Dynamic Nested Tracking Graphs

Jonas Lukasczyk, *Member, IEEE*, Christoph Garth, *Member, IEEE*, Gunther H. Weber, *Member, IEEE*, Tim Biedert, Ross Maciejewski, *Member, IEEE*, and Heike Leitte, *Member, IEEE*

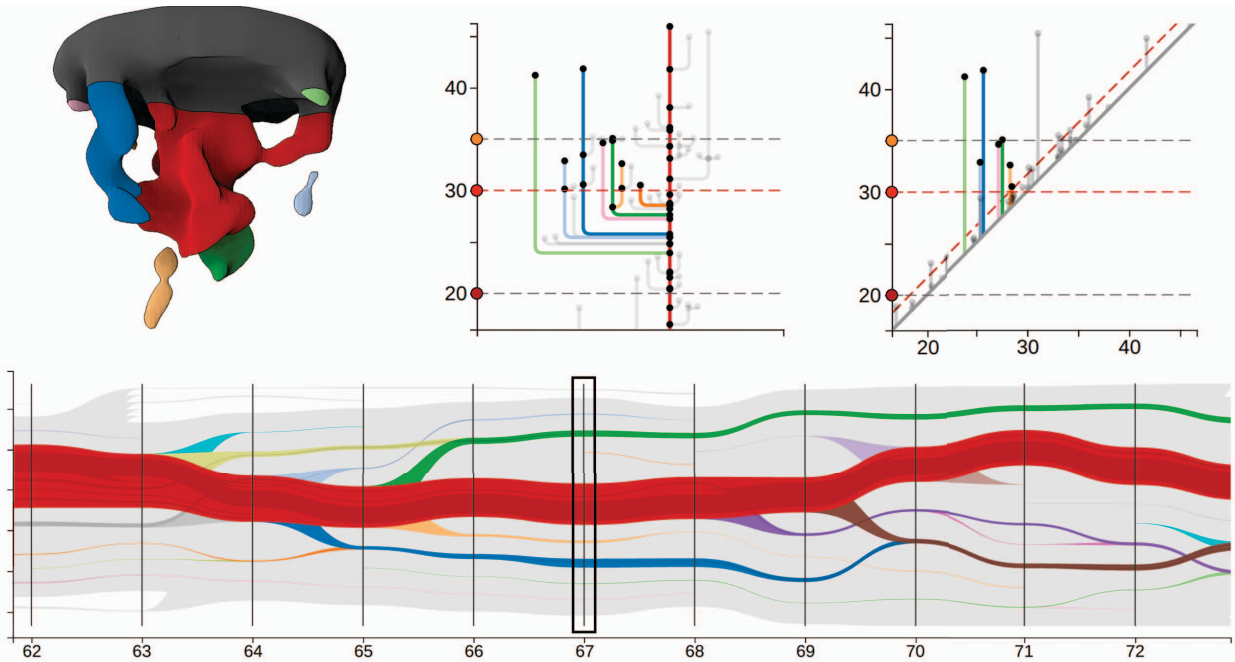

Fig. 1. The topology-based visual analytics framework supports the feature-centered navigation of Cinema databases consisting of image and analysis products generated during large-scale simulation runs, where numerous features are organized into a manageable amount of hierarchical groups that can be explored in a level-of-detail approach. Here, the interface shows an ensemble member of the viscous finger dataset, where colors encode individual fingers for salt concentration level 30. The prime interaction device of the framework is a nested tracking graph (NTG) that simultaneously displays the temporal evolution of superlevel set components for multiple levels (bottom). The NTG is used to navigate through time and retrieve component images from the database (top left), whereas the split tree (top center) and persistence diagram (top right) support the user in selecting important levels and filter criteria.

Abstract— This work describes an approach for the interactive visual analysis of large-scale simulations, where numerous superlevel set components and their evolution are of primary interest. The approach first derives, at simulation runtime, a specialized Cinema database that consists of images of component groups, and topological abstractions. This database is processed by a novel graph operation-based nested tracking graph algorithm (GO-NTG) that dynamically computes NTGs for component groups based on size, overlap, persistence, and level thresholds. The resulting NTGs are in turn used in a feature-centered visual analytics framework to query specific database elements and update feature parameters, facilitating flexible post hoc analysis.

Index Terms—Topological Data Analysis, Nested Tracking Graphs, Image Databases, Feature Tracking, Post Hoc Visual Analytics

1 Introduction

In many applications, interesting features can be characterized via superlevel set components—i.e., connected areas within scalar fields that exceed a given threshold (level). Examples include highly turbulent regions in flow fields (vortices), areas in combustion simulations above a fuel consumption rate threshold (burning regions), and parts of the universe exceeding a certain dark matter density (halos). Large-scale simulations that model such physical processes pose additional challenges to the already complex task of feature identification, tracking, and visualization. Specifically, it is often infeasible to write every simulation state to disk due to bandwidth and disk space constrains. These limitations necessitate *in situ* algorithms that store the least amount of information needed to still support flexible *post hoc* analysis; including the capability to select, filter, track, and render features. Additionally, visual analytic interfaces for massive amounts of features require level-of-detail techniques that intelligently organize features in groups.

This work describes an approach that addresses these issues by combining and extending so-called Cinema databases [2] (for *in situ* database generation) and nested tracking graphs (NTGs) [19] (for *post hoc* database exploration). A NTG consists of layers of common tracking graphs, where the branches (tracks) of a layer visualize the evolution of individual superlevel set components for a fixed level, and branches of different layers are drawn inside each other based on the nesting hierarchy of the components (Fig. 1). The presented approach is based

- *Jonas Lukasczyk, Christoph Garth, and Heike Leitte are with Technische Universität Kaiserslautern. E-mail: {lukasczyk,garth,leitte}@cs.uni-kl.de.*
- *Gunther Weber is with Lawrence Berkeley National Laboratory and University of California, Davis. E-mail: ghweber@lbl.gov.*
- *Tim Biedert is with NVIDIA Corporation. E-mail: tbiedert@nvidia.com.*
- *Ross Maciejewski is with Arizona State University. E-mail: rmacieje@asu.edu.*

Manuscript received 31 Mar. 2019; accepted 1 Aug. 2019.
Date of publication 16 Aug. 2019; date of current version 20 Oct. 2019.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2019.2934368

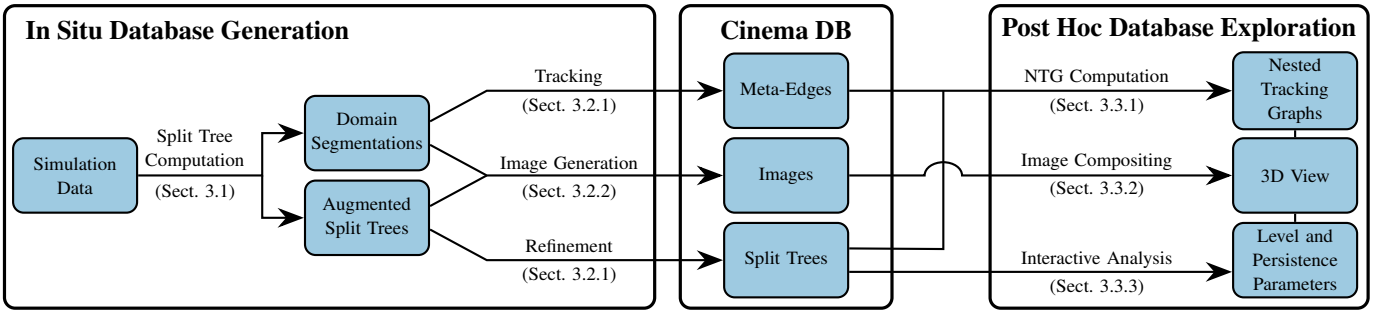

Fig. 2. Processing pipeline of the presented approach that consisits of the *in situ* database generation and the *post hoc* database exploration. During simulation runtime, the approach derives for each timestep the split tree and its associated domain segmentation to compute tracking information, images of feature groups, and refined split trees, which are stored in a Cinema database. During *post hoc* analysis, the database elements are used to dynamically compute nested tracking graphs, composite 3D views of feature groups, and to visualize the split trees and their corresponding persistence diagrams, which are all in turn integrated in a feature-centered visual analytics framework to effectively explore the underlying simulation.

on the fact that superlevel set components merge as the level decreases, and therefore lower-level branches of the NTG automatically bundle higher-level branches. Thus, NTGs can be used to control the maximum number of visible branch bundles that represent feature groups. This hierarchical decomposition is also used during database generation to store images of these groups instead of individual components, which reduces the amount of stored information while still supporting flexible *post hoc* analysis via linking and brushing. This work also describes a novel graph operation-based NTG algorithm (GO-NTG) that efficiently computes NTGs *post hoc* based on topological abstractions that are derived and stored at simulation runtime. Combining these contributions yields a scalable methodology for effective *post hoc* analysis of simulations containing numerous features. To summarize, the contributions of this work are:

- A graph operation-based NTG algorithm (GO-NTG);
- The topology-based generation and navigation of Cinema databases;
- A scalable visual analytics framework that enables the post hoc analysis of large-scale simulations with numerous features; and
- The implementation of all algorithms in the Topology ToolKit [32].

