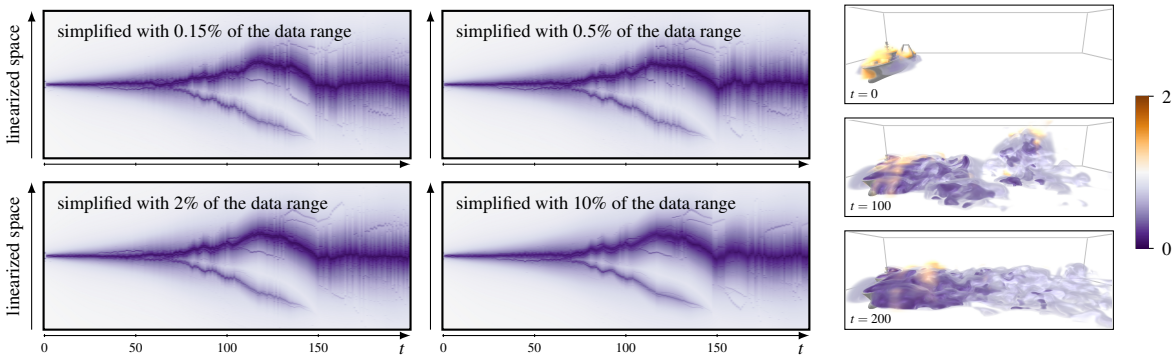


Figure 14. The temporal merge tree map for the *Cylinder* data set reveals three phases with increasing topological complexity: the startup phase where most of the fluid is still at rest, the initiation of the shedding with just primary vortex structures, and the fully developed von Kármán vortex street. The scalar field is the Okubo-Weiss criterion restricted to values $Q > 0.04$.



(a) Temporal merge tree maps simplified with different persistence thresholds.

(b) Volume renderings.

Figure 15. Topological simplification leads to a smoother temporal merge tree map for the *Tangaroa* data set, while retaining the overall Gestalt. Our method reveals two large feature groups splitting from each other at around $t = 25$, which can also be seen in the volume rendering at $t = 100$.

6 Conclusions and Future Work

We introduced *temporal merge tree maps* as a static visualization for time-dependent scalar fields. It is based on a feature-based domain linearization which allows us to compress all spatial dimensions into one, while keeping features intact and preserving the data context. We use this in a 2D layout where the temporal dimension is placed orthogonally to the linearized spatial dimension. We developed a scheme to optimally preserve temporal coherence. We compared our method to related work and applied it to several data sets.

While our results show astonishing temporal coherency compared to previous work and taking into account the dimensionality reduction, we can clearly see some temporal discontinuities in the final results. In some cases, there are simply big changes in the data between two time steps. A higher resolution of the data could help in those cases. In other cases, the number of superarcs spikes for some time steps. Currently, we apply topological simplification to each time step individually, which could be responsible for such spikes between time steps. We will leave it to future research to devise a global topological simplification scheme that incorporates all time steps at once. Similarly, it would be interesting to try out other heuristics for solving the discrete optimization problem and to study if the heuristic presented here could improve layouts in feature-based methods. Other avenues for future research include the usage of contour trees instead of merge trees, and context-aware subsampling [1] for the creation of the final image.

It is highly interesting to investigate how our concept can be applied to other kinds of data, e.g., vector fields or point clouds, such that we can get the same kind of spatio-temporal insight for that data as we have now for scalar fields with temporal merge tree maps.

Acknowledgments

The authors wish to thank the anonymous reviewers, Michael Ankele, and Jiahui Liu for valuable feedback. This work was supported through grants from the Swedish Foundation for Strategic Research (SSF, Project BD15-0082) and the Swedish e-Science Research Centre (SeRC). The presented concepts have been implemented in the Inviwo framework.

REFERENCES

[1] S. Avidan and A. Shamir. Seam Carving for Content-Aware Image Resizing. In *ACM SIGGRAPH 2007 Papers*, pp. 10–es, 2007. doi: 10.1145/1275808.1276390 9

[2] D. Bauer and R. Peikert. Vortex tracking in scale space. In *Data Visualization 2002. Proc. VisSym 02*, pp. 233–240, 2002. doi: 10.5555/509740.509779 2

[3] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*. Elsevier Science Publishing Co., Inc., New York, 1976. 4

[4] S. Camarri, M.-V. Salvetti, M. Buffoni, and A. Iollo. Simulation of the three-dimensional flow around a square cylinder between parallel walls at moderate reynolds numbers. In *XVII Congresso di Meccanica Teorica ed Applicata*, 2005. 6

[5] G. Cantor. Ein Beitrag zur Mannigfaltigkeitslehre. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1878(84):242–258, 1878. 3

[6] H. Carr. *Topological Manipulation of Isosurfaces*. phdthesis, The University of British Columbia, 2004. doi: 10.14288/1.0051287 2

[7] R. Dafner, D. Cohen-Or, and Y. Matias. Context-based Space Filling Curves. *Computer Graphics Forum*, 19(3):209–218, 2000. doi: 10.1111/1467-8659.00413 3, 7

