# GPU-ACCELERATED SUMMARIZATION AND RECONSTRUCTION TECHNIQUES FOR BIG DATA ANALYSIS AND VISUALIZATION

A Dissertation

Submitted to the Graduate School

of the University of Notre Dame

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

by

Martin Imre

Chaoli Wang, Director

Graduate Program in Computer Science and Engineering

Notre Dame, Indiana

April 2020

# GPU-ACCELERATED SUMMARIZATION AND RECONSTRUCTION TECHNIQUES FOR BIG DATA ANALYSIS AND VISUALIZATION

Abstract

by

Martin Imre

With the ever-growing amount of data we are capable of collecting nowadays, the need for methods to analyze and generate insight into the big data emerges. Data visualization offers a powerful tool to allow humans to better understand the data. However, the large-scale data generated through simulation or collected from real-world scenarios leads to an insurmountable amount to sift through for humans, even when the data are visualized in a concise format. To combat this, it is necessary to simplify the data using summarization techniques. While summarization allows an overview that is easier to digest for humans, it comes with the drawback of omitting parts of the data. To overcome this drawback, data reconstruction techniques allow for a level-of-detail analysis of the underlying data. They further make it possible to synthesize missing data. For both data summarization and reconstruction, it is important to tackle different kinds of data in their respective ways. In this dissertation, I describe several ways to summarize and reconstruct time-varying multivariate, vector field data, and graph data.

Time-varying multivariate volumetric data typically stem from scientific simulations that describe physical or chemical processes, usually in either two or three spatial dimensions, a temporal dimension, and contain multiple variables. Volume visualization techniques are usually used to visualize and analyze them. In this dissertation, I focus on data analysis using isosurface rendering, a commonly used volume visualization technique.

While isosurface rendering allows detailed insight into time-varying multivariate data sets, the sheer complexity of them makes it often impossible to analyze and visualize all the data at once. A major challenge is the selection of interesting or important parts to bring to the attention of the analyst. Previous works have presented algorithms to select salient values, however, these algorithms do not scale well enough for big data analysis. In this dissertation, I present an acceleration and analysis framework to efficiently analyze and visualize complete large-scale time-varying multivariate data sets using isosurfaces.

A different type of data produced by scientific simulation is vector field data. At the core of describing fluid dynamics, these data sets are composed of vector fields revealing the underlying development of the flow. When it comes to the simulation of unsteady flow, scientists often face the challenge of only being able to generate output at a coarse temporal granularity. This leads to the problem of reconstructing the data at missing time steps with high quality for postprocessing and analysis. In this dissertation, I study three different deep-learning approaches to reconstruct missing vector fields from a small set of stored ones.

Another way to navigate through data is by using a graph representation. Graph representations are commonly used to show the relationship among entities. A typical way to depict graph is a node-link diagram with entities being the nodes and their connections the links. Similar to isosurface rendering, graph visualization suffers from the data overloading issue. Further, computing a pleasing and information revealing layout for large graphs can take a long time. To combat these problems, I present a framework to efficiently summarize a graph into sparse levels, compute a layout, and then reconstruct this information for denser levels of the graph.

CONTENTS

## FIGURES

## TABLES

## ACKNOWLEDGMENTS

First and foremost, I want to thank my advisor, Professor Chaoli Wang, for support and persistence throughout graduate school. I am grateful for his diligence in reviewing my works and his critical thinking. In addition, I would like to thank my dissertation committee, Professor Ronald Metoyer, Professor Collin McMillan, and Dr. Hanqi Guo. I greatly appreciate all the feedback and the insightful questions they had. I also want to thank them for their flexibility in scheduling a remote dissertation defense spanning four different time zones with up to nine hours of time difference during the COVID-19 pandemic. Besides that, I want to thank Dr. Hanqi Guo for being a great host and mentor during my summer internship at Argonne National Laboratory in 2018.

Very special thanks goes to Joyce Yeats, who supported me throughout this journey through the last four years with concerns of various natures. Her never-ending effort to answer questions and solve roadblocks will never be forgotten.

I want to thank all fellow students I have collaborated with and shared lab spaces with. Additionally, I want to thank my friends, who ameliorated my workday with discussions, debugging sessions, and other shenanigans during work and free time. Lastly, I want to thank my girlfriend for supporting me throughout this endeavor and provided a home during the COVID-19 pandemic.

My Ph.D. research was supported in part by the U.S. National Science Foundation through grants IIS-1456763, IIS-1455886, CNS-1629914, and DUE-1833129. I would also like to thank the Notre Dame's Notebaert Professional Development Fund, which supports my travel to IEEE VIS 2018.

# CHAPTER 1

# INTRODUCTION

Most scientific simulations create data sets that are time-varying and multivariate. To successfully analyze them and generate insight, visualization techniques should be used. While this is easy when it comes to a single data point describing a time step and a variable, it becomes a challenging task when considering a complete large-scale simulation data set. These simulations can easily have hundreds of time steps and tens of variables. Visualizing all of those at the same time is practically impossible. This poses a strong need for proper summarization techniques to extract the most important information from the data.

Data summarization, sparsification, and reduction are powerful techniques that facilitate giving an overview of the underlying data. Using them, it is important to produce a high-quality summarization, that extracts the most important aspects of the data. While previous works have introduced various techniques for summarization, e.g., via representative selection, most of them lack scalability. In this dissertation, I focus on accelerating previous approaches to allow them to scale for large-scale data sets. Obtaining a scalable approach to analyze time-varying multivariate data sets, for example, grants the possibility to analyze them as a whole rather than separately. When simplifying data in such ways, we often have to omit part of the data. This calls for proper methods to reconstruct the missing information.

Data reconstruction methods tackle the aforementioned problem. Scientists are often faced with the decision on the granularity of output data from a simulation. This problem is aggravated by slow storage devices which are a common bottleneck in supercomputing. This leads to the data being compressed by only storing them at a coarse granularity, e.g., only storing every *n*th time step. Recovering the missing information is desired, but poses significant challenges. Some reconstruction of them are solely concerned with synthesizing missing data, e.g., generating a missing time step between two neighboring ones. Others are trying to augment the existing data by identifying and highlighting features. In this dissertation, I focus on three different ways of data reconstruction. First, we use deep-learning techniques to identify features, that are not clearly defined, in scalar fields. Second, we reconstruct missing time steps of flow simulations applying deep-learning as well. Third, we combine summarization with reconstruction to allow a quick layout computation for visualization of large graphs.

When inspecting the data set one can make a distinction among the type of data. In this dissertation, I consider time-varying multivariate, unsteady flow field, and graph data. In this chapter, I will introduce each type of data together with some challenges in analyzing them and how we design summarization and reconstruction techniques to solve them.

## 1.1 Time-Varying Multivariate Volumetric Data

Time-varying multivariate volumetric data describe the evolution of multiple variables in three spatial dimensions as scalar fields over time. These data sets can have hundreds of time steps and tens of variables. To obtain comprehensive insights, one should conduct a complete investigation by considering the time-varying and multivariate nature of scientific data sets and enabling comparative visualization across time and variable. This, however, remains challenging due to the size and complexity of the data sets.

A single variable at a given time step can easily be analyzed by visualizing it via volume visualization. There are two popular techniques for the rendering of scientific volume data sets: isosurface rendering and direct volume rendering. The former extracts and visualizes surface geometry equal to a given isovalue while the latter maps voxels to optical quantities (color and opacity) for back-to-front or front-to-back compositing. Figure 1.1 shows an example of both techniques. In this dissertation, I focus on the analysis using isosurface

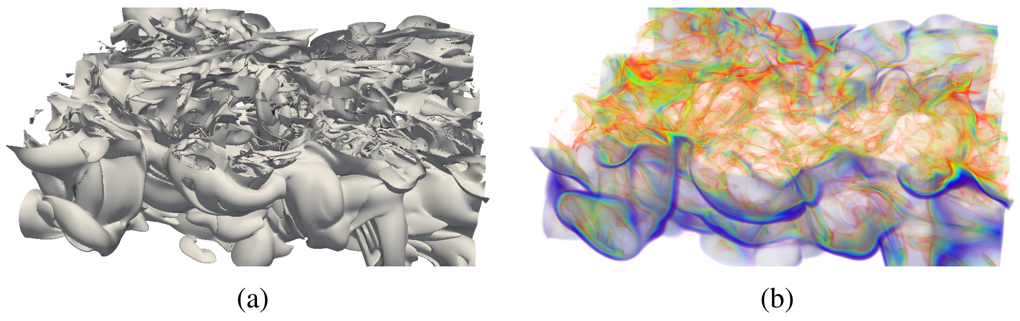


Figure 1.1. Example of direct isosurface rendering (a) and volume rendering (b) showing the combustion data set.

rendering.

### 1.1.1 Acceleration Isosurface Similarity Map Computation

One key question for this surface-based approach is how to select salient or representative isosurfaces [24, 202, 220] so that the structures of the entire data set can be visually perceived by observing these representatives. A commonly adopted strategy is to select the most distinctive isosurfaces from a set of sample ones based on a similarity measure. Early approaches relied on data histograms and higher order moments [202]. Recently, Bruckner and Möller [24] employed mutual information to identify the similarity among isosurfaces and depict them in a so-called *isosurface similarity map* (ISM). Given a discrete set of $n$ sampled isovalues, the self-similarity map is an $n \times n$ symmetric matrix recording the similarity values for each pair of isosurfaces. Haidacher et al. [83] used similarity maps to compare two different variables of a single volume for multi-modal surface similarity. In this case, the similarity map is asymmetric.

However, the biggest impediment for leveraging ISMs to study time-varying multivariate data sets is the huge computation cost involved. The previous work [24] reported around 25 minutes to compute a single similarity map using 256 isosurfaces of a volume with the size $512 \times 512 \times 361$. For a data set with ten variables and a hundred time steps, this amounts to more than 17 days just for getting self-similarity maps for all volumes, not to mention the additional time to compute asymmetric similarity maps for different pairs of variables or time steps.

To tackle these issues, I present a cost-effective GPU-accelerated solution that optimizes every step of the process to efficiently generate ISMs. The major contributions are the generation of distance field based on isosurface approximation, a heuristic to improve the performance for the closest point query using *bounding volume hierarchy* (BVH)-trees. Further, we implement everything in CUDA to leverage the massive GPU parallelism.

In Chapter 3, I present the GPU-accelerated solution that paves the way to computing ISMs for large-scale time-varying multivariate data sets.

### 1.1.2 Overcoming Oversampling and Generating Nearly Equally Spaced Isosurfaces

The isosurface-based analysis of time-varying multivariate data strongly depends on the surfaces selected to represent a variable at a time step. Previous works usually select those representative surfaces based on an initial sample of the whole value range. One major challenge exists for these approaches: it is essential for them to start with a set of reasonably good samples of isosurfaces that capture different features in a balanced way. Otherwise, the features missing in the samples will not be recovered in the later stages, or the selection may be biased by favoring the features corresponding to more samples. However, straightforward sampling techniques do not guarantee the desired set of samples. Uniform sampling is likely to miss some features when many of them reside in a small value range. Although sampling according to histograms of voxel values, i.e., placing more samples in the value ranges with more voxels, may alleviate this problem to some degree, it still suffers from oversampling as the value ranges with more voxels do not necessarily indicate more distinctive features.

In this dissertation, I present an approach for identifying nearly equally spaced isosurfaces, so that the distance between neighboring surfaces is as similar as possible to the average distance. Our solution ensures that the isosurfaces corresponding to neighboring isovalues are distinct enough according to the given distance measure. When identifying a small number of isosurfaces, we can consider the resulting isosurfaces as salient features on their own. Compared to the similarity-based approaches for identifying representative isosurfaces, our approach does not require the isosurfaces to be selected from a limited set of sample ones. It not only has a wider search space but also explicitly controls over the resulting isosurfaces, which can potentially lead to better results. Compared to the topology-based approaches, our method is more flexible when equipped with different distance measures. Although this offers great flexibility, our method relies on features being a function of isovalues. Given that precondition, our algorithm may capture the topological changes if the distance measure is topology-aware, and it may produce isosurfaces with distinct shapes if the distance measure is shape-aware. In addition, when a large set of isosurfaces is identified, the results can serve as reliable input to other volume analysis and visualization tasks.

In Chapter 4, I present a two-stage algorithm that first quickly estimates to a rough solution in the first stage, and refines it in the second stage.

### 1.1.3 Analysis and Visualization of Large-Scale Time-Varying Multivariate Data Sets

While the previously mentioned contributions allow the efficient and qualitative evaluation of a single variable at a given time step, they both lack an overarching component to compare and visualize a complete data set. The goal for this is to be able to investigate volumetric data evolution over space and time and across multiple variables and ensembles by extracting a range of isosurfaces, computing their similarities, identifying representative surfaces, and presenting everything in a visual interface to organize, summarize, and explore the underlying time-varying multivariate data set.

Several key challenges are involved in this approach. First, studying the similarities for all the surfaces across time steps and variables leads to a very large number of similarity maps. For instance, for a data set of 100 time steps and ten variables, the total number of similarity maps amounts to 55,000. These include 1000 self-similarity maps, 49,500 temporal similarity maps, and 4,500 variable similarity maps. Without an effective way to organize and visualize them, examining such a large number of similarity maps would become a daunting task. Second, even though similarity maps can help us see the summarized information, we still need to examine related isosurfaces in the original spatial view to identify where and how they are (dis)similar. Third, rendering just a few surfaces may already lead to serious occlusion and clutter which prevents clear observation and comparison. This calls for new techniques that render a number of surfaces with minimal occlusion while preserving their spatial relationships or context.

In Chapter 5, I show how we tackle the above challenges and present the *matrix of isosurface similarity maps* (MISM), a new approach for comparative visualization of time-varying multivariate volume data.