2 Related Work

This work primarily builds on the nested tracking graph (NTG) [19], a topological abstraction that records the evolution of superlevel set components across time and levels. Each layer of the NTG is a common tracking graph [5, 26, 28, 29, 34] that effectively illustrates the evolution of superlevel set components for a single level, where edges of different layers are drawn inside each other based on the nesting hierarchy of their associated components. However, the original algorithm proposed by Lukasczyk et al. [19] requires access to the entire raw simulation data, as it explicitly computes and tracks for each timestep and a predefined set of levels the corresponding component geometries, which does not scale for increasing simulation sizes due to IO constraints [2]. This work proposes a graph operation-based NTG algorithm (GO-NTG) that computes NTGs based on unaugmented merge trees [7] and meta edges [34], i.e., topological abstractions that are much smaller than the raw simulation data they are derived from. At simulation runtime, the proposed approach first computes the augmented merge tree of the current timestep with the task-parallel algorithm proposed by Gueunet et al. [13], then adds so-called meta edges [34] between vertices of the current and the previous merge tree that record the amount of overlap between the corresponding domain segments, and then stores the unaugmented merge tree together with the meta edges in a Cinema database (Fig. 2). Specifically, Sect. 3.2.1 describes an adaption of the original meta edges algorithm [34] that first refines the augmented merge tree into smaller scalar intervals to increase tracking accuracy, and then computes meta edges in parallel. The GO-NTG algorithm then processes these abstractions during *post hoc* analysis to compute NTGs, where the layout is derived with the algorithm of Lukasczyk et al. [19]. Recently, Köpp et al. [16] proposed an improved layout algorithm that could be used instead.

Several other authors have examined feature tracking based on topological abstractions. Oesterling et al. [20] proposed an algorithm to compute the time-varying merge tree by deriving a sequence of local updates that iteratively transform the trees over time. These updates are extremely expensive to compute, which makes them currently unsuitable for interactive systems and large datasets. Recently, Soler et al. [30] proposed to compute the newly introduced lifted Wasserstein distance between persistence diagrams to derive an optimal matching between critical point pairs. However, matched superlevel set components across different levels are not necessarily nested, which is why their method can not be trivially integrated into the proposed methodology. Yet, extending the method to ensure a nesting hierarchy appears fruitful, and can be addressed in future work. Bremer et al. [5] analyzed and tracked flame fronts in combustion simulations by computing a four-dimensional space-time Reeb graph that spans the entire data set in time, where edges in this graph correspond to evolving features. This allows them to flexibly change feature parameters without laborious recomputation. In later work, Bremer et al. [6] computed the overlap of static-threshold features based on a precomputed merge-tree segmentation that could be generated *in situ*. They combine this with presegmented data to build a visual analysis system.

The proposed methodology links edges of the NTGs to images of the corresponding component groups [8] that have been stored during the simulation in a so-called Cinema database [2]. Originally, these databases are structured image collections that enable the interactive *post hoc* visual analysis of extreme-scale simulations by simply browsing images that have been stored *in situ* for a fixed sampling of the parameter space. The current specification [25] supports any kind of data product; in particular depth images that can be used to composite 3D renderings of the scene *post hoc* [18]. Biedert et al. [4] also investigated the Cinema-inspired idea of combining *in situ* topological analysis and simplification with compact image-based storage in so-called contour tree depth images, which record at each pixel the list of depth values of individual contours from front to back. The proposed methodology improves on this idea by storing images of feature groups that are determined by a branch decomposition of the merge tree [22]. This makes it possible to use dynamic nested tracking graphs—whose edges are linked to the branch decompositions—as intelligent interaction devices to effectively navigate the massive databases.

3 Method

After introducing the necessary background (Sect. 3.1), this section describes the proposed approach outlined in Fig. 2 that derives a database at simulation runtime (Sect. 3.2) which supports flexible *post hoc* analysis (Sect. 3.3). This includes robust methodologies to identify and track superlevel set components (Sect. 3.2.1), presciently generate images of component groups (Sect. 3.2.2), efficiently compute NTGs in real-time (Sect. 3.3.1), compose images from the database (Sect. 3.3.2), and effectively explore the underlying simulations using the described algorithms in a visual analytics interface (Sect. 3.3.3).

3.1 Background

This section provides the formal background of nested tracking graphs, split tree segmentations, and persistence-based branch decompositions.

3.1.1 Nested Tracking Graphs

Common tracking graphs are one-dimensional simplicial complexes that effectively illustrate the evolution of superlevel set components for a fixed level. Their vertices represent individual components (white discs of Fig. 3, top), and edges connect vertices of adjacent timesteps iff their corresponding components are deemed related, either via spatial overlap or more advanced feature tracking methods (blue edges in Fig. 3, top). However, tracking graphs are limited to said level, and even slight variations can drastically change their structure, which makes it difficult to compare tracking graphs for different levels. Lukasczyk et al. [19] addressed these issues by proposing a topological abstraction called the nested tracking graph (NTG), which can be used to draw edges of different levels inside each other based on the nesting hierarchy of the components (red edges in Fig. 3, top). Formally, a NTG is a one-dimensional simplicial complex $\mathcal{N} = \mathcal{V} \cup \mathcal{E}_T \cup \mathcal{E}_N$ whose vertices $\mathcal{V}$ represent individual components for multiple timesteps and levels (white discs of Fig. 3, top), where the edges $\mathcal{E}_T$ record the tracking relationship between vertices of the same level at adjacent timesteps (blue edges in Fig. 3, top), and the edges $\mathcal{E}_N$ record the nesting hierarchy of components at the same timestep for adjacent levels (red edges in Fig. 3, top). The edges $\mathcal{E}_T$ and $\mathcal{E}_N$ are also referred to as tracking graphs and nesting trees, respectively. Thus, to compute a NTG, it is necessary to perform the following tasks:

- Task 1: Identify the superlevel set components that are present for a set of levels (the vertices $\mathcal{V}$).
- Task 2: Compute the nesting hierarchy of the components for adjacent levels (the nesting trees $\mathcal{E}_N$).
- Task 3: Determine the relationship between components for adjacent timesteps (the tracking graphs $\mathcal{E}_T$).

3.1.2 Split Tree Segmentations

This work describes a graph operation-based NTG algorithm that solves all tasks by processing intermediate graph structures that are computed, at simulation runtime, based on merge tree segmentations, or—more precisely—on split tree segmentations as the proposed method focuses on superlevel sets. Fig. 4 illustrates a split tree segmentation for one timestep, where scalar data $f : \mathcal{K} \to \mathbb{R}$ is given on the vertices of a simply connected simplicial complex $\mathcal{K}$, and values inside higher dimensional simplices are linearly interpolated. A split tree is then a one-dimensional simplicial complex $\mathcal{T}$ whose edges represent the evolution of individual superlevel set components during a positive level sweep [7, 13] (Fig. 4a). The tree also provides a domain partition $\phi : \mathcal{K} \to \mathcal{T}$ that maps any point of $\mathcal{K}$ to a vertex or edge of $\mathcal{T}$ (Fig. 4b), and a new scalar field $\psi : \mathcal{T} \to \mathbb{R}$ that assigns to each point of $\mathcal{T}$ the corresponding scalar value of f (y-axis of Fig. 4, right). To simplify notations, $\langle u, v \rangle \in \mathcal{T}$ denotes a split tree edge such that $\psi(u) < \psi(v)$, and the second vertex is called the edge representative as it can uniquely identify the edge. A crucial property of $\mathcal{T}$ and ψ—which is the basis for all following algorithms—is that each individual superlevel set component for a given level $l \in \mathbb{R}$ corresponds to a subtree of $\mathcal{T}$ where $\psi \geq l$, which is referred to as a crown (Fig. 4c).

3.1.3 Persistence-Based Branch Decompositions

The simplices of a split tree $\mathcal{T}$ can be grouped into branches $\mathcal{B}$ by first sorting all maxima by value in descending order, and then growing a new branch from each maximum towards the root until it either reaches the root, or another branch with a larger maximum (Fig. 4d right). Each resulting branch $B \in \mathcal{B}$ creates a so-called persistence pair consisting of the two endpoint vertices $u = argmin_{x \in B}(\psi(x))$ and $v = argmax_{x \in B}(\psi(x))$, where the corresponding value range $\psi(v) - \psi(u)$ is called the persistence of the branch that measures its significance. Branch decompositions are used in the proposed approach to further group branches into a fixed number of bundles by assigning less persistent branches to the most persistent branch they are attached to; for example, to depict components in groups (Fig. 4d, left).