[8] I. Demir, C. Dick, and R. Westermann. Multi-Charts for Comparative 3D Ensemble Visualization. *IEEE Transactions on Visualization & Computer Graphics*, 20(12):2694–2703, 2014. doi: 10.1109/TVCG.2014.2346448 3

[9] M.-S. Deroche, M. Choux, F. Codron, and P. Yiou. Three variables are better than one: detection of european winter windstorms causing important damages. *Natural Hazards and Earth System Sciences*, 14(4):981–993, 2014. doi: 10.5194/nhess-14-981-2014 6

[10] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990. doi: 10.1145/77635.77639 2

[11] M. Franke, H. Martin, S. Koch, and K. Kurzhals. Visual Analysis of Spatio-temporal Phenomena with 1D Projections. *Computer Graphics Forum (Proc. EuroVis)*, 40(3):335–347, 2021. doi: 10.1111/cgf.14311 2, 3, 5, 6, 7

[12] D. Guo and M. Gahegan. Spatial ordering and encoding for geographic data mining and visualization. *Journal of Intelligent Information Systems*, 27(3):243–266, 2006. doi: 10.1007/s10844-006-9952-8 3

[13] T. Günther. Visualization data, 2022. `https://cgl.ethz.ch/research/visualization/data.php`. 6

[14] H. Hersbach, B. Bell, P. Berrisford, G. Biavati, A. Horányi, J. Muñoz Sabater, J. Nicolas, C. Peubey, R. Radu, I. Rozum, D. Schepers, A. Simmons, C. Soci, D. Dee, and J.-N. Thépaut. ERA5 hourly data on single levels from 1979 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2018. Accessed on 10-Mar-2022. doi: 10.24381/cds.adbb2d47 6

[15] D. Hilbert. Ueber die stetige Abbildung einer Line auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891. doi: 10.1007/BF01199431 3

[16] P. Klacansky. Open SciVis Datasets, December 2017. `https://klacansky.com/open-scivis-datasets/`. 5

[17] J. Klemelä. Visualization of Multivariate Density Estimates With Level Set Trees. *Journal of Computational and Graphical Statistics*, 13(3):599–620, 2004. doi: 10.1198/106186004x2642 2

[18] W. Köpp and T. Weinkauf. Temporal Treemaps: Static Visualization of Evolving Trees. *IEEE Transactions on Visualization & Computer Graphics (Proc. IEEE VIS)*, 25(1):534–543, 2019. doi: 10.1109/TVCG.2018.2865265 2, 4, 7

[19] A.-P. Lohfink, F. Gartzky, F. Wetzels, L. Vollmer, and C. Garth. Time-Varying Fuzzy Contour Trees. In *2021 IEEE Visualization Conference (VIS)*, pp. 86–90, 2021. doi: 10.1109/VIS49827.2021.9623286 2

[20] A.-P. Lohfink, F. Wetzels, J. Lukasczyk, G. H. Weber, and C. Garth. Fuzzy contour trees: Alignment and joint layout of multiple contour trees. *Computer Graphics Forum (Proc. EuroVis)*, 39(3):343–355, 2020. doi: 10.1111/cgf.13985 2

[21] J. Lukasczyk, C. Garth, G. H. Weber, T. Biedert, R. Maciejewski, and H. Leitte. Dynamic Nested Tracking Graphs. *IEEE Transactions on Visualization & Computer Graphics (Proc. IEEE VIS)*, 26(1):249–258, 2020. doi: 10.1109/TVCG.2019.2934368 2

[22] J. Lukasczyk, G. Weber, R. Maciejewski, C. Garth, and H. Leitte. Nested tracking graphs. *Computer Graphics Forum (Proc. EuroVis)*, 36(3):643–667, 2017. doi: 10.1111/cgf.13164 2, 3, 7, 8

[23] P. Oesterling, C. Heine, G. H. Weber, D. Morozov, and G. Scheuermann. Computing and visualizing time-varying merge trees for high-dimensional data. In H. Carr, C. Garth, and T. Weinkauf, eds., *Topological Methods in Data Analysis and Visualization IV*, pp. 87–101. Springer International Publishing, 2017. doi: 10.1007/978-3-319-44684-4_5 2, 4

[24] P. Oesterling, C. Heine, G. H. Weber, and G. Scheuermann. Visualizing nD Point Clouds as Topological Landscape Profiles to Guide Local Data Analysis. *IEEE Transactions on Visualization & Computer Graphics*, 19(3):514–526, 2013. doi: 10.1109/TVCG.2012.120 2

[25] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, 1890. 3

[26] R. Peikert and M. Roth. The parallel vectors operator - a vector field visualization primitive. In *Proc. IEEE Visualization 99*, pp. 263–270, 1999. doi: 10.1109/VISUAL.1999.809896 2

[27] M. Pont, J. Vidal, J. Delon, and J. Tierny. Wasserstein distances, geodesics and barycenters of merge trees. *IEEE Transactions on Visualization & Computer Graphics*, 28(1):291–301, 2022. doi: 10.1109/tvcg.2021.3114839 2