### 1.1.4 Feature Identification in Plasma Fusion Simulations

While the previously introduced topics were mostly concerned with summarization techniques to analyze and visualize time-varying multivariate data, this part shows an approach of augmenting the data by detecting features in time-varying plasma fusion simulation data. Plasma fusion energy is a promising future source of energy with a minimal ecological footprint. A leading candidate for controlled fusion technique is to use a *tokamak*, which confines hot plasma in a torus-shaped reactor with a strong magnetic field. Figure 1.2 shows a schematic view of a tokamak reactor. The edge of the plasma plays a dominant role in the confinement of the plasma but needs to be managed to ensure high confinement without ruining the tokamak walls. Scientists conduct large-scale 5D gyrokinetic fusion plasma simulations using XGC [30, 31] in order to study the edge and improve the tokamak design. The output is on a number of discrete 2D planes around the tokamak (similar to the blue rings in Figure 1.2), with each poloidal plane representing a 2D cross-

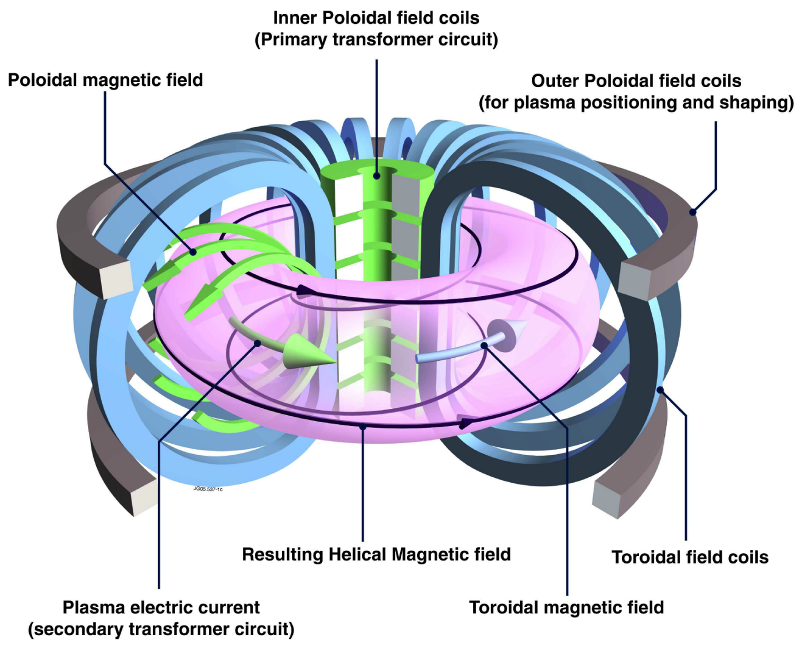

Inner Poloidal field coils
(Primary transformer circuit)
Poloidal magnetic field
Outer Poloidal field coils
(for plasma positioning and shaping)
Resulting Helical Magnetic field
Toroidal field coils
Plasma electric current
(secondary transformer circuit)
Toroidal magnetic field

Figure 1.2. The schematic view of a tokamak (source: European Fusion Development Agreement-Joint European Torus).

section of the simulated tokamak reactor.

An open problem in plasma fusion research is the detection of *blobs*—regions of high turbulence that are thought to be a major cause of lost confinement at the edge—in fusion simulation data. Blob detection is challenging because blobs are *non-well-defined features*. While experienced scientists are able to empirically identify blobs, defining these features mathematically remains an open challenge [32, 42].

D'Ippolito et al. [52] characterized blobs as higher-than-background density filaments propagating outwards and lower-than-background density filaments propagating inwards, and thus blobs have certain shapes, sizes, value ranges, and could only appear in certain areas of the domain. Scientists urge the need for reliable detection methods to better understand the formation and impact of these blobs.

In practice, scientists empirically encircle a blob as a simply-connected and closed isocontour, or a *local contour*, in the output 2D scalar field. Scientists have been using simple heuristics to find proper local contours based on their intuitions of blobs. In general, blobs have a certain size, shape, value ranges, and could only appear in certain areas of the domain.

To this end, Davis et al. [47] and Zweben et al. [239] used half-maximum levels as isovalues to compute local contours for blob detection. In Wu et al. [225], the isovalue is determined by the statistical distribution of the input data in a region of interest. However, all existing methods are based on arbitrary choices that involve manual selections and thus hard to generalize to the time-varying simulation output data.

In Chapter 6, I present a new deep-learning-based approach for detecting salient isocontours for blob identification.

## 1.2 Vector Field Data

A different kind of scientific simulation data is vector field data (VFD). These are similar to the volumetric data sets, but instead of being scalar fields, are vector fields. Flow visualization is a method for visualizing vector fields and has been a core area of research and remains an active one in scientific visualization for three decades. Among many approaches developed for flow visualization, integration-based approaches stand out as the most popular form for visualizing vector fields where we place particles or seeds in the domain to trace flow lines from the underlying flow fields for visual representation, rendering, and understanding. These flow lines include streamlines traced from steady flow fields and pathlines traced from unsteady flow fields. Figure 1.3 shows an example of those tracing results rendered. In this dissertation, I propose a deep-learning approach to reconstruct vector fields from missing time steps of a simulation.

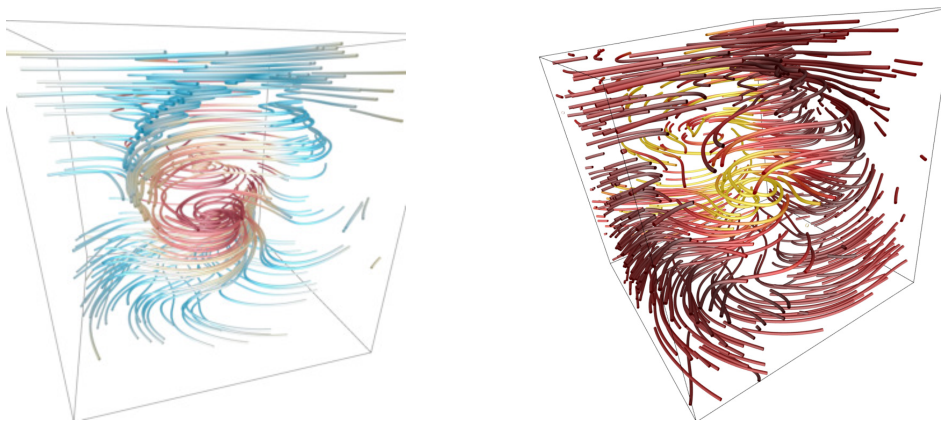


Figure 1.3. Example of stremalines (a) and pathliens (b) showing the tornado data set.

### 1.2.1 Unsteady Flow Field Reconstruction

For large-scale scientific simulations, scientists usually can only afford to store a fraction of the simulation data in the reduced form. As high-performance computing systems are often constrained with respect to data movement and storage, deciding what data are most essential to store for post hoc analysis becomes a common task for many scientists running their simulations. As data reduction becomes inevitable, data reconstruction or restoration provides a necessary means to recover data resolutions and details from the reduced data.

For unsteady flow fields, we consider a common scenario where scientists store vector field data sparsely (e.g., every tenth time step) during the simulation and our goal is to restore intermediate time steps as accurate as possible during postprocessing.

In Chapter 7, I conduct a study of three different deep learning solutions for unsteady flow field reconstruction from sparsely-output vector fields.

## 1.3 Graph Data

Graphs can represent a multitude of different kinds of data ranging from social networks to mathematical problems. Visualizing graphs can improve the understanding of the underlying data tremendously and allow insights into the results of graph algorithms. While graph algorithms have been thoroughly studied and optimized, graph-drawing algorithms still struggle when it comes to huge graphs. In this dissertation, I present a spectrum-preserving graph sparsification method, that allows to compute a layout on a very sparse representation of and map it back to the original graph.

### 1.3.1 Spectrum-Preserving Sparsification for Big Graph Visualization

Spectral methods are playing an increasingly important role in many graph-based applications [201], such as scientific computing [192], numerical optimization [41], image processing [187], data mining [172], machine learning [48], and graph analytics [101]. For example, classical spectral clustering algorithms leverage the eigenvectors corresponding to a few smallest nontrivial (i.e., nonzero) eigenvalues of Laplacians for low-dimensional spectral graph embedding, which is followed by a k-means clustering procedure that usually leads to high-quality clustering results. Although spectral methods have many advantages, such as easy implementation, good solution quality, and rigorous theoretical foundations [119, 133, 167], the high memory and computation cost due to the involved Laplacian eigenvalue problems could hinder their applications in many emerging big graph analytical tasks [36, 58, 101].

*Graph sparsification* refers to the approximation of a large graph using a sparse graph. Compared to the original graphs, sparsified graphs provide a number of advantages for subsequent analysis and visualization. For example, sparsified transportation networks allow for developing more scalable navigation or routing algorithms for large transportation systems; sparsified social networks enable more effective understanding and prediction of information propagation in large social networks; and sparsified matrices can be leveraged to efficiently compute the solution of a large linear system of equations. Recent research efforts on *spectral graph sparsification* allow computing nearly-linear-sized subgraphs or sparsifiers (i.e., the number of edges is similar to the number of nodes in the subgraph) that can robustly preserve the spectrum (i.e., eigenvalues and eigenvectors) of the original graph Laplacian. This leads to a series of “theoretically nearly-linear-time” numerical and graph algorithms for solving sparse matrices, graph-based semi-supervised learning, spectral graph clustering, and max-flow problems [41, 70, 120, 124, 188–192]. However, the long-standing question of whether there exists a practically efficient spectral graph sparsification algorithm for tackling general large-scale, real-world graphs still remains. For instance, the state-of-the-art nearly-linear time spectral sparsification methods leverage Johnson-Lindenstrauss Lemma to compute effective resistances for the edge sampling procedure [189]. This requires solving the original graph Laplacian multiple times, thus making them impractical for handling real-world big graph problems.

In Chapter 8, I present *spectrum-preserving sparsification* (SPS), a spectrum-preserving framework for sparsification and visualization of big graph data. This framework allows edge and node reduction, spectral clustering, and level-of-detail exploration to support the adaptive visual exploration of big graph data based on a nearly-linear time spectrum-preserving big graph sparsification framework.

# CHAPTER 2

# RELATED WORK

In this chapter, I will give an overview of previous works on analysis and visualization of scientific data. Although the main focus of my work lies in summarization and reconstruction, it is important to understand the state of the art in different subareas of scientific visualization and the problems I am tackling in this dissertation. Similar to Chapter 1, I will first discuss analysis and visualization of time-varying multivariate data, followed by works about vector field data, and graph data. Additionally, I will summarize recent developments using deep learning in the field of scientific visualization. Doing so, I will point out problems that are tackled in later chapters of this dissertation.

## 2.1 Analysis and Visualization of Volume Data

To analyze and visualize volumetric data sets, researchers have sought different kinds of methods to understand the structures of volumes. Among those, there are three different analysis techniques with a closer relation to our work. First, *distribution-based* methods focus on the distributions of certain properties of the volume and identify the salient structure based on their corresponding statistical characteristics. Second, *similarity-based* methods measure the similarity between volume representations such as isosurfaces and derive the representative ones based on their similarities. Third, *topology-based* methods analyze the topological structure of the volumes and highlight the structures corresponding to topological changes.

**Distribution-based methods.** Understanding the relationships between the volume distribution and the isosurfaces allows us to identify salient features. For instance, Tenginakai et al. [202] detected salient isosurfaces using local higher order moments (LHOMs). LHOMs are computed and plotted for different sample values for a semi-automatic selection. Scheidegger et al. [179] applied Federer's Coarea Formula to improve the isosurface statistics by weighting with the inverse gradient magnitude. Duffy et al. [55] developed a mathematical model for continuous functions and proved the convergence to continuous statistics for regular lattices. Pekar et al. [166] proposed to use Laplacian weighted histograms for salient isovalue detection. However, the distribution of a volume data set does not translate to the spatial relationship among surfaces extracted, which is the focus of this dissertation.

**Similarity-based methods.** Recent works often seek to measure the similarities between a set of sample isosurfaces and derive the structure of the entire volume. For example, Bruckner and Möller [24] evaluated the similarity between isosurfaces and organized them in the form of an ISM. The similarity between two isosurfaces is defined as the mutual information shared by the distance fields of the two isosurfaces. Representative isosurfaces are identified using the ISM, which stores all pairwise similarity values. Haidacher et al. [83] extended this approach to compare isosurfaces extracted from multiple volumes. Wei et al. [220] proposed a similarity measure between two isosurfaces based on intermediate level-set surfaces. The values on the intermediate surfaces are sampled from the volume and their entropy values are used to evaluate whether the level-set surfaces align well with the intermediate isosurfaces. Recently, Ma et al. [146] used a tensor-based perceptual distance measure that simulates the human visual system and employed *k*-means clustering to select representative isosurfaces for comparing different volumetric data compression approaches.

Ultimately, all of these methods share a bottleneck in calculating the distance fields, which is a common task in many graphics and visualization applications. As a common need of many applications, accelerating the distance field computation has been extensively studied [126, 132, 138, 182, 233]. Yu et al. [233] presented the parallel distance tree that distributes the workload to multiple processors guided by a coarse global distance tree. Each processor then constructs a local distance tree and derives the distance fields. To compute the distance field, the bounding volume hierarchy (BVH)-tree is often used to identify the closest points. Liu and Kim [138] proposed the multi-BVH that combines the octree and BVH-tree. The use of octree provides additional information to reduce the number of BVHs to be traversed. Karras [113] introduced a GPU-based method to construct BVH-trees in parallel, which is one of the fastest GPU solutions available.

In Chapter 3, I will explain how we used Karras' BVH-tree construction together with a GPU-based approximation scheme to drastically accelerate the ISM computation.

**Topology-based methods.** These methods extract structures that essentially characterize properties of space such as convergence, connectedness, and continuity, providing a concise description of the overall structure of a volume. Bajaj et al. [7] proposed the contour spectrum, an interface combining the contour tree together with a variety of isosurface statistics, such as area and enclosed volume. Bremer et al. [21] presented the cancellation tree for describing the simplification of a Morse-Smale complex. Each simplification step cancels a pair of critical points, i.e., minima and maxima. The cancellation tree encodes the simplification steps and provides the connections among critical points. They further extended this approach to the hierarchical merge tree, which is a tracking graph that describes the temporal evolution of features [22]. Carr et al. [29] proposed to use the contour tree to encode the nesting relationships among isosurfaces. It also serves as an interface that allows users to select contours for operations such as removal, evolution, and tracking. Correa et al. [45] introduced the topological spine that connects critical points along the steepest ascending or descending directions. In addition, it includes geometric and contour nesting information, providing better spatial reasoning.

Although rigorous, topology-based methods normally capture minute topological changes, which lead to a large number of isosurfaces for a volume with complex topological variations. This, however, may not always be necessary for users to understand the overall structure of the volume. In contrast, a small set of nearly equally spaced isosurfaces would be more amenable for user observation: each surface is distinct enough and they are mutually distant in the space. Such a set of isosurfaces could also be useful as a visual summarization of the underlying volume.

In Chapter 4, I will show our work on extracting such a set of nearly equally spaced isosurfaces.

### 2.1.1 Salient Isovalue Selection

Often, the goal of volume analysis is to select a set of salient isovalues. Similar to volume analysis, salient isovalue selection algorithms can be categorized into *statistics*-, *topology*- and *similarity*-based methods.

For statistics-based analysis, Bajaj et al. [7] introduced the contour spectrum to facilitate an easier choice of isovalue by displaying several statistics in a 2D curve in relation to the isovalue. Pekar et al. [166] proposed a fast one-pass algorithm to analyze volumetric data sets and find intensity transitions between different surface materials based on gray-value histograms. Carr et al. [28] showed that histogram is inferior to statistics like area, or approximation of those, e.g., triangle count, in terms of representing the underlying function distribution. Meyer et al. [154] revisited the relationship between histograms and the underlying geometric structure of a volume data set. They used geometric measure theory to establish a convergence between the area statistics and the histogram.

For topology-based analysis, a prominent technique is using contour trees to inspect the development of underlying geometry based on the change of isovalue. Carr et al. [27] showed that contour trees can be computed in all dimensions, while Pascucci et al. [165] introduced branch decomposition to simplify the analysis of contour trees. Carr et al. [29] introduced simplification based on local geometric measures of feature importance to allow an interactive and flexible selection of isovalues.

For similarity-based analysis, together with ISM, [24] introduced a priority-ranking algorithm for isovalues based on the similarity maps. Other works have extended their approach to multimodal analysis [83]. A shortcoming of these methods is that they all limit their selection to a single or multiple isovalues for the whole volume.