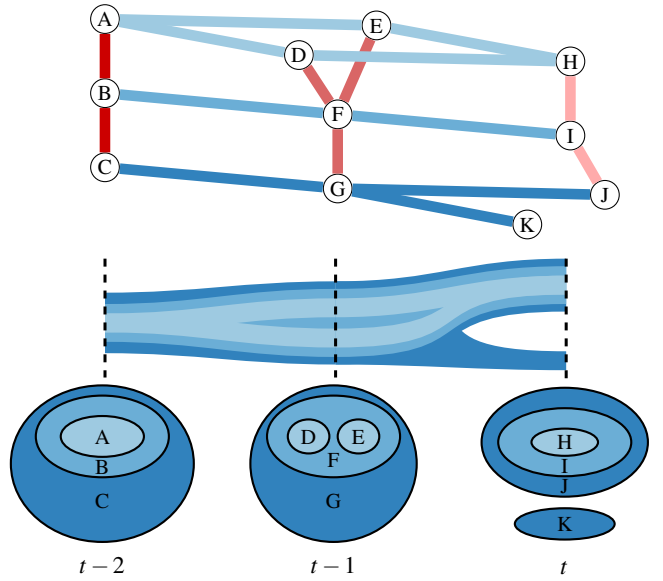

Fig. 3. (Bottom) Superlevel set components of a time-varying scalar-field for three levels (dark to light blue). (Top) 3D illustration of a NTG where components are represented by vertices, and tracking graphs for each level are shown in shades of blue, and nesting hierarchies for each timestep in shades of red. (Middle) 2D NTG layout where tracking graph edges are drawn inside each other based on the nesting hierarchies.

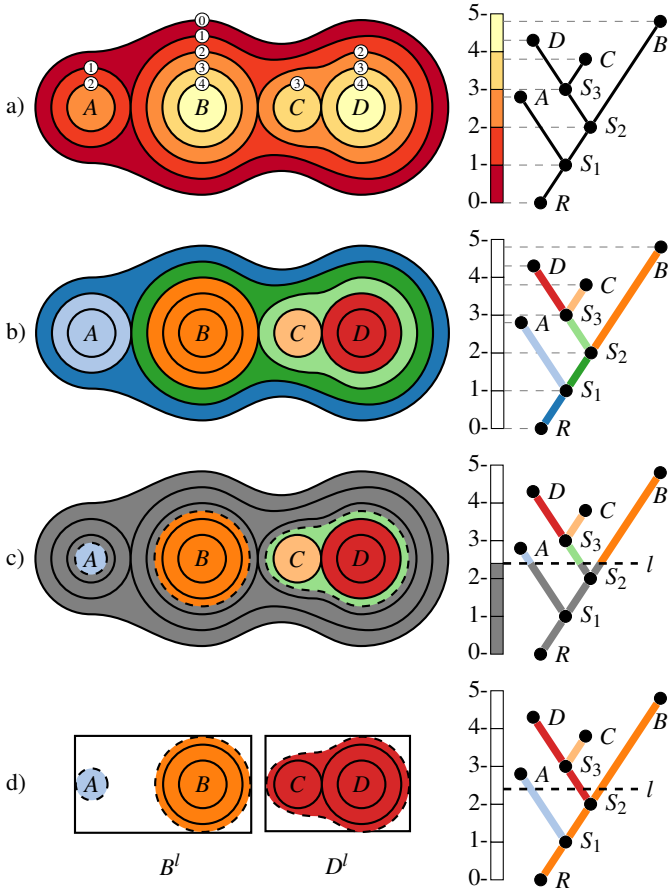

Fig. 4. Illustration of a split tree segmentation $\mathcal{S} = (\mathcal{T}, \phi, \psi)$ for a scalar field (a, left) consisting of a split tree $\mathcal{T}$ (a, right), its scalar field ψ (y-axis, right), and the domain segmentation ϕ (b). Each individual superlevel set component for a level l corresponds to one connected subtree of the split tree above the level threshold, and vice versa (c). A persistence-based branch decomposition of $\mathcal{T}$ can be used to further group branches into bundles whose components can be depicted in one image (d).

3.2 In Situ Database Generation

During the simulation, the proposed approach computes the complete split tree segmentation $\mathcal{S} = (\mathcal{T}, \phi, \psi)$ for every timestep with the task-parallel algorithm proposed by Gueunet et al. [13], which is implemented in the Topology ToolKit [32]. Optionally, the scalar field can be simplified by persistence to remove noise [10, 11, 33]. Next, the unaugmented split tree $\mathcal{T}$ and its scalar function ψ are directly stored in a Cinema database to later solve Task 1 and 2 of the NTG computation, whereas $\mathcal{S}$ is used to derive *in situ* so-called meta edges between unaugmented merge trees of adjacent timesteps (Sect. 3.2.1) to efficiently solve Task 3 *post hoc*. $\mathcal{S}$ is also used to store, at simulation runtime, a reduced set of images of component groups (Fig. 4d) that can later be composed again into 3D scenes (Sect. 3.2.2).

3.2.1 Split Tree Segmentation-Based Tracking

To compute tracking graphs, it is necessary to solve the correspondence problem between superlevel set components of adjacent timesteps. During *post hoc* analysis, it is no longer possible to explicitly compute component geometries and test them for overlap as the raw simulation data is no longer available. *In situ* algorithms address this problem by precomputing tracking information that can later be used to derive tracking graphs without reprocessing the original data [6, 20, 34]. A prime example of such an approach are so-called meta edges [34] that record the overlap of component groups for discrete level intervals. Alg. 1 is an adaption of this approach that processes two split tree segmentations that are defined on the same simplicial complex $\mathcal{K}$. Recall, each segmentation partitions the domain into connected regions, called segments, that correspond to individual split tree edges (Fig. 4b), and each superlevel set component is completely contained in the domain segments of its crown (Fig. 4c). Alg. 1 uses these facts to collectively track components based on the segments they are contained in. As a direct consequence, the accuracy of the tracking depends on the level intervals of the split tree edges—i.e., the granularity of the segmentation—where the accuracy can be increased by introducing regular vertices on edges to further subdivide segments (Fig. 5). If two segments of adjacent timesteps overlap at the same vertex v, then it is guaranteed that at least the components for the smallest level among both corresponding intervals—called the base level—intersect at v. Following this principle, Alg. 1 records the amount of spatial overlap between segments via meta edges $\mathcal{M}$ that connect the representatives of the corresponding split tree edges (all arrows of Fig. 5).

Specifically, Alg. 1 initializes the set of meta edges $\mathcal{M}$ as an empty set, and then iterates over each vertex v of the complex $\mathcal{K}$ to add edges to $\mathcal{M}$. Fig. 5 illustrates this process for one iteration, where each iteration first retrieves the edges of both split trees that correspond to v via the domain segmentation functions ϕ_{t-1} and ϕ_t (split tree edges with white discs in Fig. 5, bottom). As explained earlier, this overlap only guarantees that the superlevel sets for the base level of both intervals intersect at v. Therefore, the algorithm determines the base level b using ψ_{t-1} and ψ_t, and then traverses both trees towards the root until it finds the so-called base edges whose intervals include b (thin split tree edges in Fig. 5, bottom). Subsequently, the algorithm adds a meta edge to $\mathcal{M}$ between the representatives of the base edges. Furthermore, if components of these edges overlap, then so do the components of the edges towards the root. Thus, the algorithm also synchronously traverses both trees towards the root and adds meta edges to $\mathcal{M}$ between the representatives of the visited edges (red edges of Fig. 5). This procedure can additionally record the amount of spatial overlap between segments on the meta edges, and the size of segments on their respective split tree vertices. These properties are used during *post hoc* analysis to filter and scale NTG edges (Sect. 3.3.3). After all iterations, the set of meta edges $\mathcal{M}$ between the two segmentations is complete (all arrows of Fig. 5), and $\mathcal{M}$ is stored in the Cinema database.

This procedure is executed every time the simulation advances one timestep, and it is only necessary to keep the segmentation of the previous and current timestep in memory. Formalizing the algorithm in this fashion makes it also possible to execute iterations in parallel, and subsequently unify the generated meta edges. The stored meta edges are then processed *post hoc* by the GO-NTG algorithm (Sect. 3.3.1).