[28] S. Popinet. Free computational fluid dynamics. *ClusterWorld*, 2(6), June 2004. 6

[29] S. Popinet, M. Smith, and C. Stevens. Experimental and Numerical Study of the Turbulence Characteristics of Airflow around a Research Vessel. *Journal of Atmospheric and Oceanic Technology*, 21(10):1575–1589, oct 2004. doi: 10.1175/1520-0426(2004)021<1575:eansot>2.0.co;2 6

[30] H. Saikia, H.-P. Seidel, and T. Weinkauf. Fast similarity search in scalar fields using merging histograms. In H. Carr, C. Garth, and T. Weinkauf, eds., *Topological Methods in Data Analysis and Visualization IV*, pp. 121–134. Springer International Publishing, 2017. doi: 10.1007/978-3-319-44684-4_7 2

[31] H. Saikia and T. Weinkauf. Global feature tracking and similarity estimation in time-dependent scalar fields. *Computer Graphics Forum*, 36(3):1–11, June 2017. doi: 10.1111/cgf.13163 2

[32] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 100(5):401–409, 1969. doi: 10.1109/T-C.1969.222678 3

[33] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *IEEE Computer*, 27(7):20–27, 1994. doi: 10.1109/2.299407 7

[34] B. Shneiderman. Tree Visualization with Tree-Maps: 2-d Space-Filling Approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992. doi: 10.1145/102377.115768 2

[35] D. Silver and X.Wang. Tracking and visualizing turbulent 3D features. *IEEE Transactions on Visualization & Computer Graphics*, 3(2):129–141, 1997. doi: 10.1109/2945.597796 2, 5

[36] D. Stalling and T. Steinke. Visualization of vector fields in quantum chemistry. Technical report, ZIB Preprint SC-96-01, 1996. 6

[37] H. Theisel, J. Sahner, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Extraction of Parallel Vector Surfaces in 3D Time-Dependent Fields and Application to Vortex Core Line Tracking. In *Proc. IEEE Visualization 2005*, pp. 631–638, 2005. doi: 10.1109/VISUAL.2005.1532851 2

[38] H. Theisel and H.-P. Seidel. Feature Flow Fields. In *Data Visualization 2003. Proc. VisSym 03*, pp. 141–148, 2003. doi: 10.5555/769922.769938 2

[39] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. *IEEE Transactions on Visualization & Computer Graphics*, 24(1):832–842, 2018. doi: 10.1109/TVCG.2017.2743938 3, 6

[40] S. Volke, M. Middendorf, M. Hlawitschka, J. Kasten, D. Zeckzer, and G. Scheuermann. dPSO-Vis: Topology-based Visualization of Discrete Particle Swarm Optimization. *Computer Graphics Forum*, 32(3pt3):351–360, 2013. doi: 10.1111/cgf.12122 2

[41] S. Volke, D. Zeckzer, M. Middendorf, and S. Gerik. Visualizing Topological Properties of the Search Landscape of Combinatorial Optimization Problems. In *Topological Methods in Data Analysis and Visualization IV*, pp. 69–85. Springer International Publishing, 2017. doi: 10.1007/978-3-319-44684-4_4 2

[42] W. von Funck, T. Weinkauf, H. Theisel, and H.-P. Seidel. Smoke Surfaces: An Interactive Flow Visualization Technique Inspired by Real-World Flow Experiments. *IEEE Transactions on Visualization & Computer Graphics*, 14(6):1396–1403, 2008. doi: 10.1109/TVCG.2008.163 6

[43] G. Weber, P.-T. Bremer, and V. Pascucci. Topological Landscapes: A Terrain Metaphor for Scientific Data. *IEEE Transactions on Visualization & Computer Graphics*, 13(6):1416–1423, 2007. doi: 10.1109/TVCG.2007.70601 2

[44] J. Weissenböck, B. Fröhler, E. Gröller, J. Kastner, and C. Heinzl. Dynamic Volume Lines: Visual Comparison of 3D Volumes through Space-filling Curves. *IEEE Transactions on Visualization & Computer Graphics (Proc. IEEE VIS)*, 25(1):1040–1049, 2019. doi: 10.1109/TVCG.2018.2864510 3

[45] W. Widanagamaachchi, C. Christensen, P.-T. Bremer, and V. Pascucci. Interactive Exploration of Large-Scale Time-Varying Data Using Dynamic Tracking Graphs. In R. S. Barga, H. Pfister, and D. H. Rogers, eds., *2012 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 9–17, 2012. doi: 10.1109/LDAV.2012.6378962 2, 4, 7

[46] L. Zhou, C. R. Johnson, and D. Weiskopf. Data-Driven Space-Filling Curves. *IEEE Transactions on Visualization & Computer Graphics (Proc. IEEE Vis)*, 27(2):1591–1600, 2020. doi: 10.1109/TVCG.2020.3030473 3, 5, 7, 8