In Chapter 6, I will show how our approach of detecting blobs in plasma fusion is related to the problem of salient isovalue selection. However, it differs in that the outputs of salient isovalue selection are a levelset of a given threshold in the entire scalar field. I instead need to identify salient local contours defined by different levels with different thresholds.

### 2.1.2 Analysis and Visualization of Time-Varying Multivariate Data

**Data analysis and visualization.** Analysis and visualization of time-varying multivariate data is an important yet challenging topic in scientific visualization [115, 147]. For time-varying data, researchers have studied efficient data organization and rendering based on spatiotemporal coherence [184, 210, 212, 214], transfer function specification [110], direct rendering of multiple time steps into a single image [224], and in situ visualization [232]. They also experimented with illustration-inspired [112] and importance-driven [213] techniques for visualizing time-varying data. For multivariate data, researchers have investigated query-driven visualization using compound range queries [196] and fuzzy queries using textual pattern matching [71]. They also explored variable correlation including point-wise correlation coefficients [34, 75, 197] and gradient similarity measure [178], causal relationships between variables using transfer entropy [215], variable grouping using mutual information (surprise and predictability) [14], and information flow between variables based on association rules (informativeness and uniqueness) [140].

In Chapter 5, I will show how we investigate the relationships among different volumes of time-varying multivariate data by extracting representative isosurfaces from each volume based on its similarity map and computing the similarity maps among representative isosurfaces from volumes of different time steps and variables.

**Visual abstraction and interface.** Time-varying multivariate data are high-dimensional and complex, which poses a great challenge to effective visual exploration and comparison. The conventional way of only visualizing these data in the original space-time view does not solve the inherent occlusion and clutter problem, nor does it offer a viable solution for data feature or relationship selection, exploration, and tracking. Therefore, researchers have studied various alternatives that map data and their relationships to lower-dimensional, often abstract spaces and representations to assist relationship overview, interaction, and navigation [209, 211]. Examples include the tri-space (i.e., spatial, temporal, and variable) interface that allows exploration of the temporal and variable dimensions of data along with the spatial context [3], multifield-graphs for studying correlation fields in a hierarchical manner [178], interactive storyboard for overall time-varying data visualization and feature exploration [145], attribute cloud that enables users to examine high-dimensional multivariate attributes in a 2D space [109], recurrence plot that shows the temporal similarity between two time-dependent signals as a matrix for recurrence detection [6, 68, 151], AniViz for animation creation from time-varying multivariate volume data [4], TransGraph for investigating the transition relationships in time-varying data [77, 78], multivariate transfer function interface design that tightly couples parallel coordinates plots and dimension projection plots [79], and iTree for compacting, indexing, and exploring time-varying data [76].

Chapter 5 describes the visual interface that we have designed, which organizes the MISM for an overview, filtering, and summarization, as well as detailed examination of volume relationships over spatial, temporal, and variable domains.

**Comparative visualization.** To study data similarities and differences, one can employ comparative visualization to compare different time steps, spatial locations, data variables, or modalities [115]. Gleicher et al. [72] proposed a general taxonomy of visual designs for comparison in information visualization based on the three building blocks of *juxtaposition*, *superposition*, and *explicit encodings*. Verma and Pang [207] pointed out that scientific data can be compared at the *image*, *data*, or *feature* level, depending on the level of data abstraction. They studied comparative visualization tools for analyzing flow fields which allow the comparison of individual streamlines and stream ribbons as well as a dense field of streamlines. Woodring and Shen [223] developed a volume shader for time-varying multivariate data visualization. The shader allows users to select multiple data volumes to create comparative visualization along with the presence of contextual information. Schneider et al. [181] presented a solution for interactive comparison of scalar fields using isosurfaces. They defined features in two scalar fields by the largest contour segmentation and matched these features using a similarity measure based on their spatial overlap. A thumbnail gallery of feature pairs and a graph representation are used to show all relationships between individual contours. Lampe et al. [129] designed a curve-centric volume reformation technique to create compelling comparative visualizations. The technique deforms the volume surrounding a curve and preserves the spatial neighborhood to the curve, thus permitting arc-length parameterized data visualizations in parallel for comparison. Malik et al. [149] presented a comparative visualization technique that uses a multi-image view and an edge explorer for dimensional measurement using 3D X-ray computed tomography. Their generic solution can be applied to other application areas such as parameter study of imaging modalities and detection of artifacts in data sets. Schmidt et al. [180] designed a comparative visualization tool named VAICo for visualizing similarities and differences in large image sets. This tool not only preserves contextual information but also enables detailed analysis of subtle variations.

Our purpose is to investigate the evolution of time-varying data in the context of multiple variables using surface-based visual representation. In Chapter 5, I will explain how we achieve this goal by clustering and filtering a large number of ISMs organized as a matrix along with comparative surface summarization and rendering techniques that reveal the spatial relationships of the corresponding isosurfaces.

## 2.2 Analysis and Visualization of Flow Data

Flow visualization has been the topic of an ample amount of previous works. As this dissertation focuses mainly on one particular aspect, namely, vector field reconstruction, I refer interested readers to the overview of texture-based [130], integration-based [153], topology-based [131], partition-based [175], illustration-based [19], and surfaced-based [61] flow field visualization techniques.

### 2.2.1 Vector Field Reconstruction

Vector field reconstruction has been studied extensively, starting in the 1990s. Mussa-Ivaldi [158] presented a least-square based method to reconstruct 2D vector fields from samples. Gouesbet and Letellier [73] used a multivariate polynomial approximation for global vector field reconstruction of time-continuous dynamical systems. Another algorithm to reconstruct 2D vector fields was introduced by Lage et al. [128]. They used a sparse set of samples to first fit a polynomial locally to describe each velocity component and then globally approximated the vector field using the partition of unity. Letellier et al. [136] extracted topological features to characterize and reconstruct 3D vector fields for experimental electrochemical systems. Chen et al. [38] applied triangulation to reconstruct velocities from samples along streamlines and then linearly interpolated the grid points of the vector field from the triangulation. Xu et al. [228] used Shannon's entropy to compute the information content of every voxel in a vector field to guide the streamline seeding process. Xu and Prince [227] introduced *gradient vector flow* (GVF), which has been used for vector field reconstruction from a set of representative streamlines by Tao et al. [199]. This two-stage algorithm recovers a vector field from streamlines by first estimating velocities for voxels with streamlines passing and then interpolating the missing voxels by minimizing the Laplacian over the whole vector field. Recently, Han et al. [86] reconstructed steady vector fields from streamlines using deep learning. As deep learning is an emerging technique to successfully tackle problems in many areas of computer science there has also been some work in flow visualization recently, which I discuss next.

In Chapter 7 I will analyze three different deep learning solutions to reconstruct VFD for unsteady flow simulations.

## 2.3 Deep Learning Techniques

In recent years, deep learning techniques have established themselves as a promising choice of technique to tackle large scale problem. With the ability to analyze a large amount of data and generate a non-linear correlation to solve a task, they have outperformed many state-of-the-art algorithms. The computer vision and natural language processing communities are at the forefront of this research, while the visualization community has just started to witness the application of deep learning techniques to solve visualization problems. In this dissertation, I show the application of two such techniques to reconstruct missing time steps of unsteady flow fields and to highlight features as isocontours in a time-varying plasma fusion simulation.

### 2.3.1 Deep Learning in Scientific Visualization

With deep learning techniques emerging as strong candidates to tackle problems in different areas of research, they recently have also found their applications in the domain of scientific visualization. Part of these works has tackled problems related to volume visualization. Zhou et al. [238] applied a *convolutional neural network* (CNN) for volume upscaling. They adopted techniques from image super-resolution [54] and trained their network using downsampled input volumes to reconstruct the original ones. Shi and Tao [185] proposed a framework to estimate viewpoints for volume visualization. Their framework also uses a CNN to extract the viewpoint parameters from a rendered image. Berger et al. [12] used a *generative adversarial network* (GAN) to generate and guide the selection of a viewpoint and the design of transfer functions for volumetric data sets. Cheng et al. [39] presented a deep-learning approach to assist feature extraction from complex structures for volume visualization, which is typically difficult to detect with conventional methods. Porter et al. [169] established a CNN to select representative time steps for time-varying multivariate data. Han and Wang [85] designed TSR-TVD, a recurrent generative framework to interpolate intermediate volume data for time-varying volumetric data. He et al. [93] proposed InSituNet, a neural network that synthesizes rendering images to allow parameter space exploration of ensemble simulations. Weiss et al. [221] generated high-resolution isosurface rendering from low-resolution ones by applying an image-based deep learning framework.

For flow visualization, Hong et al. [97] employed *long short-term memory* (LSTM) to estimate the access pattern for parallel particle tracing. Han et al. [87] clustered streamlines and surfaces automatically via a deep learning framework based on autoencoders. Eckert et al. [59] presented ScalarFlow, a framework to reconstruct accurate physics-based data from a small number of videos. Prantl et al. [170] proposed two-stage deformation-aware neural networks to learn the weighting and synthesis of dense volumetric deformation fields in the space-time representation of physical surfaces from liquid simulations. Kim et al. [117] synthesized fluid simulations from a set of reduced parameters using a GAN model. Doing so, they designed a novel loss function that guarantees divergence-free velocity fields. Guo et al. [80] designed CNNs that generate super-resolution of 3D VFD in the spatial domain with a scaling factor of 4 or 8.

### 2.3.2 Image Segmentation Using Neural Networks

The method used in Chapter 6 is inspired by U-Net [173], a deep CNN for medical image segmentation. U-Net detects cells in light microscopy images by downsampling and upsampling an input image while copying over features between the downward and upward phases. It was later adapted to work in 3D [43, 155]. Other approaches use hierarchical cascaded models [183] or create semantic segmentation of objects [143]. Nabla-net [152] uses an encoder-decoder framework or biomedical image segmentation. Other researchers proposed interactive segmentation using CNNs [216, 217] to guide users during a segmentation task.

For blob detection, we cannot directly use a segmentation technique to identify salient local contours. The typical output in such a task is a pixel-wise segmentation mask which often does not encompass additional information. In our case, we need super-levelsets that are encircled by closed isocontours, which is not necessarily the case in the output of a straightforward segmentation. However, adopting such an approach to extract isocontours instead of regions with different values can overcome this as shown in Chapter 6.

### 2.3.3 RNN and CNN for Sequence Learning

Recurrent learning techniques use an input series and adapt a set of inner states to capture temporal information from the input series. The widest-used form of *recurrent neural network* (RNN) is LSTM [95]. LSTM uses three different gates to determine how the information from the input and previous states influence the output. LSTM has previously been used to tackle many problems, for example, anomaly detection in time series [148], speech recognition [74], and human trajectory prediction in videos [5], just to name a few. With an adaptation to the LSTM, *convolutional long short-term memory* (ConvLSTM) [186] uses the convolution operation and Hadamard product to replace the fully-connected operation inside the LSTM in order to consider spatial features, making it a spatiotemporal component for RNNs. Pascanu et al. [164] pointed out two problems with RNN training, namely gradient vanishing and gradient exploding. They further proposed gradient norm clipping as a solution. Venugopalan et al. [206] used stacked LSTM to combat these problems.

Many impressive results for sequence learning tasks have been achieved using CNNs. Niklaus et al. [163] introduced CNN to perform frame interpolation. Their network learns a kernel from input frames and then uses it to generate missing frames. Nguyen et al. [161] introduced a deep linear embedding model for intermediate frame interpolation. Their framework first extracts features from each frame and then linearly interpolates the intermediate frames in the feature space before reconstructing the corresponding frame from the interpolated features. Jiang et al. [111] employed CNNs to estimate the optical flows in both directions from two given frames. They then generated in-between frames based on these optical flows. Yu et al. [231] presented QANet, a CNN and self-attention based solution for reading comprehension tasks. Their framework is seven times faster and achieves similar performance compared with RNN.

The work presented in Chapter 7 differs from these works mentioned above. First, we use CNNs for vector field interpolation employing skip connection to incorporate temporal information. Second, we consider the different value ranges of the three vector components in 3D vector fields. Third, in contrast to video interpolation tasks where content loss and perceptual loss are most important, we approximate magnitude loss and employ angle loss to facilitate the generation of high-quality synthesized vector fields for unsteady flow.

## 2.4 Graph Visualization

Graphs are commonly used to describe the interconnectedness of multiple entities to analyze the underlying network. To visualize graphs, the most common technique is to use a node-link diagram with entities being depicted as nodes and their relationships as links between them. While this representation allows an intuitive understanding of the matter, the ever-growing size of graphs poses the challenge of generating a layout that not only reveals information efficiently but is also visually pleasing. The main challenges to graph visualization are scalability, readability, and aesthetics. As graph visualization is a vastly studied topic, I focus on the main related works here and refer the interested reader to the following surveys [11, 50, 82, 94, 211].

### 2.4.1 Graph Layouts

There are many different ways of generating a graph layout, for example, force-directed layouts [69], geometric layouts [53], or spectral layouts. Force-directed layouts use a physics-based simulation that assigns attractive and repulsive forces between pairs of nodes, aiming to minimize the energy function of these forces. In Chapter 8, I focus on spectral layouts.

### 2.4.2 Spectral Graph Drawing

Among the spectral methods for graph drawing, the *eigen-projection* method uses the first few nontrivial eigenvectors of the graph Laplacian matrix or the top dominant eigenvectors of the adjacency matrix. Hall [84] used the eigenvectors of the Laplacian to embed graph vertices in a space of arbitrary dimension. The entries of the $k$ eigenvectors related to the smallest nonzero eigenvalues are used as a node's coordinates. This is referred to as $k$-dimensional graph spectral embedding. Pisanski and Shawe-Taylor [168] took Hall's method to generate pleasing drawings of symmetrical graphs such as fullerene molecules in chemistry. Brandes and Willhalm [20] used eigenvectors of a modified Laplacian to draw bibliographic networks. Note that for *regular* graphs (where every node has the same degree), the eigenvectors of the Laplacian equal those of the adjacency matrix, but in a reversed order. This is not the case for *non-regular* graphs. Using the Laplacian is advantageous as it is rooted in a more solid theoretical basis and gives better results than those obtained using the adjacency matrix.

Koren et al. [122, 123] proposed *algebraic multigrid computation of eigenvectors* (ACE), an extremely fast algorithm for drawing very large graphs. ACE identifies an optimal drawing of the graph by minimizing a quadratic energy function, which is expressed as a general eigenvalue problem and efficiently solved using fast algebraic multigrid implementation. Harel and Koren [89, 90] designed *high-dimensional embedding* (HDE) for aesthetic drawing of undirected graphs. HDE first embeds the graph in a very high dimension and then projects it into the 2D plane using principal component analysis. This algorithm is fast, exhibits the graph in various dimensions, and supports interactive exploration of large graphs. Koren [121, 122] presented a modified approach that uses *degree-normalized eigenvectors* to achieve aesthetic graph layouts. The degree normalized eigenvectors adjust the edge weights to reflect their relative importance in the related local scale. As such, the modified solution can allocate each cluster an adequate area in the drawing and avoid drawing extremely dense clusters. Hu et al. [101] designed a spectral graph drawing algorithm that includes *node projection*, *node dispersion*, and *sphere warping*. They first projected nodes onto a $k$-dimensional sphere, then dispersed nodes around the sphere's surface to separate apart densely connected clustered nodes, and finally warped the $k$-dimensional sphere's surface to a 2D space using multidimensional scaling. Their algorithm can clearly show the topology and community structures of the graph.