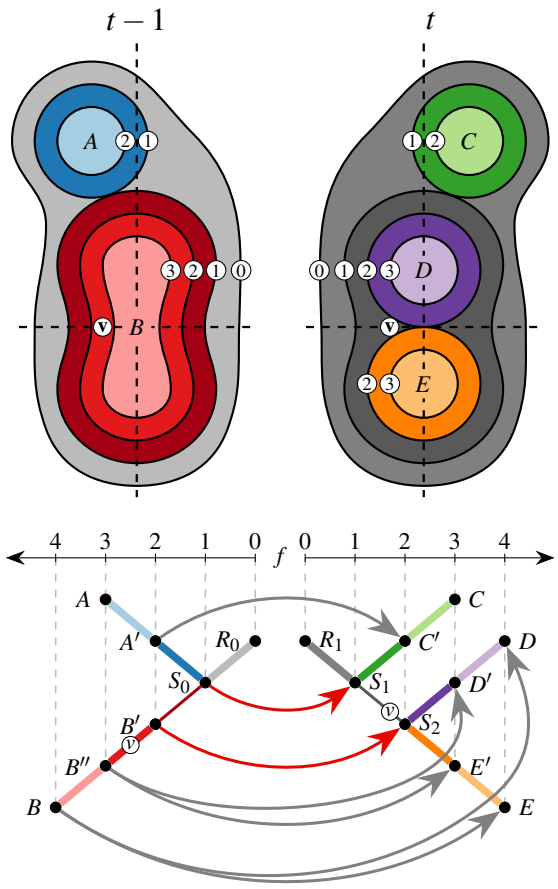

Fig. 5. Illustration of the segmentation-based tracking approach that processes two split trees (bottom) and their respective domain segmentations (top) of two adjacent timesteps (left and right). From the previous to the current timestep, the maximum B splits into the two maxima D and E, and the maximum A (later labeled C) moves from the left to the right side of the domain. The overlap of segments are recorded by so-called meta edges between their corresponding representatives (all arrows). For example, the dark blue and dark green segments overlap, which justifies the meta edge $\langle A', C' \rangle$. The light blue and the light green segments, however, do not overlap, and thus there exists no meta edge between A and C. The figure also highlights a single iteration of Alg. 1 for a vertex v of the domain. First, the algorithm retrieves for the segments in which v resides the corresponding edges of both split trees (split tree edges with white discs), then determines the corresponding base edges in both trees (thin split tree edges), and finally adds meta edges (red arrows) between the representatives of the base edges and all connected edges towards the root. Note, the complete set of meta edges (red and gray arrows) correctly record the overlap of segments across all intervals. Yet, the accuracy of the matching depends on the resolution of the intervals.

Algorithm 1: ComputeMetaEdges($\mathcal{K}$, ($\mathcal{T}_{t-1}$, ϕ_{t-1}, ψ_{t-1}), ($\mathcal{T}_t$,ϕ_t,ψ_t))

```
M ← ∅  // Set of Meta Edges
foreach vertex v ∈ K do
    // Get edges that correspond to segments
    (e_{t-1}, e_t) ← GetSegmentEdges( v, T_{t-1}, φ_{t-1}, T_t, φ_t )
    // Get edges that include base level
    b ← min( min ψ_{t-1}(e_{t-1}), min ψ_t(e_t) )
    (ē_{t-1}, ē_t) ← GetBaseEdges( b, T_{t-1}, ψ_{t-1}, e_{t-1}, T_t, ψ_t, e_t )
    // Connect all representatives towards the root
    AddMetaGraphEdges( M, ē_{t-1}, ē_t, T_{t-1}, ψ_{t-1}, T_t, ψ_t )
StoreInCinemaDB( M, t − 1 )
```

3.2.2 Image Generation

To provide an interactive 3D rendered view of the simulation *post hoc*, the proposed approach also stores, at simulation runtime, images of superlevel set component groups that can later be composed into 3D scenes. The following algorithm is built on top of the original Cinema approach [2] that generates images for a Cartesian product of the parameter space; e.g., images of contours for three isovalues at every fifth timestep from 20 different camera angles. Database viewers then enable users to browse the structured image stores by selecting interesting parameter combinations from parallel coordinate plots [35], by performing queries [3, 32], or by snapping to the closest available camera locations while navigating an emulated 3D view [21]. It is also possible to approximate the depicted surfaces to enable free camera movement [18]. However, a limitation of Cinema databases is that the flexibility of the *post hoc* analysis is limited by the generated images. Thus, if the database does not contain individual feature images, it is not trivially possible to toggle their visibility. Storing an image of each feature is also problematic as this drastically increases the amount of database elements. Therefore, it is necessary to intelligently depict feature groups with common *post hoc* analysis tasks in mind.

In the context of tracking superlevel set components in large-scale simulations, analysts should at least be able to toggle the visibility of components that are locally clustered together, and further filter components based on persistence. To this end, the proposed approach partitions components into a predefined number of groups based on a branch decomposition $\mathcal{B}$ of the current split tree $\mathcal{T}_t$, and a list of persistence intervals. The algorithm then generates images for each component group (Fig. 6). Specifically, the inputs of Alg. 2 are the entire simplicial complex $\mathcal{K}$, its scalar function f, a split tree segmentation $(\mathcal{T}_t, \phi_t, \psi_t)$, a set of camera specifications C, a set of levels L, a sorted list of persistence thresholds P, and the maximum number of component groups n; i.e., each timestep yields at maximum $|C| \cdot |L| \cdot |P| \cdot n$ images. First, the algorithm sorts all branches by persistence in descending order, and then inserts the n most persistent branches into their own new group (line 1-6). Each remaining branch is then inserted into the group that contains the most persistent branch it is attached to (lines 7-11). Note, such a branch and the corresponding group must exist as the branches are processed in sorted order.

Next, the algorithm iterates over the groups $G \in \mathcal{G}$, and the persistence intervals defined by P, to determine in each iteration the branches $\mathcal{B}' \subseteq G \in \mathcal{G}$ inside the current persistence interval $(P_i, P_{i+1}]$. Then, the algorithm derives for each level $l \in L$ the set of individual contours X of the current group, i.e., the borders of the superlevel set components. This is done by first determining the branches that include the current level, where each such branch B indicates the existence of an individual superlevel set component (Fig. 4c). To derive the set of simplices $\mathcal{K}' \subseteq \mathcal{K}$ that together completely contain the component of B, the algorithm first collects the set of edges $\mathcal{T}'$ that are connected to B above the current level (the crown of B that exceeds the level), and then retrieves all simplices of $\mathcal{K}$ that share at least one vertex with the subtree domain $\phi^{-1}(\mathcal{T}')$. It is necessary to include the tetrahedra adjacent to the subtree domain as they might contain parts of the linearly-interpolated contours.

Finally, the algorithm renders for all camera angles C a depth image and an ID mask of all group contours, where the depth images are used during *post hoc* analysis to compose 3D views, and a pixel of the ID mask stores the representative of the split tree edge that corresponds to the depicted contour. The images are then stored in the Cinema database, where they are also associated to the parameters that uniquely identify the images: their group ID, persistence interval, level, and camera angle. To efficiently retrieve during *post hoc* analysis an image that depicts a specific contour, the algorithm also stores, in line 28, the branch groups $\mathcal{G}$ of the current timestep in the Cinema database.

Note, the image generation is embarrassingly parallel as images for component groups and camera angles can be rendered independently. A limitation of this approach is that the sampling resolution of the parameter space is directly proportional to the resulting image database size. Moreover, the parameter sampling has to be determined beforehand, in which case adequate parameters might be unknown.

Fig. 6. Illustration of the image generation process for $\sim 5k$ vortices of the jet dataset at timestep 2000 based on two groups (cool and warm) and two persistence intervals (light and dark).

Algorithm 2: GenerateImages($\mathcal{K}$, f, $\mathcal{T}_t$, ϕ_t, ψ_t, C, L, P, n)

```
1  // Get branches sorted by persistence in descending order
2  B ← ComputeBranchDecomposition( T, ψ_t )
3  // Create groups for the first n most persistent branches
4  G ← ∅
5  for i ← 0 to n − 1 do
6      NewGroup( G, B_i )
7  // Add remaining branches to closest group
8  for i ← n to |B| do
9      B ← GetMostPersistentAttachedBranch( B, B_i, ψ_t )
10     G ← GetGroup( G, B )
11     AddToGroup( G, B_i )
12 // Generate group images for all persistence intervals and levels
13 foreach group G ∈ G do
14     foreach threshold p_i ∈ P where p_i ≠ max(P) do
15         // Filter grouped branches by persistence
16         B' ← { B ∈ G | p_i < (max ψ_t(B) − min ψ_t(B)) ≤ p_{i+1} }
17         foreach level l ∈ L do
18             // Add contour for each filtered branch that includes level
19             X ← ∅  // Set of contours
20             foreach B ∈ B' where min ψ_t(B) < l ≤ max ψ_t(B) do
21                 T' ← GetUpperTreeOfBranch( B, T_t, ψ_t, l )
22                 K' ← { σ ∈ K | σ ∩ φ_t^{-1}(T') ≠ ∅ }
23                 AddContour( X, K', f, l )
24             // Render depth and ID image of contours for each camera
25             foreach camera c ∈ C do
26                 I ← RenderContours( X, c )
27                 StoreInCinemaDB( I, G, p_i, p_{i+1}, l, c )
28 StoreInCinemaDB( G, t )
```

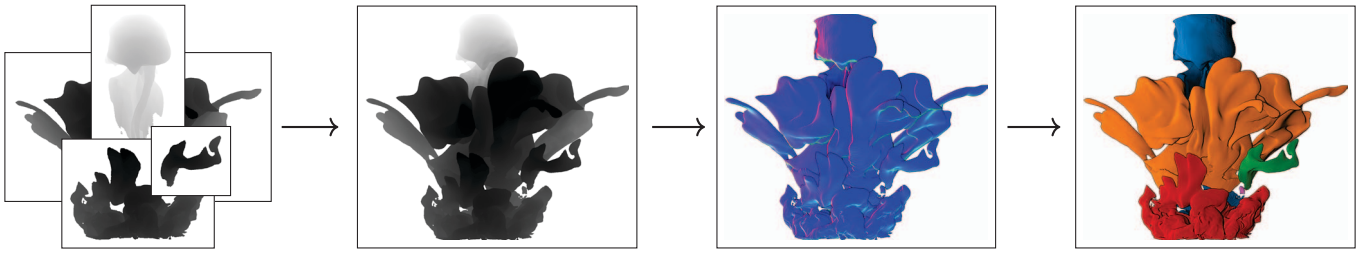

Fig. 8. Depth image-based rendering pipeline: multiple depth images and ID masks (a) are respectively composed into a single image (b), which are shaded based on approximated surface normals (c) and screen space ambient occlusion (d).

3.3 Post Hoc Database Exploration

This section describes the novel graph operation-based NTG algorithm (Sect. 3.3.1), the image compositing pipeline (Sect. 3.3.2), and the visual analytics framework (Sect. 3.3.3) that all use the generated Cinema database to effectively explore the underlying simulation.

3.3.1 Dynamic Nested Tracking Graphs

The core element of the *post hoc* analysis interface is a NTG that enables users to browse the simulation data across time and levels. Computing the NTG with the original algorithm [19] would make it necessary to predefine a set of levels, explicitly compute the superlevel set components for those levels, and then test the resulting component geometries for spatial overlaps across time (to determine their evolution) and across levels (to determine their nesting hierarchy). Thus, updating the graph is inefficient and unsuitable for large-scale simulations and *in situ* use cases. Alg. 3 describes a novel graph operation-based NTG algorithm (GO-NTG) that efficiently solves the three tasks of the NTG computation for a sorted list of adjacent timesteps T, a sorted list of levels L, and the graph structures that have been stored in the Cinema database at simulation runtime: the split trees $\mathcal{T}$, their scalar functions ψ, and the meta edges $\mathcal{M}$.

First, the algorithm determines the superlevel set components that are present for all timesteps and levels based only on the split trees $\mathcal{T}$ and their corresponding scalar fields ψ. Given a timestep $t \in T$ and a level $l \in L$, the algorithm inserts a new vertex into the set $\mathcal{V}$ for each edge $\langle u, v \rangle \in \mathcal{T}_t$ whose corresponding level interval includes l, as each such an edge represents an individual superlevel set component (red vertices in Fig. 7). In the following, each vertex of $\mathcal{V}$ is denoted as v_t^l to compactly indicate its corresponding timestep t, level l, and edge representative v in the split tree $\mathcal{T}_t$.

The nesting hierarchy $\mathcal{E}_N$ (red edges in Fig. 7) of the computed vertices $\mathcal{V}$ follows immediately from the structure of the split trees (black edges in Fig. 7). To identify the connections between vertices at level $l_i \in L$ for $i > 0$ (children) with vertices at level $l_{i-1} \in L$ (parents), the algorithm simply traverses the tree from each child towards the root until it encounters a parent and then inserts a new edge into $\mathcal{E}_N$ accordingly. Since the algorithm descends in a rooted tree, there always exists exactly one parent for each child.

The last task needs to establish the relationships between vertices at the same level for adjacent timesteps t and $t+1$. This can be done efficiently via the meta edges $\mathcal{M}_t$ of timestep t. Specifically, for each two vertices u_t^l and v_{t+1}^l one can determine if their corresponding segments overlap by checking if $\mathcal{M}_t$ contains the meta edge $\langle u, v \rangle$. If it does, the algorithm adds the edge $\langle u_t^l, v_{t+1}^l \rangle$ to $\mathcal{E}_T$. It is possible to filter tracking graph edges via an overlap threshold, or relax the tracking accuracy by adding edges if there exists an meta edge for a vertex pair further down in the split tree. Such a relaxation enables the tracking of fast moving components whose corresponding segments only overlap for lower levels. The advantage of the proposed algorithm is that such criteria can be chosen *post hoc* without access to the raw simulation data, and that all its steps can be trivially parallelized. Finally, the NTG is visualized according to the original approach of Lukasczyk et al. [19], where the width of edges can be linearly interpolated based on the segment sizes stored on the split tree vertices.

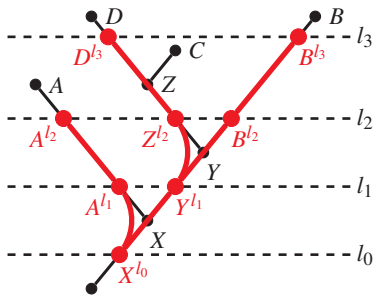

Fig. 7. Vertex and nesting tree computation based on split trees. Vertices correspond to edge-cuts (red nodes) for a set of levels (dashed lines), where each vertex represents a single superlevel set component, and is labeled by its corresponding edge representative, level, and timestep (here omitted). To determine their nesting hierarchy (red edges), the algorithm traverses the split tree from each vertex at level l_i with $i > 0$ towards the root, until the algorithm reaches its parent at level l_{i-1}.

Algorithm 3: GO-NTG(T, L, $\mathcal{T}$, ψ, $\mathcal{M}$)

```
1  V, E_N, E_T ← ∅  // Vertices, Nesting Trees, Tracking Graphs
2  // Compute Vertices
3  foreach timestep t ∈ T do
4      foreach level l ∈ L do
5          foreach edge ⟨u,v⟩ ∈ T_t where ψ_t(u) < l ≤ ψ_t(v) do
6              AddVertex( V, v, l, t )
7  // Compute Nesting Trees
8  foreach vertex v_t^l ∈ V where l ≠ min(L) do
9      AddEdge( E_N, v_t^l, GetParent(v_t^l, V, T_t, ψ_t) )
10 // Compute Tracking Graphs
11 foreach vertex u_t^l ∈ V where t ≠ max(T) do
12     foreach vertex v_{t+1}^l ∈ V do
13         if ⟨u,v⟩ ∈ M_t then
14             AddEdge( E_T, u_t^l, v_{t+1}^l )
15 return V ∪ E_N ∪ E_T
```

3.3.2 Image Retrieval and Compositing

To retrieve the image of a component corresponding to a vertex $v_t^l \in \mathcal{V}$ of the NTG for a specific camera angle, one first determines its branch group $G \in \mathcal{G}_t$, and then computes the persistence interval of the branch containing the edge represented by the vertex $v \in \mathcal{T}_t$ (Sect. 3.2.2). All parameters are then used to retrieve the closest available image in the database.

Fig. 8 illustrates the Depth Image Based Rendering (DIBR) pipeline that composes multiple depth images and ID masks into a single image. To improve spatial perception, the images are shaded based on approximated surface normals and screen space ambient occlusion [24], where components are colored based on the ID masks (Fig. 8c-d).

3.3.3 Visual Analytics Framework

Fig. 1 shows all linked views of the *post hoc* visual analytics framework that enable users to effectively explore the generated Cinema database: a composed 3D scene (top left), a split tree (top center), a persistence diagram (top right), and a nested tracking graph (bottom). User interface (UI) elements that correspond to an individual superlevel set component are consistently colored across all views, i.e., edges of the NTG, images of the components, branches of the split tree, and critical-point pairs of the persistence diagram. The core element of the interface is the NTG that illustrates the evolution of components for multiple levels, whereas the split tree shows their nesting hierarchy for the current timestep, and the persistence diagram shows their significance. The NTG is used to select time intervals, individual timesteps, and specific components, and the split tree and persistence diagram support analysts in choosing appropriate levels and persistence thresholds. The current persistence threshold is drawn as a diagonal red line in the persistence diagram, and levels of the NTG are drawn as horizontal lines in the split tree and persistence diagram, where the line of the currently selected level is also colored red. The 3D view is composed of images that are closest to the current parameter settings, i.e., the closest available database elements for a requested view angle, persistence interval, and selected level. Filtered components or components that do not exists for a selected level are grayed out in all views.