Most spectrum-based graph visualization techniques [20, 89, 122, 123, 168] only place their focus on graph layout. Besides drawing the graph using spectral sparsification, we integrate spectral clustering and edge bundling to help users better examine the graph for effective visual understanding. This is particularly important when handling big graph data as visual understanding of the complex and diverse graph relationships is the key.

### 2.4.3 Spectral Methods for Graph Applications

To address the computational bottleneck of spectral methods in graph-related applications, recent research efforts aimed to reduce the complexity of the original graph Laplacian through various kinds of approximations. For example, k-nearest neighbor (kNN) graphs maintain $k$ nearest neighbors for each node, whereas $\varepsilon$-neighborhood graphs keep the neighbors within the range of distance $\varepsilon$ [157]. Williams and Seeger [222] introduced a sampling-based approach for affinity matrix approximation using the Nyström method, while its error analysis has been proposed in [235]. Chen and Cai [37] presented a landmark-based method for representing the original data points for large-scale spectral clustering. Yang et al. [229] proposed a general framework for fast approximate spectral clustering by collapsing the original data points into a small number of centroids using k-means or random-projection trees. Liu et al. [139] introduced a method for compressing the original graph into a sparse bipartite graph by generating a small number of "supernodes". Satuluri et al. [177] proposed a graph sparsification method for scalable clustering using a simple similarity-based heuristic. However, existing graph approximation methods cannot efficiently and robustly preserve the spectrums of the original graphs, and thus may lead to degraded or even misleading results. Recently, spectral perturbation analysis was applied to spectral graph sparsification and reduction in order to reduce the graph to nearly-linear-sized with high spectral similarity [64, 65, 237]. This progress makes it possible to develop much faster algorithms such as the symmetric diagonally dominant (SDD) matrix solvers [236] as well as spectral graph partitioning algorithm [65]. Note that these recent works on graph sparsification [64, 236, 237] only address spectral graph simplification but not spectral graph drawing using a multilevel approach.

In Chapter 8, I will detail how we apply spectral methods to efficiently simplify a given input graph for visualization.

### 2.4.4 Quality Metrics for Graph Sampling

An important question for graph sampling is how to evaluate the quality of the simplified graph. To evaluate the similarity between the original and sampled graphs, Hu and Lau [100] employed three metrics: (1) *total variation distance* which measures all the difference between two distributions; (2) *Kullback-Leibler divergence* which captures the difference between the two distributions accounting for the bulk of the distributions; and (3) *Kolmogorov-Smirnov statistic* which captures the maximum vertical distance of the cumulative distribution function of the two distributions. Zhang et al. [234] computed seven statistical properties, namely, *degree distribution*, *betweenness centrality distribution*, *clustering coefficient distribution*, *average neighbor degree distribution*, *degree cen-*

*trality distribution*, *edge betweenness centrality distribution*, and *hop distribution*, to quantitatively compare different graph sampling methods. Recently, Hong et al. [98] used five metrics, namely, *degree correlation assortativity*, *closeness centrality*, *clustering coefficient*, *largest connected component*, and *average neighbor degree*, to evaluate their graph sampling methods, which improve random-based sampling by considering the block-cut tree.

A problem with the above statistical metrics and properties is that they are not well-suited to capture the visual quality of the corresponding graph layout. This is especially the case for large social and biological networks where nodes and edges could easily become "blobs" in the drawing of dense graphs with a few hundred vertices or sparse graphs with a few thousand vertices. Wu et al. [226] pointed out that quality metrics based on statistical or topological properties do not translate to visual quality. Their study shows that three visual factors significantly influence the representativeness of sampled graphs: *cluster quality*, *high degree nodes*, and *coverage area*. Eades et al. [57] proposed a shape-based quality metric for large graph visualization by treating the quality of a drawing $D$ of a graph $G$ as the similarity between $G$ and the "shape" of the set of vertex locations of $D$. Nguyen et al. [162] generalized this metric to compare proxy graphs using the shape-based quality metric. In our work, we use this so-called *proxy quality metric* to evaluate the graph after spectral edge sparsification (where only edges are removed) and employ statistical metrics to further evaluate the graph after multilevel node reduction (where nodes are aggregated to form pseudo-nodes).

# CHAPTER 3

# EFFICIENT GPU-ACCELERATED COMPUTATION OF ISOSURFACE SIMILARITY MAPS

In this chapter, I present an approximation scheme to accelerate the ISM computation. This is achieved by avoiding extracting full isosurfaces and instead only evaluate sample points as the first step. This not only allows for a quicker computation in the first stage but also drastically reduces the points for the distance field computation in the second stage. The distance fields are then sped up using a parallel BVH-tree on the GPU. This allows for a much faster computation of ISM and paves the way for subsequent large scale analysis.[1]

## 3.1 Our Approach

Our approach to generating ISMs consists of three steps: sampling a set of isosurfaces, calculating distance fields, and computing the similarity map. First, we sample over the value range to generate a set of isosurfaces. Instead of computing the actual surfaces, we approximate each surface with a loosely ordered set of points for performance gain. Second, we compute a distance field for every sampled isosurfaces. We leverage BVH-trees for fast queries of the closest points and use a heuristic approach to improve the performance of traversing the tree based on the loose order of points. Finally, we calculate the similarity of every pair of sampled isovalues and organize all similarity values in a similarity map. The similarity between two sampled isovalues is given by the mutual information of their corresponding distance fields.

[1]This work was published as M. Imre, J. Tao, and C. Wang. Efficient GPU-accelerated computation of isosurface similarity maps. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 180–184, 2017

### 3.1.1 Isosurfaces Approximation

Isosurfaces are usually generated using the marching cubes algorithm [144], which is not only costly but also unnecessary for distance field calculation. The marching cubes algorithm computes intersection points of the isosurfaces with each voxel and connects these points for the exact surface. However, to calculate a distance field with reasonable precision, it is unnecessary to compute multiple points for one voxel or know the connections of those points.

Our approach approximates an isosurfaces $S$ corresponding to a value $v$ by generating one point for each 3D grid cell containing a part of $S$. A grid cell is a $1 \times 1 \times 1$ cube whose vertices are the eight voxels in a $2 \times 2 \times 2$ spatial region. Whether a grid cell contains a part of $S$ is determined by checking the values of the eight voxels corresponding to it. If some of the values are larger than $v$ and some are smaller than $v$, linearly interpolated values at some points in the grid cell must be $v$, which means that the grid cell contains a part of $S$. In this case, the center of this grid cell will be recorded as a sample point of $S$ (for efficiency, we use the center instead of computing the actual intersection point). Otherwise, the grid cell does not contain any part of $S$ and will be ignored. Our implementation leverages the parallelism of GPU to scan the grid cells simultaneously. Each GPU thread checks a grid cell and stores the result in a binary volume. Then we employ a GPU compaction scheme to convert the binary volume into an array of point coordinates (i.e., the approximation of an isosurfaces). The points are listed in the scanline order, providing loose spatial relationships among points.

Note that the error of using this approximation to compute the distance from a point $p$ to a surface $S$ is bounded by $\sqrt{3}/2$ (half the diagonal length of the grid cell) due to the triangle inequality. This error is acceptable, since downsampled distance fields are usually used to compute the similarity map [24]. In Figure 3.1, we can see that our approximation (c) yields a very similar similarity map as the exact marching cubes algorithm does (a). We further select three representative isosurfaces for each similarity map, which are shown in

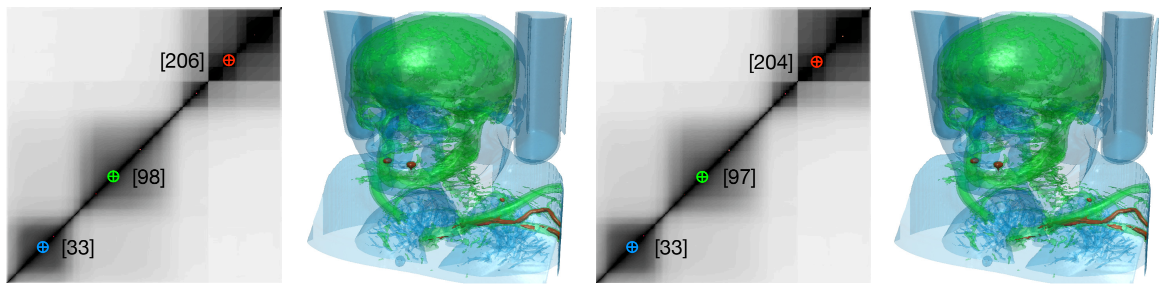


Figure 3.1. Comparing similarity map and representative isosurfaces generated using actual isosurfaces ((a) and (b)) against our approximation ((c) and (d)) with the CT data set. The isosurfaces are colored in the descending order of their importance: red, green, blue. The number in the brackets indicates the index of sampled isovalue. The marching cubes algorithm is used to extract the actual surfaces shown in (b) and (d) for given representative values.

Figure 3.1 (b) and (d), respectively. Although there are slight differences in the selected isovalues, those are barely visible in the surface rendering. This minor difference is neglectable as the main goal of selecting the most salient values and depicting their surfaces is still achieved.

### 3.1.2 Distance Field Computation

A distance field of an isosurfaces is a volume with every voxel containing the distance from the center of this voxel to the isosurfaces. In order to compute this, it is necessary to find the closest point of an isosurfaces for every voxel in the distance field. To decimate the search space for this problem, we downscale the resulting distance field. A typical solution for such a search problem is a tree structure. In our approach, we leverage BVH-trees to accelerate the search for the closest point. A BVH-tree is a binary tree used to subdivide a scene into a tree structure. Each node of a BVH-tree consists of a bounding volume that includes all of its descendants. A leaf consists of a single primitive — in our case a single point. Although other bounding volumes increase the traversal speed, we use bounding boxes for lower construction time and apply Karras's algorithm [113] to build BVH-trees in parallel. This is essential since a tree needs to be built for every isosurface's approximation.

To compute the distance fields of a given isosurface's approximation, we spawn a CUDA thread to search the closest point in the BVH-tree for each voxel by traversing the tree in a depth-first search manner. During the traversal for a voxel $v$, we maintain an upper bound of the closest distance from $v$ to the isosurfaces. The sub-tree rooted at an internal node $n$ will not be traversed, if the distance $d(v,n)$ between $v$ and $n$ is larger than the upper bound. The distance $d(v,n)$ between a voxel $v$ and an internal node $n$ is given by the minimum distance from $v$ to the bounding box of $n$. The distance $d(v,n)$ between a voxel $v$ and a leaf $n$ is given by the distance between $v$ and the corresponding point of $n$. Specifically, the traversal procedure can be described in the following four steps:

1. Take a set of sample points $P$ and calculate the initial upper bound $d_u = \min_{p \in P} d(v,p)$.
2. Add the root to a stack.
3. Pop the first node $n$ from the stack. If $n$ is an internal node, for each of its child node $n_c$, compute the distance $d(v,n_c)$. If $d(v,n_c) < d_u$, push $n_c$ onto the stack. If $n$ is a leaf, compute the distance $d(v,n)$ between $v$ and $n$, and update $d_u$ with $\min(d_u, d(v,n))$.
4. Repeat Step 3 until the stack becomes empty.

For an efficient traversal, a good initial upper bound is needed. Unlike the existing approaches, which usually initialize the upper bound when the first leaf is encountered, we estimate the upper bound before the traversal by computing the distance from the voxel to a set of sample points. Note that the points approximating an isosurfaces are listed in the scanline order. By evenly sampling the approximating points in that array, we obtain a set of sample points that is roughly evenly spaced over the isosurfaces. This provides a tighter estimation of the initial upper bound, allowing more branches to be pruned, especially at the beginning stage of the traversal.

### 3.1.3 Similarity Map Generation

The similarity between two isosurfaces are measured by the mutual information of their corresponding distance fields. To calculate the mutual information, we compute a joint histogram of the two distance fields, where each entry $(i, j)$ in the joint histogram contains the number of voxels that fall into bins $i$ and $j$ in the first and second distance fields, respectively. Then, the mutual information can be derived from the joint histogram. In our GPU implementation, we spawn a CUDA thread for each pair of sampled isosurfaces. Each thread reads two distance fields, computes the joint histogram and mutual information, and stores the value of mutual information in the similarity map.

## 3.2 Results and Discussion

We used multiple data sets with different characteristics to benchmark our application on a desktop with an Intel i7-4790 quad-core CPU running at 3.6 GHz, 32 GB main memory, and an NVIDIA GeForce GTX 760 GPU with 2 GB memory. Throughout the evaluation we used Bourke's [18] implementation of the marching cubes algorithm as a baseline. Table 3.1 shows the total time (in seconds) spent in each step of the similarity map generation pipeline. The timing results were collected with a GPU implementation for all 256 isovalues using each data set.

It took us under 2.5 minutes to generate an isosurfaces similarity map for most of these data sets. The only exception is the Combustion data set, which has the largest volumes and the greatest numbers of points in the surfaces. As we can see from Table 3.1, the three variables (*CHI*, *mixture fraction*, and *vorticity*) of the Combustion data set have the same dimensions and cost similar time to approximate the isosurfaces. But these variables have a high fluctuation in the number of points in surface approximation, which leads to different running times to compute the distance fields of the same size.

TABLE 3.1

TIMING AND PARAMETERS ISOSURFACE APPROXIMATION

| | | time | | surface | | distance | similarity | marching cubes | | approximation | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| data set | variable | step | dimensions | points | approx. | field | map | surface | DF | surface | DF |
| Ionization | GT | 75 | $600 \times 248 \times 248$ | 148,213 | 78.12 | 9.54 | 35.45 | 2607 | 14.50 | 590 | 9.54 |
| Ionization | GT | 146 | $600 \times 248 \times 248$ | 185,897 | 78.02 | 10.27 | 37.35 | 2617 | 12.67 | 590 | 10.27 |
| Ionization | H | 150 | $600 \times 248 \times 248$ | 156,086 | 77.85 | 7.32 | 21.10 | 2586 | 9.93 | 590 | 7.32 |
| Combustion | CHI | 45 | $480 \times 720 \times 120$ | 1,984,240 | 90.14 | 57.92 | 45.66 | 4033 | 69.91 | 695 | 57.92 |
| Combustion | CHI | 90 | $480 \times 720 \times 120$ | 2,823,210 | 89.45 | 82.53 | 46.05 | 4524 | 102.77 | 683 | 82.53 |
| Combustion | mixture fraction | 45 | $480 \times 720 \times 120$ | 375,039 | 88.06 | 16.69 | 43.17 | 3085 | 20.63 | 666 | 16.69 |
| Combustion | mixture fraction | 90 | $480 \times 720 \times 120$ | 495,256 | 87.85 | 20.45 | 43.72 | 3137 | 25.05 | 666 | 20.45 |
| Combustion | vorticity | 45 | $480 \times 720 \times 120$ | 1,462,340 | 88.56 | 42.62 | 40.55 | 3710 | 52.62 | 669 | 42.62 |
| Combustion | vorticity | 90 | $480 \times 720 \times 120$ | 1,980,120 | 88.87 | 54.55 | 41.24 | 4031 | 66.72 | 672 | 54.55 |
| CT | – | – | $256 \times 256 \times 230$ | 248,607 | 32.90 | 8.05 | 17.16 | 1199 | 10.02 | 243 | 8.05 |
| Hurricane | precipitation ratio | 17 | $500 \times 500 \times 100$ | 639,957 | 53.81 | 16.28 | 24.21 | 2075 | 19.60 | 402 | 16.28 |
| Hurricane | precipitation ratio | 36 | $500 \times 500 \times 100$ | 790,787 | 53.97 | 20.36 | 25.06 | 2164 | 24.68 | 402 | 20.36 |
| Hurricane | water vapor ratio | 17 | $500 \times 500 \times 100$ | 408,140 | 53.56 | 11.98 | 25.99 | 1959 | 15.66 | 402 | 11.98 |
| Hurricane | water vapor ratio | 36 | $500 \times 500 \times 100$ | 448,670 | 53.67 | 12.91 | 26.15 | 1975 | 16.46 | 402 | 12.91 |

All timing results are in seconds and for all isosurfaces of 256 sampled isovalues, “surface points” shows the average numbers of points per isosurface, distance fields were downscaled by 8 in each dimension. We used 1500 sample points prior to the traversal. The last columns of “surface” and “DF” show the running time for surface computation and distance field calculation, respectively.