The interface provides three key mechanisms to handle numerous components: 1) before parameter updates the interface indicates the resulting numbers of components, split tree branches, and NTG edges; 2) components can be filtered based on size, persistence, and overlap thresholds; and 3) if numerous components have been chosen for visualization, the interface initially groups them together based on the nesting hierarchies and persistence values to generate a manageable amount of UI elements. Specifically, instead of rendering the entire split tree at once, the interface initially draws only a user-controlled number of the most persistent branches. Analysts then have the option to further expand individual branches, where the number of children is encoded by the width of the parent branch. Similarly, instead of rendering numerous tracks of the NTG for a certain level, these tracks are initially represented by their parent edges at the lower layers, and analysts can interactively toggle their visibility in a level-of-detail approach.

Layout updates of the graphs are only performed when necessary, or on request. For example, tightening the thresholds filters more components, which results in less NTG edges, split tree branches, and critical point pairs. Instead of updating the graph layouts immediately, the corresponding UI elements are simply removed, so that analysts can easily comprehend the updates without reorienting themselves within a new layout (Fig. 9 middle and bottom). However, analysts always have the option to recompute the layouts while ignoring the filtered components to generate smoother graphs. The interface also provides visual consistency when a new level is added to the NTG. Specifically, the layout algorithm described by Lukasczyk et al. [19] processes the layers of the NTG individually, and then stacks them in a bottom-up approach. As a consequence, inserting a level does not effect the layers of levels smaller than the new level.

Overall, the interface enables analysts to follow the history of individual components and component groups, filter them based on various metrics, and explore the simulation in a focus+context approach.

4 Results

This section evaluates the proposed methodology based on three real-world examples. The first case study compares the *post hoc* tracking algorithm to the explicit approach [19] by contrasting the resulting graphs for the 2016 scientific visualization contest dataset [14] (Sect. 4.1). To substantiate the claim that the proposed approach can be used to effectively explore large-scale simulations with numerous components, the other two case studies deal with much larger and more complex datasets—i.e., the simulation ensemble of the 2018 scientific visualization contest [15] (Sect. 4.2), and a computational fluid dynamics simulation with thousands of vortex features (Sect. 4.3).

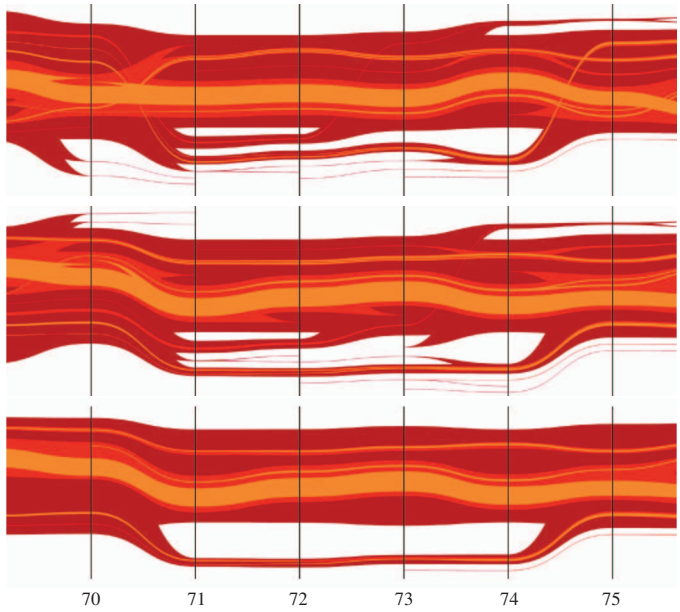

Fig. 9. NTGs of the viscous finger dataset for salt concentration levels 25, 30, and 35 (red to yellow). (Top) NTG generated with the explicit overlap-based tracking approach of Lukasczyk et al. [19]. (Middle and Bottom) NTGs generated with the split tree-based tracking approach, where the bottom graph is filtered by persistence, size, and overlap thresholds.

4.1 Viscous Fingering

This case study compares the results of the graph operation-based tracking approach and the original overlap-based algorithm [19] for the finite pointset method (FPM) simulation ensemble that was provided for the 2016 scientific visualization contest [14]. This ensemble was generated to explore the process of viscous fingering: an instability phenomenon that occurs at the interface between two mixing fluids of different viscosity, and is prominent in many fields of science and engineering—including geology, hydrology, and chromatography. Specifically, the ensemble members model the mixing process of salt solutions inside a water filled cylinder with an infinite salt supply at its top (gray surface in Fig. 1), where simulations incorporate stochastic effects to model the aleatoric uncertainty inherent in the strongly non-linear mixing process. While the solutions sink down to the bottom of the cylinder, they form characteristic structures with increased salt concentration values, called viscous fingers (colored components in Fig. 1). Fingers can be identified algorithmically by first sampling the salt concentration density of the pointsets on a regular grid and then deriving superlevel set components below the salt supply [17]. Lukasczyk et al. [19] demonstrated that NTGs can be used to effectively summarize shared properties of the fingers across different runs, timesteps, and initial parameters.

The top and middle row of Fig. 9 show two NTGs for the same simulation run, where the first graph is derived with the original approach that explicitly computes the overlap of superlevel set components, and the second graph is derived with the graph operation-based algorithm that processes meta edges and split trees. The graphs mostly match, except that the new algorithm adds more edges than the original approach. This is due to the segmentation-based tracking approach, as components inside a segment are collectively tracked based on the largest component (Sect. 3.2.1). Thus, the new algorithm detects at least the same amount of overlaps as the old approach, but also matches components whose volumes not explicitly overlap, which has advantages and disadvantages. For instance, the volumes of a fast moving component might not overlap in time, and therefore the original algorithm identifies the components in each timestep as new emerging features, which is semantically incorrect. However, the corresponding domain segments are likely to overlap since they correspond to the same moving maximum, and therefore the segmentation-based algorithm identifies the components as a single moving feature (thin lines of Fig. 9). To increase the accuracy of this matching, it is necessary to

choose an appropriate segmentation refinement level during the meta edge generation (Sect. 3.2.1). In all presented experiments, this refinement level was set to the persistence threshold that was used to remove noise, which yielded adequate results. In fact, choosing the refinement level in this way produces the same NTG as the explicit approach for the example shown in Fig. 9. The segmentation-based algorithm makes it also possible to interactively restrict or relax tracking criteria by respectively requiring a minimum amount of overlap, or by additionally matching segments that are connected via meta edges further down in trees. For instance, the bottom row of Fig. 9 shows the NTG of the second row which has been aggressively filtered to reduce clutter.

The main advantage of the segmentation-based algorithm is that once the meta edges have been computed, the NTG algorithm no longer requires access to the volumetric simulation data. Processing the meta edges and split trees is also significantly faster than explicitly computing superlevel set components and their respective overlaps: deriving NTGs for one ensemble member for the same parameters on the same hardware takes on average ~ 6 seconds with the old approach, and ~ 0.1 seconds with the new algorithm. NTGs for the following jet and asteroid case studies can still be computed in milliseconds, whereas the explicit approach requires several minutes. This speedup enables analysts to interactively update level, persistence, overlap, and size constraints. To summarize, the *post hoc* tracking algorithm is capable of tracking even more features than the explicit approach, and enables users to interactively generate and filter NTGs.

Obviously, an image database for such a small dataset requires far more disc space than the original data (Table 1). In fact, storing images become only beneficial for extremely large datasets, since the primary advantage of an image database is that its size grows proportional to the parameter sampling, independent of the size of the depicted simulation [2]. This can be observed in all presented experiments.

4.2 Asteroid Impacts

This case study examines an ensemble of extreme-scale simulations that are part of a threat assessment study of asteroid ocean impacts [15, 23], where individual ensemble members correspond to various impact scenarios based on different asteroid sizes, impact angles, and airburst heights. Specifically, each simulation is labeled according to the following convention: the first letter is an ensemble index, the second letter corresponds to the airburst height above sea level (A: None, B: $5km$, and C: $10km$), the third letter represents the asteroid diameter (1: $100m$, 3: $250m$, and 5: $500m$), and the fourth letter indicates the impact angle (0: 27.4°, 1: 45°, and 2: 60°). The main objective of the assessment is to explore the relationship between these parameters and the severeness of the tsunami they create upon impact. The following case study will demonstrate that the proposed approach enables analysts to efficiently explore and compare these different impact scenarios.

The original simulations advance an Eulerian grid that is adaptively refined at significant areas using the xRAGE simulation code [12]. The simulations compute, among others, a temperature field on a regular grid with either 300^3 or 500^3 vertices. To generate a Cinema database according to the proposed approach, these temperature fields are streamed into an emulated *in situ* environment that processes each timestep. Table 1 shows the total computation time and size of analysis and image products on a cluster node with an *Intel E5-2640v3* processor (16 cores) and 256GB memory. The stated time measurements include the computation of split tree segmentations [13], topological simplifications [33], and meta edges. Note, the image generation process is embarrassingly parallel, so the actual image generation time is much lower in practice. The provided image database contains 512^2 depth images and ID masks for 24 camera positions, 6 level values, 2 persistence intervals, and 4 component groups, which enables users to adequately rotate the 3D view and update parameters. Although the size of simulation *yA*31 is almost five times bigger on the 500^3 grid than on the 300^3 grid, their respective image databases are roughly the same size since components are depicted in a fixed number of groups. This demonstrates that the database size is decoupled from the size of the underlying data. Thus, the proposed approach can scale to very large data sizes with an acceptable trade-off in *post hoc* flexibility.