Generally, for all data sets, we can observe that the time for surface approximation is proportional to the size of volumes and the time for distance field computation depends on both the dimensions and surface points. Furthermore, the traversal time for the BVH-trees highly depends on the choice of sample points. For this evaluation we opted to choose 1500 as this tends to yield a swift traversal time for all of the data sets.

To demonstrate the quality of the isosurfaces similarity maps produced by our approach, we identified representative isosurfaces following [24]. Figure 3.2 shows the three most significant isosurfacess for certain data sets and variables in the first row and the corresponding similarity maps in the second row. The blue isosurfaces are the most important ones, followed by green and orange. The corresponding isovalues are highlighted in the similarity maps as well. For each variable, we can observe that the three selected isosurfaces capture the principal patterns. In addition, Figure 3.1 shows that we obtained similar similarity maps and representative isosurfaces as reported in [24]. With our acceleration solution and the modern graphics hardware, the performance is greatly improved, as the time to generate the similarity map is reduced from 22.1 minutes (refer to Table 1 in [24]) to less than one minute (refer to the CT data set in Table 3.1).

Furthermore, we examined the difference of similarity maps when applying different levels of downsampling for the distance fields. Specifically, we downscaled the distance fields by 8 along each dimension and compared this to a version with a downscale by 4. While the overall time to calculate 256 distance fields for a data set already shows a 3.5-fold increase, the similarity map computation suffered even more from increasing the accuracy. On average the similarity map calculation took 7.8 times longer than that with a lower resolution of distance fields. We computed the difference of every entry in the similarity maps using the two downscaling factors for all data sets listed in Table 3.1. Our results show an average difference of 1.5% using the lower resolution distance fields, which is fairly acceptable in exchange for significant performance gain. Further reducing the downsampling rate inevitably leads to an even higher increase in computation time

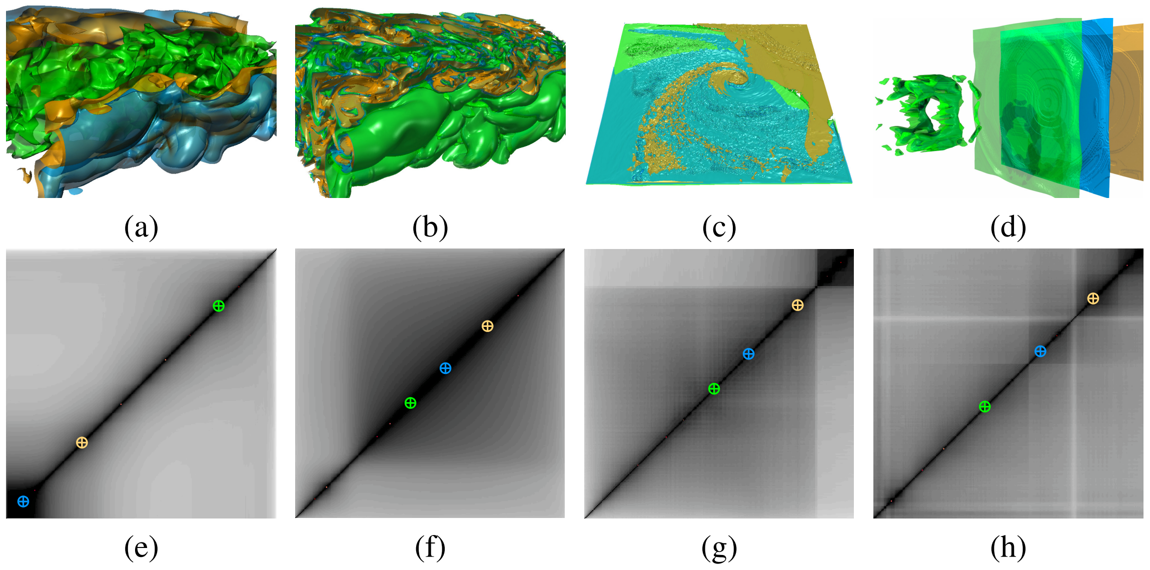


Figure 3.2. Rendering of the most important isosurfaces (a) to (d), with their corresponding similarity maps (e) to (h). The first and second columns show *mixture fraction* and *vorticity* of the Combustion data set, respectively. The third column shows *water vapor ratio* of the Hurricane data set, and the fourth column depicts *GT* of the Ionization data set.

while not improving the results by a reasonable amount.

### 3.2.1 Isosurfaces Approximation

With our approximation, we reduced the time for generating the surfaces points by an ample amount. Table 3.1 compares the running time of the marching cubes algorithm and our approximation both running on CPU. As we can see, our approximation outperforms the marching cubes algorithm, achieving an average speedup of 5.1 times for surface creation.

Another advantage of the approximation is that the points in the surface are loosely sorted, which allows our heuristic sampling method to find a tight initial upper bound for the tree traversal used in distance field computation. This increases the performance of distance field calculation by 20% on average. Although this only amounts to a difference of a few seconds in a process that takes around 10 minutes on average, once the GPU approximation is realized, this difference would lead to up to 10% of the total time before creating the similarity map.

### 3.2.2 Initial Upper Bound Estimation

We examined the impact of the number of samples to the performance of BVH-trees traversal when estimating the initial upper bound. As one could expect, the size of the sample point set has a high influence on the duration of the traversal. With more sample points, the time to estimate an initial upper bound increases, while the traversal time decreases due to a tighter upper bound. Figure 3.3 shows the average time spent on traversing the trees for a given number of sample points. Without an initial upper bound (i.e., zero sample point), the traversal takes the longest time, as the tree needs to be examined until the first leaf is reached. After that the average duration for the traversal falls rapidly in the beginning and then slowly approaches the minimums. Most data sets have their lowest traversal time between 2000 and 2300 points and just fluctuate by one millisecond.

Interestingly, the two variables *CHI* and *vorticity* of the Combustion data set reach their minimums early with just 1200 and 1500 sample points respectively. After that, the running time rapidly increases from 94.58 to 380.36 (*CHI*), and from 69.25 to 248.82 (*vorticity*). These two variables have higher numbers of points in surface approximation (refer to Table 3.1), which makes it surprising since we expect that in those cases a higher quality upper bound would be more beneficial. The rapid increases of the running time may be related to the different characteristics shown in the isosurfaces of these two variables. We compared the representative isosurfaces of the variables *mixture fraction* and *vorticity* that are shown in Figure 3.2 (a) and (b), respectively. Other than having tremendously more points per isosurfaces, the isosurfaces of *vorticity* distribute more evenly over the entire volume than those of *mixture fraction*. The same nature can be observed for isosurfaces

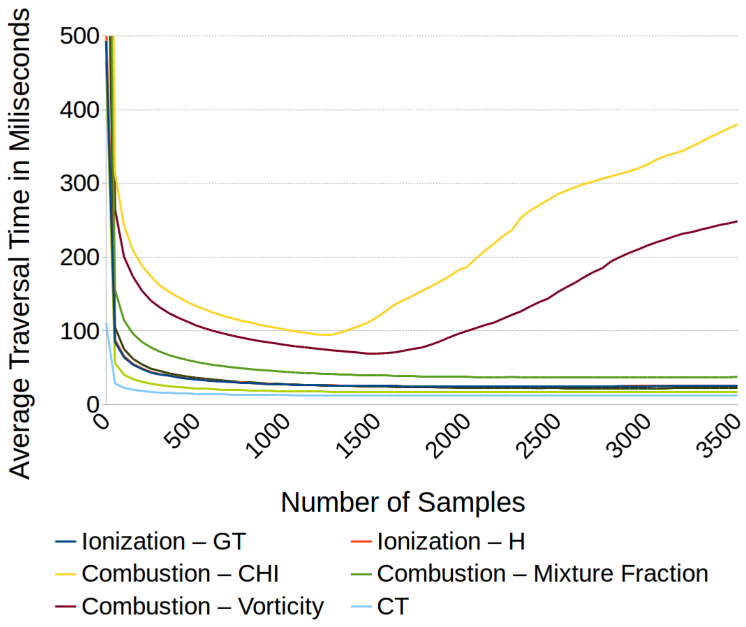


Figure 3.3. Average traversal time for different numbers of sample points. We sampled from 0 to 3500 points with a step size of 50.

of *CHI*. We suspect that the high number of points, coupled with their spatial distribution of *CHI* and *vorticity*, allows an easy identification of tight upper bounds, rendering more examination prior to the traversal unnecessary.

We expect a similar increase of the running time for the other data sets with higher numbers of sample points as this process converges toward a brute force approach. But the performance gained by using a tighter upper bound usually outweighs these costs during the traversal for less than 1500 sample points. As we can see in Figure 3.3, it is safe to take any value between 1000 and 1500 for the data sets we chose. In general, if the data sets examined are of a reasonable size, a value in that range will always be preferable over a smaller value for selecting the initial upper bound.

## 3.3 Conclusions

With our approach we manage to achieve a considerable speedup without impacting the quality when creating ISMs. Our isosurface approximation allows fast generation of the point sets needed for computing the distance fields with small bounded errors. Furthermore, we use BVH-trees to efficiently compute the distance fields. The loosely sorted points from our approximated surface generation enables the use of our heuristic for finding a tight upper bound before initiating the tree traversal. Our experiment demonstrates that the number of samples can be safely determined to achieve significant speedup. Our work allows users to extract the most important isosurfaces from a data set within a much short amount of time, opening up the possibility to perform a more thorough analysis employing multiple variables and time steps.

# CHAPTER 4

# IDENTIFYING NEARLY EQUALLY SPACED ISOSURFACES FOR VOLUMETRIC DATA SETS

In this chapter, I present a two-stage approach to identify nearly equally spaced isosurfaces. Using the accelerations from Chapter 3, we can quickly compute a distance between a set of isosurfaces. This allows us to get an almost equal spacing in the first stage by optimizing the distance to be as equal as possible to the average. In the second stage we refine these distances by adapting the isovalues of non-neighboring isosurfaces in an alternating fashion. Although the distance measures are not continuous, we still achieve nearly equally spaced isosurface. This allows for an overview of the data set or a better pre-sampling for other analysis tools.[1]

## 4.1 Our Approach

We propose a two-stage approach for finding nearly equally spaced isosurfaces. Both stages run over several iterations aiming for convergence. First of all, during the *estimation stage*, we measure the distance between every pair of neighboring isosurfaces and resample the isovalues based on these distances using linear interpolation. This stage, however, assumes piecewise linearity of the distance function between neighboring isosurfaces, which does not hold in general. In our experiment, it approaches approximate solutions in a few iterations but normally does not converge to the optimal solution. Therefore, we introduce the *refinement stage* that adopts a binary search strategy to adjust each isovalue so that

[1]This work was publish as M. Imre, J. Tao, and C. Wang. Identifying nearly equally spaced isosurfaces for volumetric data sets. *Computers & Graphics*, 72:82–97, 2018

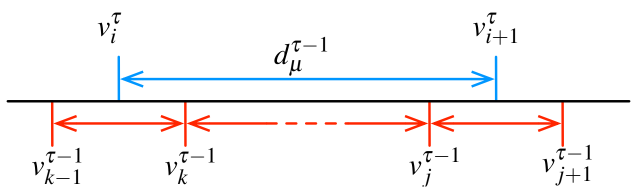


Figure 4.1. The estimation stage. Identifying isovalue $v_{i+1}^\tau$ based on its left neighbor $v_i^\tau$ and the distances evaluated at the previous time step $\tau-1$.

its surface has equal distance to its two neighbors. By repeating this process for several iterations, we achieve nearly equal distances between all neighboring isosurfaces.

In this chapter, we denote a set of isovalues at iteration $\tau$ as $V^\tau = \{v_1^\tau, v_2^\tau, \dots, v_n^\tau\}$, the isosurface corresponding to an isovalue $v_i^\tau$ as $s_i^\tau$, and the distance between two values $v_i^\tau$ and $v_j^\tau$ as the distance between their respective surfaces, $d_{i,j^\tau}$, or more generally, $d(v_i, v_j)$.

### 4.1.1 Estimation Stage

For the estimation stage, we start from a uniformly sampled set of isovalues $V^0$ and gradually adapt the isovalues based on the previous set of isovalues and their distances. Specifically, at each iteration $\tau$, we first approximate the isosurface using the approximation scheme introduced in Chapter 3 (refer to Chapter 4.1.4), $s_i^{\tau-1}$ of each isovalue $v_i^{\tau-1}$ at the previous iteration $\tau-1$ and evaluate the distance $d_{i,i+1}^{\tau-1}$ between every pair of neighboring isovalues $v_i^{\tau-1}$ and $v_{i+1}^{\tau-1}$ using their corresponding approximated isosurfaces. The average distance $d_\mu^{\tau-1} = (\sum_{i=1}^{n-1} d_{i,i+1}^{\tau-1})/(n-1)$ ($\mu$ stands for the average) is considered to be the target distance to achieve at the current iteration $\tau$.

Then, starting from the first isovalue $v_i^\tau = v_{\min}$, where $v_{\min}$ is the minimum isovalue, we resample each isovalue $v_{i+1}^\tau$ that has approximately the target distance $d_\mu^{\tau-1}$ to its left neighbor $v_i^\tau$, as illustrated in Figure 4.1. The distance is estimated under two assumptions. First, the distance between neighboring isovalues can be linearly interpolated. For example, the isovalue $v_i^\tau$ falls between two previous isovalues $v_{k-1}^{\tau-1}$ and $v_k^{\tau-1}$. We assume that the distance between $v_i^\tau$ and $v_k^{\tau-1}$ can be linearly interpolated using the distance $d_{k-1,k}^{\tau-1}$. Second, we assume that the distance can be added to estimate the distance between non-neighboring isovalues. For example, we assume that the distance $d_{k,j}^{\tau-1}$ can be obtained using the summation of all neighboring distances between $v_k^{\tau-1}$ and $v_j^{\tau-1}$, i.e., $d_{k,j}^{\tau-1} = d_{k,k+1}^{\tau-1} + \cdots + d_{j-1,j}^{\tau-1}$. In this way, we can iteratively identify the entire set of isovalues $V_\tau$. This process can also be considered as a parameterization based on the distances evaluated from previous neighboring isovalues.