Fig. 10 shows for timestep 108 of simulation run *yA*31 (500^3) the split tree (bottom right), a composited 3D view (bottom left), and an NTG that is once colored by layer (top), and once colored by individual components for level $0.2eV$ (middle). Here, the NTG clearly illustrates that at the time of impact the entire region around the impact site is a single burning volume that disperses over time into four sub volumes (blue, red, orange, and green UI elements). The fact that NTG layers intelligently partition components into groups enables analysts to explore different levels and the corresponding components by expanding edges of the split tree and NTG. With this focus+context approach analysts can select individual components and their respective tracks for detailed examination, and further filter the graph based on size, overlap, and persistence thresholds, where the split tree and persistence diagram indicate important parameters. Based on a component selection, the interface then composes images from the Cinema database into a 3D rendering of the simulation. Note, although the database contains only two persistence intervals and therefore the visibility of some components can not be toggled off, they can at least be colored gray to indicate that they are filtered. All interface elements together guide users while examining specific components or component groups; e.g., the split tree indicates that the green component contains the global maximum, the NTG shows its temporal evolution, and the 3D view shows the component in the physical domain. The low overhead of the GO-NTG algorithm and the image compositing enable analysts to quickly update parameters and interactively cycle through the ensemble.

4.3 Jet Simulation

This case study focuses on a jet simulation and was chosen to illustrate the utility of the proposed approach for feature-rich datasets. The simulation describes a high-velocity fluid jet entering a medium at rest. Due to viscous effects, a large vortex ring is generated at the top of the jet that quickly breaks down into a large number of smaller vortices as the flow transitions towards turbulence. From the original velocity data, vorticity magnitude is computed and subjected to analysis to identify individual vortices as superlevel set components of high vorticity.

Fig. 6 shows the roughly five thousand superlevel set components that exist for level 500 at timestep 2000. Even after topological simplification, the split tree of that timestep still consists of more than $100k$ branches. As explained previously, the image database size only grows proportional to the parameter sampling and not to the feature complexity and quantity (Table 1). Moreover, grouping components based on split tree branches has the advantage that each group constitutes a local component cluster. Toggling the visibility of these groups therefore supports effective spatial peeling. As even hundreds of images can be composed at interactive framerates, the proposed analysis framework enables analysts to quickly browse through time and update parameters. However, to provide more flexibility, it is necessary to generate image databases for a larger number of component groups and persistence intervals, which significantly increases the databases sizes even further. To summarize, the demonstrated case studies show that image databases are not necessarily small, but seem to grow significantly smaller when moving towards extreme-scale simulations [2].

Dataset	#Cells	#Steps	$\lvert\mathbf{C}\rvert\cdot\lvert\mathbf{L}\rvert\cdot\lvert\mathbf{P}\rvert\cdot\mathbf{n}$	$\mathbf{T_A}$	$\mathbf{T_I}$	$\mathbf{S_S}$	$\mathbf{S_A}$	$\mathbf{S_I}$
VF Run1	$25.1 \cdot 10^3$	100	24×5×1×1	2 m	3 m	90 MB	3 MB	1 GB
VF Run2	$25.1 \cdot 10^3$	100	24×5×1×2	2 m	7 m	90 MB	3 MB	2 GB
VF Run3	$25.1 \cdot 10^3$	100	24×5×1×3	2 m	15 m	90 MB	3 MB	3 GB
yA31	$12.4 \cdot 10^7$	260	24×6×2×4	45 h	43 h	121 GB	28 MB	21 GB
yA31	$2.6 \cdot 10^6$	260	24×6×2×4	16 h	13 h	26 GB	17 MB	20 GB
yB31	$2.6 \cdot 10^6$	260	24×6×2×4	17 h	13 h	26 GB	12 MB	19 GB
yC31	$2.6 \cdot 10^6$	260	24×6×2×4	15 h	14 h	26 GB	14 MB	22 GB
Jet	$3.3 \cdot 10^7$	3000	24×6×2×4	25 h	15 d	375 GB	108 MB	260 GB

Table 1. Statistics of the presented case studies. From left to right: name, cell count (all regular grids), number of timesteps, image sampling, total aggregated computation time of analysis and image products, and total sizes of simulations, analysis products, and images (512^2 pixels each).

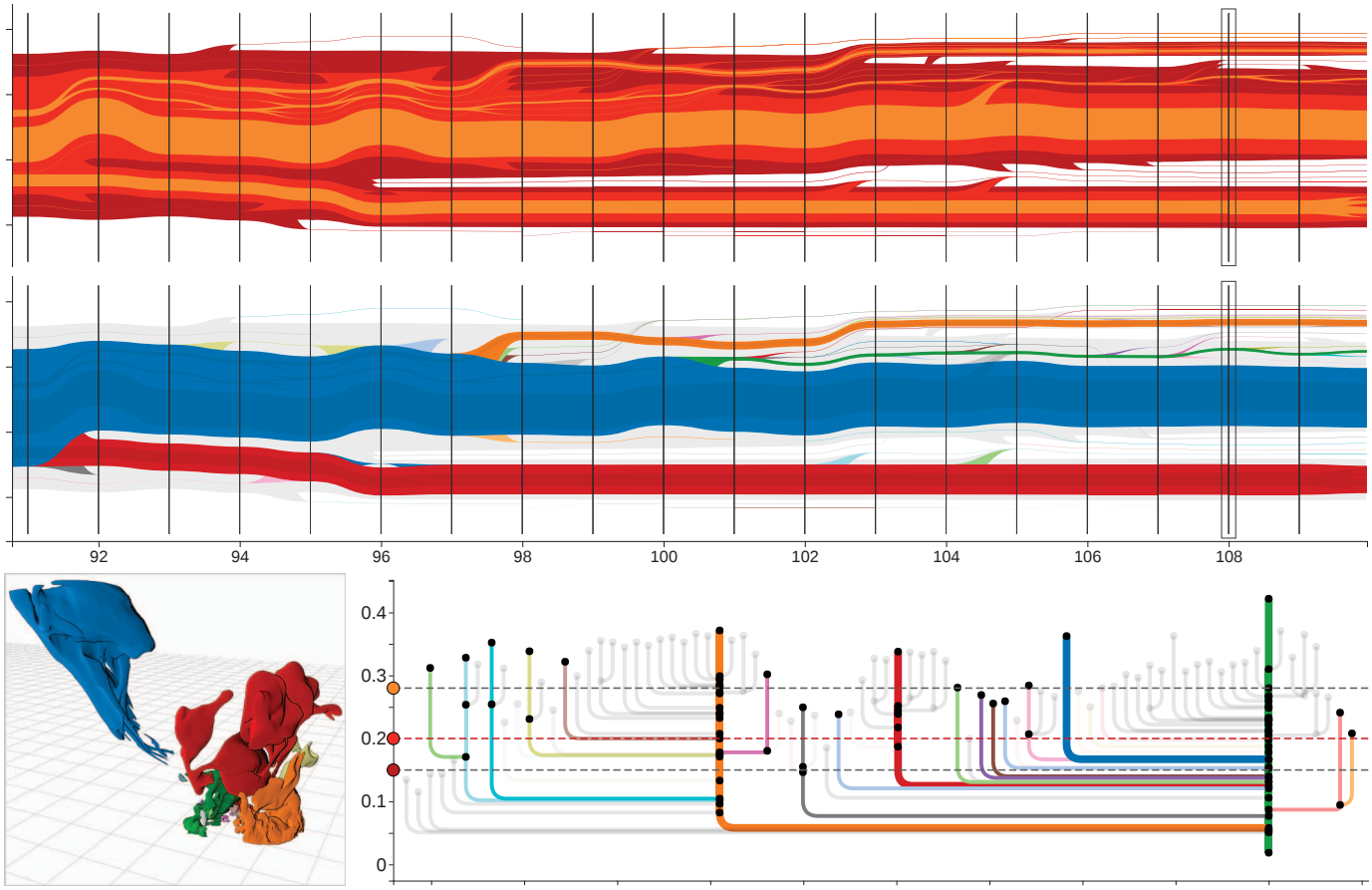

Fig. 10. Analysis of the asteroid impact yA31 temperature field. (Top and Middle) NTGs visualizing the evolution of the temperature field for the levels $0.15eV$, $0.2eV$, and $0.28eV$, where the second graph highlights level $0.2eV$. (Bottom) The 3D composited view and the merge tree at timestep 108 for level $0.2eV$. The proposed approach naturally partitions the temperature volumes into groups: the asteroid trail (dark blue), the cloud that raises to the stratosphere (dark red), the flame front that thrusts forward over the ocean (dark orange), and the burning region at the impact site (dark green).