We repeat this process for several iterations until a predefined minimum iteration number $\delta_e$ is reached and the variation of neighboring distances stops decreasing. As shown in the first two rows of Figure 4.5, we can see that the estimation stage approaches the desired solution within a small number of iterations. Note that the computation of distances between neighbors, which is the most costly step, can be performed in parallel for each iteration. As previously mentioned, this stage is unlikely to converge since the two aforementioned assumptions do not hold for many volumetric data sets. In most cases, it is more likely to have $d_{i,j} + d_{j,k} > d_{i,k}$ due to the triangle inequality. Therefore, the estimation stage only provides a rough solution, and an additional refinement stage is needed to obtain the optimal solution.

### 4.1.2 Refinement Stage

In the refinement stage, we advocate a binary search strategy: placing the candidate isovalue in the middle of its two neighbors to identify an isosurface having the equal distance to its two neighboring isosurfaces. The distance function in this stage neither assumes linearity nor violates the triangle inequality. Unlike the estimation stage, this stage provides a slower but more robust process of convergence. This is achieved by adjusting the odd-indexed isovalues and even-indexed ones alternatively, as shown in Figure 4.2 (a). Specifically, the refinement stage is performed in multiple steps ($\delta_r$). At odd steps, we adjust the red isovalues $v_{i-2}, v_i, v_{i+2}$ with odd indices (assuming $i$ is an odd number), so that they

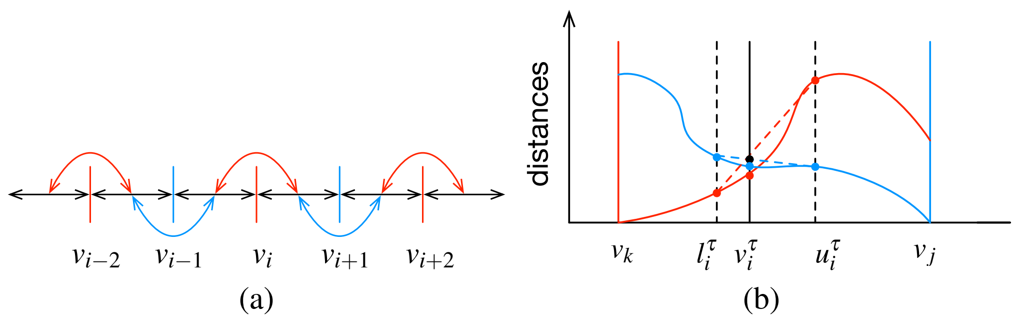


Figure 4.2. The refinement stage. (a) Adjusting odd-indexed and even-indexed isovalues alternatively. (b) Identifying an isovalue $v_i$ that has the equal distance to its two neighbors $v_k$ and $v_j$. The red (blue) curve in (b) represents the distance function from an isovalue in $[v_k, v_j]$ to $v_k$ ($v_j$).

have the equal distance to their neighbor isovalues, i.e., $d_{i-3,i-2} = d_{i-2,i-1}$, $d_{i-1,i} = d_{i,i+1}$, and $d_{i+1,i+2} = d_{i+2,i+3}$, as indicated by the red arrows. Since the blue isovalues with even indices are fixed at odd steps, each odd-indexed isovalue can be adjusted independently in parallel. At even steps, we adjust the blue isovalues in the same fashion. Note that the blue and red arrows connect all distances between neighboring isovalues, which leads to equally spaced isovalues when this stage converges.

In every step, we use several iterations ($\delta_\tau$) of a binary search strategy to identify an isovalue $v_i$ that has the equal distance to its two neighbors $v_{i-1}$ and $v_{i+1}$, as illustrated in Figure 4.2 (b). This means that the goal becomes finding one intersection point of the red and blue curves. At each iteration $\tau$, we maintain a lower bound $l_i^\tau$ and an upper bound $u_i^\tau$ that contain the intersection point between them. The lower and upper bound are initialized as $v_k$ and $v_j$, respectively, i.e., $l_i^0 = v_k$ and $u_i^0 = v_j$. The lower bound maintains a property that it is always closer to $v_k$ than $v_j$, i.e., $d_{k,l_i} < d_{l_i,j}$, and the upper bound maintains a similar property in the opposite way, i.e., $d_{k,u_i} > d_{u_i,j}$. Due to these properties, the red and blue curves must intersect somewhere in the middle as long as the distance functions are continuous.

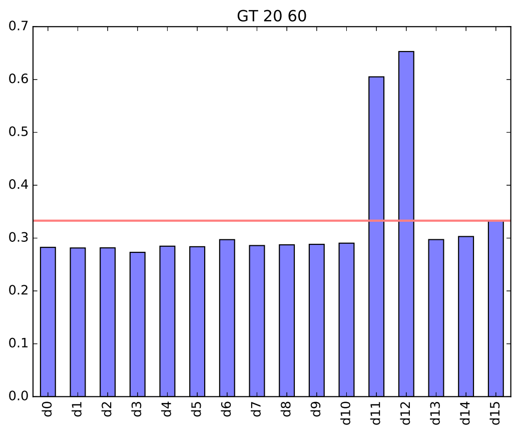
At each iteration, we assume that the two distance functions change linearly between the bounds, as shown by the blue and red dashed lines in Figure 4.2 (b), and compute the intersection point, as indicated by the black dot. This intersection point provides the new isovalue $v_i^\tau$ at step $\tau$, as shown by the black solid line in Figure 4.2 (b). We compute the distances $d_{k,i}^\tau$ and $d_{i,j}^\tau$ and determine whether $v_i^\tau$ will replace the lower or upper bound. In this example, since $d_{k,i}^\tau < d_{i,j}^\tau$, indicated by the intersection points between the black solid line and the two curves, we replace the lower bound with $v_i^\tau$, i.e., $l_i^{\tau+1} = v_i^\tau$, so that the properties of the lower and upper bounds still hold.

It is clear that the smaller the search range gets, the better the distance functions can be approximated by linear functions. As shown in Figure 4.2 (b), the curves between the bounds are nearly straight even if the distance functions themselves are not linear. This allows the desired isovalue to be identified within a small number of iterations. In our experiment, we find that five iterations per step is sufficient. For more details, please refer to Figure 4.5 and Chapter 4.2.1.

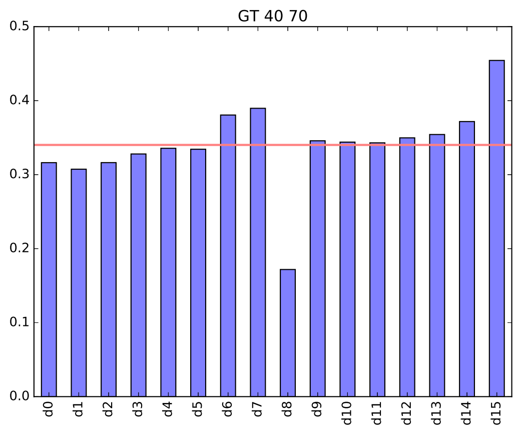
### 4.1.3 Convergence Stabilization

During the refinement stage, some proposed isovalues potentially lead to an adverse change of distance. This typically translates into distance values, named *spikes*, that are much higher/lower than the average distance, causing large average and maximum errors. Figure 4.3 demonstrates two common types of spikes: (1) the binary search successfully finds an isovalue that has mostly equal distances to its two neighbors, but these distances are larger than the average distance; and (2) the binary search fails to identify a desired isovalue, and the distance between this isovalue and one of its neighbors becomes a spike.

The first type of spikes between three isovalues $v_{i-1}$, $v_i$, and $v_{i+1}$ can be expressed as $d_{i-1,i} \approx d_{i,i+1}$ and $d_{i-1,i}, d_{i,i+1} >> d_\mu$. These spikes usually appear due to the underestimation of the differences of isosurfaces in the interval $[v_{i-1}, v_{i+1}]$. For example, in Figure 4.3 (a), after one step of the refinement stage, the binary search identifies an isovalue $v_{12}$ whose



(a)



(b)

Figure 4.3. An example of neighboring distances from different time steps of the GT variable of the Ionization data set. Every bar shows the distance $d(i, i+1)$. (a) shows one reason for a spike where two neighbors are too far apart. (b) shows a different cause, that is related to a jump discontinuity in the distance function.

distances to its neighbors (i.e., $d_{11,12}$ and $d_{12,13}$) are about twice as high as the average distance, meaning that the interval between $v_{11}$, $v_{12}$, and $v_{13}$ may be too large. This type of spikes may gradually disappear since $v_{i-1}$ and $v_{i+1}$ will be moved closer to $v_i$ in the next step of the refinement stage. For example, since $d_{10,11}$ is much smaller than $d_{11,12}$, $v_{11}$ will be moved closer to $v_{12}$ to reduce $d_{11,12}$ for an equal distance between $v_{10}$, $v_{11}$, and $v_{12}$. However, this type of spikes still causes a steep increase of the average and maximum errors, leading to an unstable status during the refinement stage.

To alleviate this problem, we propose a *spike treatment* that rejects isovalues leading to spikes. Formally, for every isovalue $v_i^\tau$ that has been changed in step $\tau$, we compare $d_{i-1,i}^\tau$ and $d_{i,i+1}^\tau$ to $d_\mu^\tau$ using their relative differences with respect to $d_\mu^\tau$.

If any of the two difference values surpasses a predefined spike threshold $\delta_s$, we reject $v_i^\tau$ and replace it with $v_i^{\tau-1}$. Note that the old value $v_i^{\tau-1}$ has more agreeable distances, as $v_{i-1}$ and $v_{i+1}$ are static in this step. Intuitively, by avoiding the steep changes, this treatment postpones, instead of preventing, the salient isovalue $v_i$ to be discovered. Therefore, the

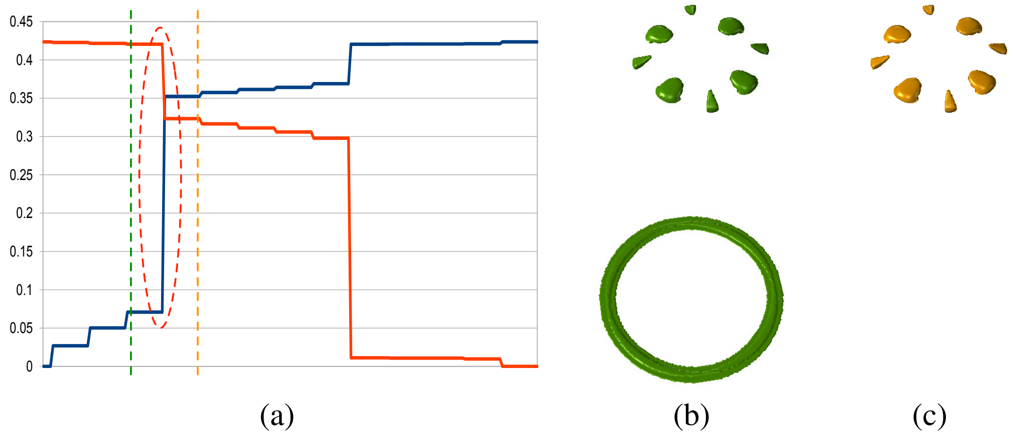


Figure 4.4. An example of discontinuous distance functions using the GT variable of the Ionization data set. (a) shows the distances to isovalue 20547.8 (blue) and to isovalue 20680.4 (red). The horizontal axis represents sampled isovalues and the vertical axis represents the distance. (b) and (c) are the two isosurfaces corresponding to the isovalues highlighted by the dashed lines in (a).

entire refinement stage will exhibit a more smoothly convergence toward the best solution.

For the second type of spikes, the binary search fails to identify an isovalue $v_i$ with equal distances to its neighbors. In this case, at least one of the distances $d_{i-1,i}$ and $d_{i,i+1}$ will differ from the average distance $d_\mu$. For example, in Figure 4.3 (b), the distance $d_{8,9}$ is much smaller than the other distances between neighbors, therefore, leading to a larger error. This type of spikes is usually caused by a *jump discontinuity* in the distance functions between the neighboring isovalues.

A jump discontinuity (henceforth jump) appears when the distance function between two isovalues is discontinuous. An example is demonstrated in Figure 4.4. For the purpose of analysis, we consider two distance functions $d_l(v_i)$ and $d_r(v_i)$, which map an isovalue $v_i$ to its distances to its left and right neighbors, respectively. In this example, we densely sample 200 isovalues between two fixed isovalues 20547.8 and 20680.4, and compute the distance from each sampled isovalue to the fixed ones. In Figure 4.4 (a), the distances to isovalues 20547.8 and 20680.4 are plotted as blue and red lines, respectively. Unlike the case of two smooth distance functions, as demonstrated in Figure 4.2 (b), a steep change occurs, highlighted by the red dashed ellipse. This indicates that the two distance functions are not continuous at the corresponding isovalue.

Figure 4.4 (b) and (c) show the isosurfaces corresponding to the two isovalues on the two sides of this critical isovalue. We can see that this isovalue actually corresponds to the topological change with the lower ring emerging. The blue line appears below (above) the red line before (after) this change. This means, given the properties of the lower and upper bounds, the lower (upper) bound will always be on the left (right) of this isovalue. After several steps, the binary search will be trapped in a small value range centered at this critical isovalue. This does not only lead to a large error by itself but also stops the distance values from propagating from one side of the critical isovalue to the other side. Therefore, when a jump appears, we may only achieve two equal distances on the two sides of the critical isovalue.

To tackle this problem, we propose a *jump treatment* that first identifies the isovalue $v^*$ of the jump and balances the distances on its two sides. By definition, a jump is a discontinuous point in the distance functions. Therefore, $v^*$ can be detected through examining the following criterion

$$\begin{aligned} d(v_i, v^* + \varepsilon) &>> d(v_i, v^*), \text{ and} \\ d(v^*, v_j) &>> d(v^* + \varepsilon, v_j). \end{aligned} \tag{4.1}$$

Instead of explicitly detecting $v^*$, we examine this criterion at each iteration of the refinement stage. Once a jump is encountered, we fix the upper and lower bounds of the binary search so that the jump will reside in the bounded interval, and push the isovalues from one side of the jump to the other side. Let $V_l = \{v_k | \ 0 \leq k \leq i\}$ and $V_r = \{v_k | \ j \leq k < n\}$ be the isovalue sets on the left and right sides of the jump, respectively, and let $d_\mu(V_l)$ and $d_\mu(V_r)$

be the average distances of the neighboring isovalues in $V_l$ and $V_r$, respectively. Without loss of generality, assuming $d_\mu(V_l) < d_\mu(V_r)$, we take an isovalue from $V_l$ and push it to $V_r$, so that the set of isovalues $V_l$ becomes sparser and the set of isovalues $V_r$ gets denser. This will lead to an increase of $d_\mu(V_l)$ and a decrease of $d_\mu(V_r)$, thus achieving a better balance of the average distances on both sides.

### 4.1.4 Distance Measures

We experiment our approach with two different distance measures: the *mean of the closest point distances* (MCP) [156] and the ISM measure [24]. Other distance measures may be applied as well, according to the specific analysis goals.

**MCP distance.** The MCP distance between two isosurfaces $s_i$ and $s_j$ uses the Euclidean distance to compute the closest distance for every point $p_k$ on $s_i$ to any point $p_l$ on $s_j$ and vice versa. The MCP distance of $s_i$ and $s_j$ is defined as follows

$$\begin{aligned} d_{\mathrm{MCP}}(s_i, s_j) &= \frac{1}{2}\left(d(s_i, s_j) + d(s_j, s_i)\right), \text{ where} \\ d(s_i, s_j) &= \frac{\sum_{p_k \in s_i} \min_{p_l \in s_j} \|p_k - p_l\|}{|S_i|}. \end{aligned} \tag{4.2}$$

**ISM distance.** The ISM measure inspects the mutual information of the distance fields corresponding to two isosurfaces $s_i$ and $s_j$. Based on the uniformly sampled distance fields of the two isosurfaces, a joint histogram can be computed to derive the mutual information. Again, we use the Euclidean distance to compute the distance fields for an isosurface $s$. For each grid point in the distance field, we record two closest distances from that point to the two isosurfaces $s_i$ and $s_j$, and compute the joint histogram of the distances. The mutual information between two random variables $X$ and $Y$ can be computed from their joint histograms as follows

$$
\begin{aligned}
I(X,Y) &= H(X) + H(Y) - H(X,Y), \text{ with} \\
H(X) &= -\sum_{x\in X} p_X(x)\log\left(p_X(x)\right), \\
H(X,Y) &= -\sum_{x\in X}\sum_{y\in Y} p_{X,Y}(x,y)\log\left(p_{X,Y}(x,y)\right),
\end{aligned} \tag{4.3}
$$

where $H(X)$ and $H(Y)$ are the marginal entropies and $H(X,Y)$ denotes the joint entropy of $X$ and $Y$. In our case, $X$ ($Y$) is the distance from a grid point in the distance field to isosurface $s_i$ ($s_j$). We further normalize the mutual information

$$
\hat{I}(X,Y) = \frac{2I(X,Y)}{H(X)+H(Y)}, \tag{4.4}
$$

and convert the similarity measure into a distance measure

$$
d_{\mathrm{ISM}}(s_i,s_j) = 1 - \hat{I}(s_i,s_j). \tag{4.5}
$$

**Approximation and acceleration.** Distance measures between isosurfaces often share two common steps: constructing isosurfaces and identifying the closest points of given points. When computing the ISM distance, the distance field of an isosurface requires the distance from each grid point to the closest point on the isosurface to be computed. When computing the MCP distance between two isosurfaces, for each point on one isosurface, the closest point on the other isosurface needs to be identified. We take three considerations from Chapter 3 to accelerate these two key steps.

First, we approximate each isosurface using a point set instead of extracting the actual surface. Generating the exact isosurface produces multiple points and their connections for each voxel. However, the connections are usually not involved in the distance computation and the points are often unnecessarily dense. This approximation scheme splits the volume into uniform blocks and examines each block to determine whether it contains the isosurface. The centers of blocks that contain the isosurface are considered as an approximation of the isosurface. Using this scheme, the error for computing the closest point is bounded by $(\sqrt{3}/2)l$ (i.e., half the length of a block's diagonal $l$).

Second, we build one BVH-tree for each isosurface to organize its approximation points. This allows the closest point on an isosurface to be queried efficiently. For construction efficiency, we use bounding boxes and leverage Karras' algorithm [113] to build each BVH-tree on GPU in parallel. The BVH-trees are stored in the graphics memory, so that multiple closest point queries can be performed in parallel. In addition, given a point, we estimate the upper bound of the distance to the closest point by uniformly sampling the approximation points. Since the approximation points are loosely ordered following the scanline order, this provides a tighter upper bound and therefore avoids many unnecessary branches of the BVH-tree to be traversed.

Third, both the distance fields and the approximation of isosurfaces can be downsampled to further reduce the time cost. The approximation can be downsampled by scanning blocks of voxels. The centers of blocks that contain the isosurface become the approximation in this case. The error of the closest point is bounded by $(\sqrt{3}/2)l$, where $l$ is the edge length of a block in voxels. For the distance fields, it has been shown that the resolution can be reduced by eight folds along each dimension without sacrificing the quality of the resulting ISMs [24].

Through this acceleration measures, we can achieve a linear time complexity considering all steps to compute neighboring distances, except for building the BVH-trees. The initial approximation examines $O(|V|)$ voxel, where $|V|$ is the size of the volume. For the next step, building the BVH-tree, Karras reported the time complexity of $O(n \log n)$ in the worst case [113]. Note that number of points from the approximation, $n$, is typically much smaller than $|V|$. Using the BVH-tree, querying the closest point of a given point takes on average $O(\log n)$ steps. The number of queries for this is bounded by either the size of the distance field ($O(|V|)$) (when using ISM distance) or the size of another surface

($O(n)$) (when using MCP distance). Using the GPU, multiple queries can be performed in parallel. For the ISM distance, we further compute mutual information, by examining every point in the distance field. In our experiment, we find that computing the distance between two isosurfaces already fully utilizes the computation power of a single GPU. Therefore, the cost of our approach is linear to the number of isosurfaces if a single GPU is used. Multiple GPUs, if available, can be readily utilized as computing multiple distances is embarrassingly parallel.

## 4.2 Results

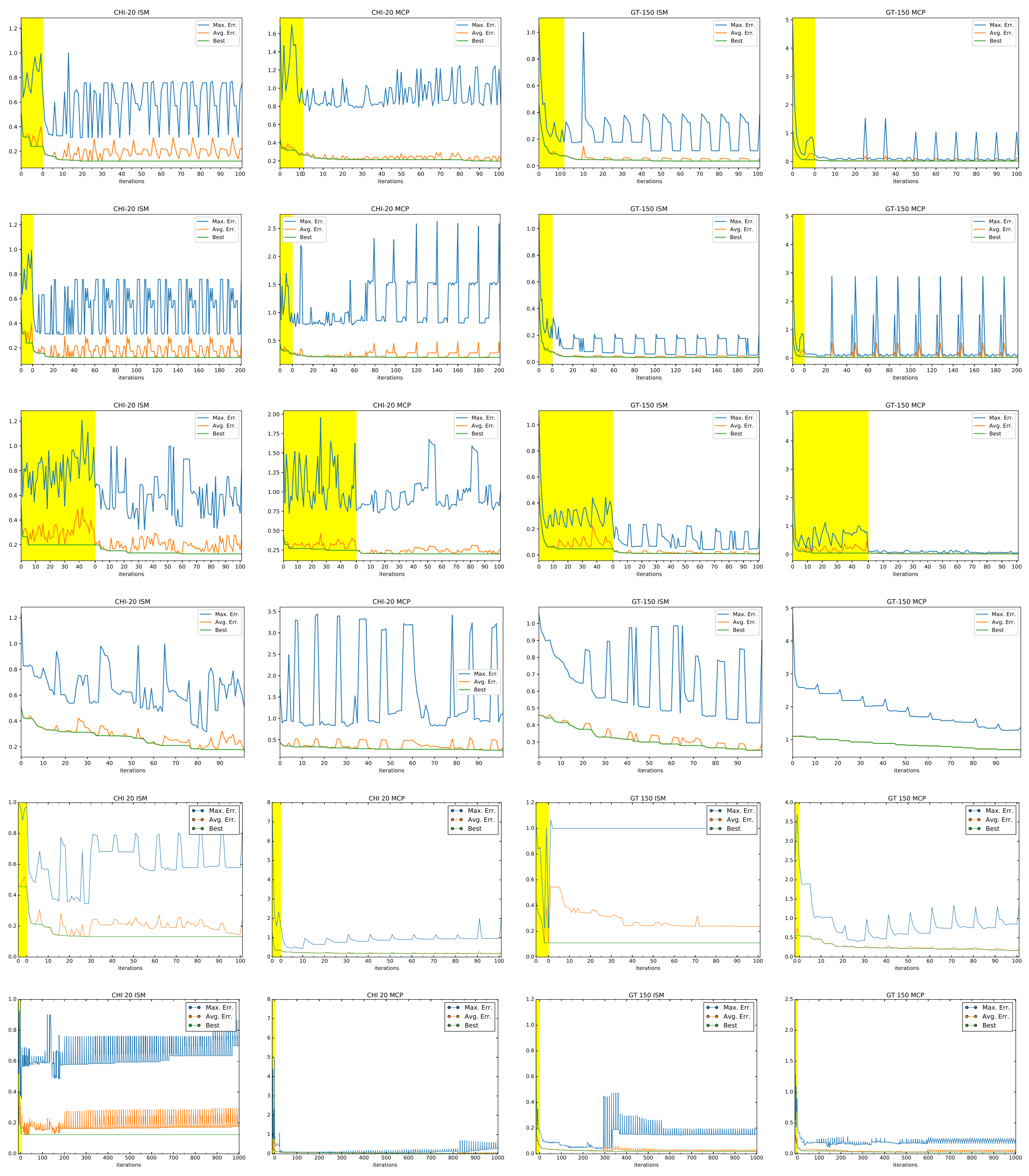
We mainly run our experiments on a desktop with an Intel Core i7-4790 quad-core CPU @ 3.6 GHz, 32 GB RAM, and an NVIDIA GeForce GTX 760 GPU accelerator. For further exploration of time-varying data sets, we leveraged a cluster with a shared GPU queue. The shared GPU queue uses the following systems:

- 8 Quantum TXR231-1000R servers with dual Intel Xeon 12-core CPU E5-2650 v4 @ 2.20GHz, 128 GB RAM, and 4 NVIDIA TITAN X (Pascal) GPU accelerators;
- 8 Quantum TXR231-1000R servers with dual Intel Xeon 12-core CPU E5-2650 v4 @ 2.20GHz, 128 GB RAM, and 4 NVIDIA Tesla P100-PCIE-16GB GPU accelerators.

The queue distributes the workload on different machines depending on the availability. Since we were only interested in the number of iteration needed to achieve a good solution, we did not restrict our runs to a single hardware configuration. In the following, we first analyze our general approach quantitatively (Chapter 4.2.1) and qualitatively (Chapter 4.2.2), and then study the impact of the spike treatment and jump treatment (Chapter 4.2.3).

### 4.2.1 Quantitative Study

**Quality measures.** We evaluate the quality of a set of selected isovalues $V = \{v_1, \dots, v_n\}$ based on the distances among neighbors (i.e., $d_{1,2}, \dots, d_{n-1,n}$) and the average distance $d_\mu$.



(a) (b) (c) (d)

Figure 4.5. Parameter choices on error curves. (a) and (b) CHI variable of the Combustion data set at time step 20. (c) and (d) show GT variable of the Ionization data set at time step 150. (a) and (c) use the ISM distance, and (b) and (d) use the MCP distance. Top to bottom: $<\delta_e, \delta_\tau, \delta_r>=<10, 5, 20>$, $<10, 10, 20>$, $<50, 5, 20>$, $<0, 5, 20>$, $<1, 5, 20>$, and $<10, 5, 200>$, respectively. In each plot, the yellow and white background colors indicate the estimation and refinement stages, respectively. The blue, orange, and green curves show the maximum error, average error, and best error over iterations.

For each distance $d_{i-1,i}$, we compute an error term $e_{i-1,i}$ to indicate the difference between this distance and the average distance

$$e_{i-1,i} = \frac{\|d_{i-1,i} - d_\mu\|}{d_\mu}, \tag{4.6}$$

where dividing the absolute difference by the average distance normalizes the error term. In this work, we quantify the quality of selected isovalues using the average error

$$e_\mu = \frac{\sum_{i=2}^{n} e_{i-1,i}}{n-1}, \tag{4.7}$$

and the maximum error

$$e_{\max} = \max_{2 \le i \le n} \{e_{i-1,i}\}. \tag{4.8}$$

Since the maximum error is usually determined by the nature of the data sets and the distance measures, as will be shown in Chapter 4.2.3, we focus on the average error and use it to determine the best solution, i.e., the set of isovalues with the minimum average error. We do not use the variation or standard deviation to evaluate whether the distances are similar since both of them are dominated by the maximum error when the other errors are small.

**Parameter choices.** Our approach has three parameters: $\delta_e$ the minimum number of iterations in the estimation stage, $\delta_\tau$ the number of iterations at each step in the refinement stage, and $\delta_r$ the number of steps in the refinement stage. For simplicity, we use a 3-tuple $< \delta_e, \delta_\tau, \delta_r >$ to denote a parameter setting. Figure 4.5 shows the results of using two variables of the Combustion and Ionization data sets for both the ISM and MCP distance measures with six different sets of parameter values ($< \delta_e, \delta_\tau, \delta_r > = < 10, 5, 20 >$, $< 10, 10, 20 >$, $< 50, 5, 20 >$, $< 0, 5, 20 >$, $< 1, 5, 20 >$, and $< 10, 5, 200 >$). For each run, we plot the maximum error, the average error, and the current best solution over iterations. The current best solution is the one with the minimum average error obtained up to the current iteration.

We first investigate the impact of parameter $\delta_\tau$. In the top two rows of Figure 4.5, we fix the two parameters $\delta_e = 10$ and $\delta_r = 20$ and compare the performance of $\delta_\tau = 5$ (first row) and $\delta_\tau = 10$ (second row). At each step, having more iterations may potentially allow better convergence of the binary searches. But overall, we do not see a noticeable improvement of accuracy using $\delta_\tau = 10$ over $\delta_\tau = 5$ since the shape of the green curves (best solution) in the same column are mostly the same. With the same number of steps ($\delta_r = 20$), this indicates that we obtain similar results using $\delta_\tau = 10$ but with twice the number of iterations as using $\delta_\tau = 5$.

Next, we study the impact of parameter $\delta_e$. In the first and third rows of Figure 4.5, we use $\delta_e = 10$ and $\delta_e = 50$, respectively. The other two parameters are fixed ($\delta_\tau = 5$ and $\delta_r = 20$). We find that more than ten iterations in the estimation stage are usually unnecessary since the best solution is mostly unchanged after ten iterations, as shown in the third row of Figure 4.5. In addition, we do not find that having more iterations in the estimation stage helps the refinement stage reach the best solution faster. The green curves in the white background, corresponding to the best solution in the refinement stage, demonstrate similar decreasing patterns.

However, we still find that the estimation stage is necessary for the refinement stage to quickly reach its best solution. In the fourth row of Figure 4.5, we experiment our approach with only the refinement stage, i.e., $\delta_e = 0$. With this setting, we find that the refinement stage approaches the optimal solution much slower. For example, using CHI of the Combustion data set and the ISM distance measure shown in (a), the best solution slowly improves over the 100 iterations without the estimation stage and reaches the minimum average error of 0.179 at the last iteration. With the estimation stage, the best solution until the 20-th iteration in the refinement stage has an average error of 0.121, which is already very close to the minimum average error of 0.120 for the entire 100 iterations. The computation time of each iteration in the estimation and refinement stages is similar, since both of them are dominated by the computation of distances between neighbors. Therefore, including the estimation stage clearly gives a better performance.