5 Conclusion and Future Work

This paper described a scalable processing pipeline that enables the interactive visual analysis of large-scale scientific simulations where superlevel set components and their evolution are of primary interest. The approach first stores split trees, meta edges, and images of component groups at simulation runtime in a Cinema database, which is later used to explore *post hoc* the underlying simulation in a topology-based visual analytics framework. To this end, the paper described a split tree segmentation-based tracking algorithm (to correlate components over time), a branch decomposition-based image generation algorithm (to provide a rendering of component geometries), and a graph operation-based NTG algorithm (to derive NTGs *post hoc* by processing the stored topological abstractions). All algorithms have been implemented in the Topology ToolKit (TTK) [32] and are therefore accessible as VTK filters [27] inside ParaView [1].

The proposed *post hoc* exploration framework enables users for the first time to navigate Cinema databases with a focus on features rather than along predetermined parameter axes. However, the experiments have shown that image databases are only advantageous if a) they are much smaller than the raw simulation data, and/or b) if accessing and rendering the raw data can not be done at interactive framerates. It also appears fruitful to replace—or supplement—the image database with downsampled volume data or simplified contour meshes. For instance, the volume data could be compressed while preserving the scalar field topology [31]. Moreover, the image generation and the branch bundling are based on branch decompositions, which might vary drastically over time and are susceptible to noise. In the presented experiments, the branch bundling appears to be stable as long as there is a relatively small number of bundles which contain many branches. In the other extreme, i.e., many bundles consisting of few branches, it becomes apparent that branches frequently change bundles. In future work, this instability need to be explored further to mitigate its effect. Note, the branch decomposition has no effect on the tracking accuracy as Alg. 1 is based on segments and not on branches. However, the accuracy of Alg. 1 depends on the scalar interval resolution of the split tree segmentation, and a suitable resolution is often not known a priori.

It is also necessary to perform additional experiments that examine the database generation during massively parallel *in situ* execution to allow scaling to state-of-the-art, largest-scale simulations, which stand to benefit from the proposed methodology. This hinges crucially on the scalability of the split tree computation that is central to the approach; here, e.g., the *parallel peak pruning* [9] approach could be used.

To facilitate practically usability, it appears possible to introduce automation of parameters (such as number of persistence intervals or camera angles stored) through heuristics or optimization techniques, to keep the generated database within a given budget while maximizing *post hoc* flexibility, or alternatively, to ensure a specified degree of flexibility while minimizing database size. For example, the integration of the *VOIDGA* approach [18] could reduce the number of stored camera locations, which would allow to sample other parameters more densely. Finally, database generation could benefit from low-level technical improvements, such as different compression methods and data formats.

Acknowledgments

This research was funded by the German research foundation (DFG) within the IRTG 2057 “Physical Modeling for Virtual Manufacturing Systems and Processes”. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

REFERENCES

[1] J. Ahrens, B. Geveci, and C. Law. ParaView: An End-User Tool for Large-Data Visualization. *The Visualization Handbook*, pp. 717–731, 2005.

[2] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. Rogers, and M. Petersen. An Image-Based Approach to Extreme Scale In Situ Visualization and Analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 424–434, 2014.

[3] G. Aldrich, J. Lukasczyk, J. D. Hyman, G. Srinivasan, H. Viswanathan, C. Garth, H. Leitte, J. Ahrens, and B. Hamann. A Query-based Framework for Searching, Sorting, and Exploring Data Ensembles. *Computing in Science & Engineering*, 2019.

[4] T. Biedert and C. Garth. Contour Tree Depth Images for Large Data Visualization. In *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization*, pp. 77–86, 2015.

[5] P. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell. Analyzing and Tracking Burning Structures in Lean Premixed Hydrogen Flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010.

[6] P. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive Exploration and Analysis of Large-Scale Simulations Using Topology-Based Data Segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2011.

[7] H. Carr, J. Snoeyink, and U. Axen. Computing Contour Trees in all Dimensions. *Computational Geometry*, 24(2):75 – 94, 2003.

[8] H. Carr, J. Snoeyink, and M. Van De Panne. Flexible Isosurfaces: Simplifying and Displaying Scalar Topology using the Contour Tree. *Computational Geometry*, 43(1):42–58, 2010.

[9] H. Carr, G. Weber, C. Sewell, and J. Ahrens. Parallel Peak Pruning for Scalable SMP Contour Tree Computation. In *IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 75–84, 2016.

[10] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological Persistence and Simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[11] H. Edelsbrunner, D. Morozov, and V. Pascucci. Persistence-Sensitive Simplification Functions on 2-Manifolds. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pp. 127–134, 2006.

[12] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, et al. The RAGE Radiation-Hydrodynamic Code. *Computational Science & Discovery*, 1(1):63pp, 2008.

[13] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Task-Based Augmented Merge Trees with Fibonacci Heaps. In *IEEE Symposium on Large Data Analysis and Visualization*, 2017.

[14] IEEE VIS. Scientific Visualization Contest. `http://www.uni-kl.de/sciviscontest/`, 2016.

[15] IEEE VIS. Scientific Visualization Contest. `http://sciviscontest2018.org/`, 2018.

[16] W. Köpp and T. Weinkauf. Temporal Treemaps: Static Visualization of Evolving Trees. *IEEE transactions on visualization and computer graphics*, 25(1):534–543, 2019.

[17] J. Lukasczyk, G. Aldrich, M. Steptoe, G. Favelier, C. Gueunet, J. Tierny, R. Maciejewski, B. Hamann, and H. Leitte. Viscous Fingering: A Topological Visual Analytic Approach. In *Applied Mechanics and Materials*, vol. 869, pp. 9–19, 2017.

[18] J. Lukasczyk, E. Kinner, J. Ahrens, H. Leitte, and C. Garth. VOIDGA: A View-Approximation Oriented Image Database Generation Approach. In *IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV)*, 2018.

[19] J. Lukasczyk, G. Weber, R. Maciejewski, C. Garth, and H. Leitte. Nested tracking graphs. In *Computer Graphics Forum*, vol. 36, pp. 12–22, 2017.

[20] P. Oesterling, C. Heine, G. H. Weber, D. Morozov, and G. Scheuermann. Computing and Visualizing Time-Varying Merge Trees for High-Dimensional Data. In *Topological Methods in Data Analysis and Visualization*, pp. 87–101, 2015.

[21] Open-Source Cinema Viewers. CVLIB. http://cinemaviewer.org/, 2018.

[22] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli. Multi-Resolution Computation and Presentation of Contour Trees. In *Proc. IASTED Conference on Visualization, Imaging, and Image Processing*, pp. 452–290, 2004.

[23] J. Patchett, G. Gisler, B. Nouanesengsy, D. Rogers, G. Abram, F. Samsel, K. Tsai, and T. Turton. Visualization and Analysis of Threats from Asteroid Ocean Impacts. *Los Alamos National Laboratory*, 2016.

[24] T. Ritschel, T. Grosch, and H.-P. Seidel. Approximating Dynamic Global Illumination in Image Space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, pp. 75–82, 2009.

[25] D. Rogers, J. Woodring, J. Ahrens, J. Patchett, and J. Lukasczyk. Cinema Database Specification - Dietrich Release v1.2. Technical Report LA-UR-17-25072, Los Alamos National Laboratory, 2018.

[26] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing Features and Tracking their Evolution. *Computer*, 27:20–27, 1994.

[27] W. J. Schroeder, K. Martin, and W. E. Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics (3rd edition)*. Kitware, Inc., 2004.

[28] D. Silver and X. Wang. Tracking Scalar Features in Unstructured Data Sets. In *Visualization '98. Proceedings*, pp. 79–86, 1998.

[29] B. S. Sohn and C. Bajaj. Time-Varying Contour Topology. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):14–25, 2006.

[30] M. Soler, M. Plainchault, B. Conche, and J. Tierny. Lifted Wasserstein Matcher for Fast and Robust Topology Tracking. In *IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV)*, 2018.

[31] M. Soler, M. Plainchault, B. Conche, and J. Tierny. Topologically Controlled Lossy Compression. In *IEEE Pacific Visualization Symposium (PacificVis)*, pp. 46–55, 2018.

[32] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[33] J. Tierny and V. Pascucci. Generalized Topological Simplification of Scalar Fields on Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2005–2013, 2012.

[34] W. Widanagamaachchi, C. Christensen, V. Pascucci, and P. Bremer. Interactive Exploration of Large-Scale Time-Varying Data using Dynamic Tracking Graphs. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 9–17, 2012.

[35] J. Woodring, J. Ahrens, J. Patchett, C. Tauxe, and D. Rogers. High-Dimensional Scientific Data Exploration via Cinema. In *IEEE Workshop on Data Systems for Interactive Analysis (DSIA)*, pp. 1–5, 2017.