The fifth row shows the results with only one iteration in the estimation stage, i.e., $\delta_e = 1$. In contrast to setting $\delta_e = 10$, letting $\delta_e = 1$ leads to a more stable convergence, implying that the algorithm got stuck in a local optimum. Note that for the CHI variable at time step 20, we sometimes obtain empty isosurfaces. In this case, we run the estimation stage for more iterations until we obtain a set without empty surfaces.

In the last row, we show the results with $< \delta_e, \delta_\tau, \delta_r > = < 10, 5, 200 >$ (1000 total iterations in the refinement stage). For some data sets, we achieve a slightly better solution several hundreds iterations later than the best solution achieved within 100 iterations. However, the overall convergence pattern does not change. We believe that, given the time-quality trade-off, a relatively good solution can be found within 100 iterations.

In Figure 4.6, we show the visual differences among three sets of isosurfaces identified in 20, 100, and 1000 iterations, respectively, using the GT variable of the Ionization data set at time step 150. While the best solution after 1000 iterations is found at iteration 333 with an average normalized error of 1.86%, the earlier ones 2.89% (at iteration 92) and 3.92% (at iteration 20) show small errors as well. Although the relative difference in average error seems enormous, their absolute difference is still small. Visual comparison confirms that similar isosurfaces are identified. In the top row of Figure 4.6, we show all isosurfaces corresponding to solutions at iteration 20 (a), 92 (b), and 333 (c), respectively. We find that the overall difference is barely visible. Therefore, to inspect more closely, we depict the fourth isosurface of the selected sets in the second row. We can clearly see a difference between the first two images ((d) and (e)) and the last one (f). Furthermore,

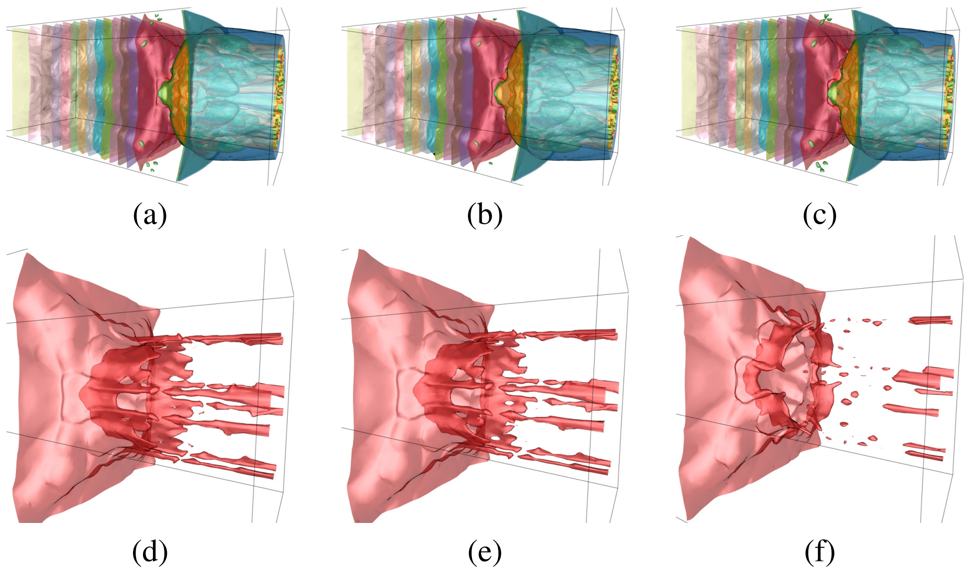


Figure 4.6. Comparison of the isosurfaces identified in different numbers of iterations using the GT variable of the Ionization data set at time step 150. The isosurfaces are selected from the best solutions after 20 ((a) and (d)), 100 ((b) and (e)), and 1000 ((c) and (f)) iterations in the refinement stage. The top row shows 17 selected isosurfaces rendered together. The bottom row shows a single surface highlighting the fine differences.

subtle differences between (d) and (e) can be seen. Although single surfaces differ among the different solution sets, the overall sets look fairly similar, offering a comprehensive overview of the volumetric data set.

In our experiment, we use a large enough value of $\delta_r = 20$ to study how the best solution evolves over iterations. We find that the setting of $< \delta_e, \delta_\tau, \delta_r > = < 10, 5, 20 >$ usually yields good results in terms of timing and error. Therefore, we use this setting for reporting the remaining results.

TABLE 4.1

PERFORMANCES USING THE ISM (A) AND MCP (B) DISTANCE.

| | dimension | avg. # iterations | | timing (sec.) | | average error | | | refine | difference to best (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| data set | x, y, z, v, t | estimate | refine | estimate | refine | initial | estimate | best | improve | 20 iter. | 40 iter. | 60 iter. | 80 iter. |
| Atmosphere | 147, 129, 49, 4, 121 | 11.1 | 57.9 | 3.36 | 27.61 | 0.437 | 0.153 | 0.081 | 47.29% | 26.62 | 13.47 | 5.71 | 1.71 |
| Climate | 360, 66, 27, 2, 120 | 11.8 | 47.5 | 2.87 | 22.21 | 1.060 | 0.251 | 0.165 | 34.12% | 10.12 | 6.37 | 1.32 | 0.07 |
| Combustion | 480, 720, 120, 5, 122 | 11.9 | 68.8 | 27.90 | 211.35 | 0.417 | 0.184 | 0.099 | 46.20% | 25.39 | 13.16 | 5.31 | 1.16 |
| Hurricane | 500, 500, 100, 11, 48 | 11.2 | 59.2 | 28.07 | 230.11 | 0.427 | 0.150 | 0.073 | 51.07% | 40.50 | 18.08 | 7.65 | 3.92 |
| Ionization | 600, 248, 248, 8, 199 | 11.5 | 63.9 | 20.70 | 164.63 | 0.597 | 0.287 | 0.188 | 34.33% | 30.12 | 20.49 | 10.85 | 2.79 |
| Vortex | 128, 128, 128, 1, 90 | 13.3 | 95.0 | 8.49 | 60.73 | 0.158 | 0.041 | 0.011 | 73.81% | 41.42 | 28.95 | 20.22 | 15.27 |

(a)

| | dimension | avg. # iterations | | timing (sec.) | | average error | | | refine | difference to best (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| data set | x, y, z, v, t | estimate | refine | estimate | refine | initial | estimate | best | improve | 20 iter. | 40 iter. | 60 iter. | 80 iter. |
| Atmosphere | 147, 129, 49, 4, 121 | 11.1 | 59.6 | 4.10 | 31.70 | 0.525 | 0.198 | 0.099 | 49.94% | 57.60 | 19.91 | 8.92 | 2.74 |
| Climate | 360, 66, 27, 2, 120 | 12.3 | 93.8 | 6.34 | 48.29 | 0.863 | 0.260 | 0.033 | 87.15% | 206.16 | 104.33 | 53.49 | 27.29 |
| Combustion | 480, 720, 120, 5, 122 | 12.9 | 72.2 | 91.51 | 646.55 | 0.560 | 0.205 | 0.099 | 51.54% | 46.03 | 27.18 | 16.89 | 9.50 |
| Hurricane | 500, 500, 100, 11, 48 | 12.8 | 88.8 | 49.57 | 379.31 | 0.281 | 0.061 | 0.010 | 82.17% | 98.95 | 49.67 | 28.76 | 11.92 |
| Ionization | 600, 248, 248, 8, 199 | 11.8 | 62.3 | 38.65 | 290.46 | 0.524 | 0.209 | 0.135 | 34.98% | 27.79 | 15.26 | 6.08 | 2.72 |
| Vortex | 128, 128, 128, 1, 90 | 11.0 | 80.0 | 15.42 | 126.15 | 0.533 | 0.194 | 0.072 | 62.81% | 61.68 | 46.20 | 28.25 | 14.36 |

(b)

The three columns of average errors show the initial average error of uniform sampling, the average error after the estimation stage, and the average error of the best solution. The column "refine improve" shows the percentage of average error reduced by the refinement stage. The four columns of "difference to best" show the percentage of difference between the average error of the best solution and the average errors after 20, 40, 60, and 80 iterations in the refinement stage.

**Timing and quality.** As shown in Table 4.1, we study the timing and quality performance using six data sets with different characteristics. For each data set, we use all the associated variables with three time steps selected (the beginning, middle, and ending time steps). Collected for each variable and each selected time step, the results are averaged for each data set.

Table 4.1 (a) shows the results using the ISM distance. Our approach produces mostly equally spaced isosurfaces with the average errors smaller than 0.1 for most of the data sets except the Climate (0.165) and Ionization (0.188) data sets. This may be related to the structures of the data sets as their initial errors are the largest among these six data sets. The number of iterations in the estimation stage is stable for all data sets and slightly above our minimum number ($\delta_e = 10$). In the refinement stage, our approach reaches the best solution around 60 iterations for most of the data sets, except the Vortex data set (averaging 95 iterations). The Climate data set even reaches solutions that are close to the best solution within 20 iterations, with only 10.12% difference. The other data sets except the Vortex data set have differences less than or around 20% within 40 iterations and less than or around 10% within 60 iterations. Although having the slowest convergence, the Vortex data set achieves the smallest average error (0.011) among all data sets after the refinement stage. In Figure 4.9 (k) and (l), we find that the green curve declines slowly after 15 iterations in the refinement stage. The higher percentages shown in the table are probably due to the small average error. The average errors are smaller than 0.3 for all the data sets after the estimation stage, and the refinement stage further reduces the average errors by at least 30%, which confirms the necessity of the refinement stage. Our approach performs efficiently using the ISM distance. To process one volume, it takes around one minute for the Atmosphere, Climate, and Vortex data sets, and less than five minutes for the other data sets.

Table 4.1 (b) shows the results using the MCP distance. In general, we find that it takes more iterations for the refinement stage to approach the best solutions using the MCP distance. Three data sets reach the best solutions after 80 iterations. Within 40 iterations, only two data sets obtain good solutions whose average errors have less than 10% differences from the best solutions. For five of the data sets, it takes 80 iterations to reach reasonably good solutions with less than 15% difference from the best solutions. However, we find that the average errors are usually smaller using the MCP distance. The largest average error is 0.135 using the Ionization data set, and all the other data sets have average errors smaller than 0.1. The refinement stage provides more significant improvement using the MCP distance. It reduces the average error by at least 80% for two data sets and 49% for five data sets. Actually, we find that the data sets with a smaller best error usually benefit more from the refinement stage and take more iterations to reach the best solution. This is likely related to the intrinsic structures of the data sets. The distance functions are probably more continuous using these data sets so that the binary search in the refinement stage is less likely to be trapped by the discontinuous points in the distance function. Although the average error is smaller, we find that the MCP distance takes more time to compute. The Combustion data set requires the longest total computation time of around nine minutes. The computation time for the other data sets varies from one to seven minutes.

### 4.2.2 Qualitative Study

**Comparison to other approaches**. For visual comparison, we generate eight isosurfaces of the GT variable of the Ionization data set at time step 10 using our approach, the ISM approach [24], and k-means clustering [146]. For our approach, we fix the minimum and maximum isovalues and compute eight isovalues between them. For the other approaches, we evenly sample 128 isovalues and identify eight representative ones. We use the approximation from Chapter 3 to compute the ISM distances between the sampled isovalues for all three variants. The implementation of the ISM algorithm is based on the priority queue algorithm [24] and the k-means clustering is based on Lloyd's algorithm [142]. Figure 4.7 (a) shows the distance matrix of the isosurfaces selected by our approach. Note

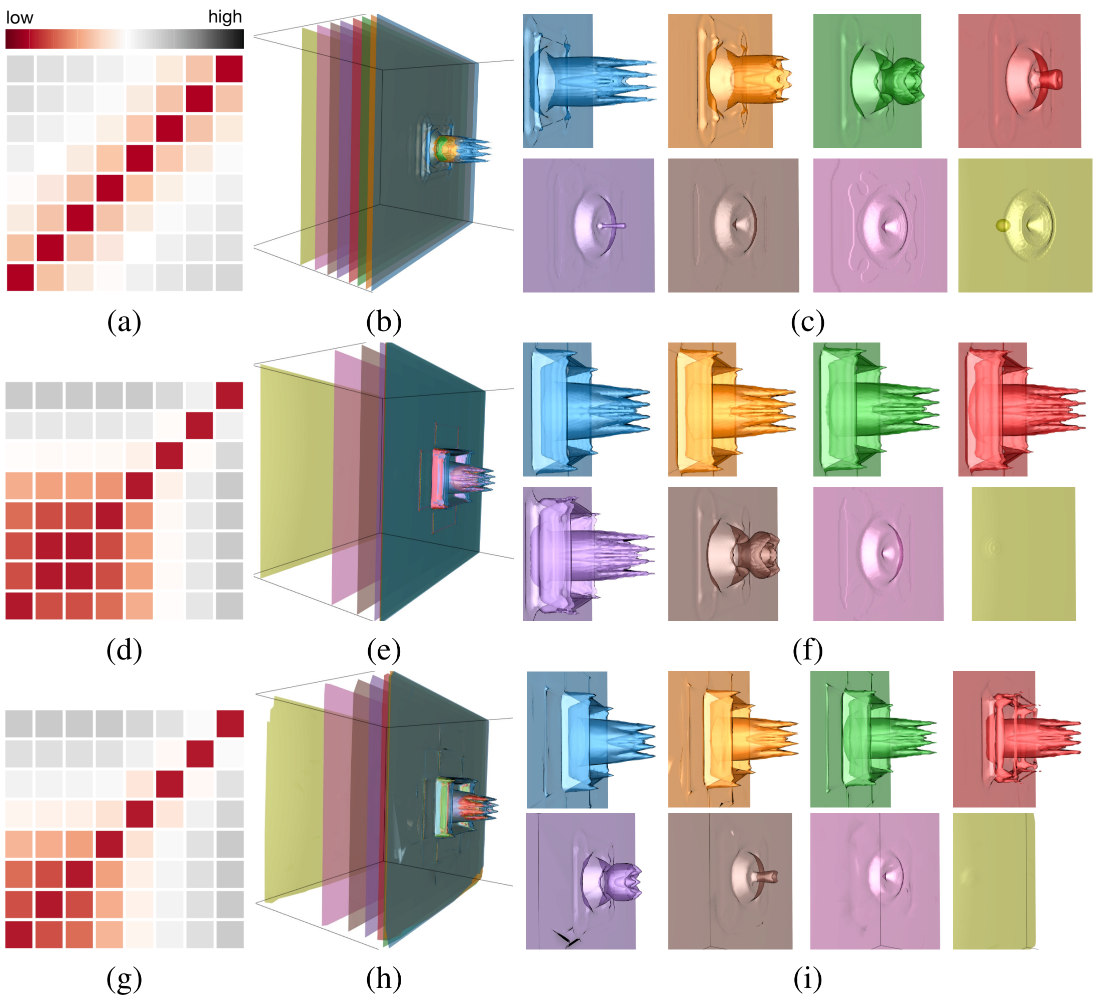


Figure 4.7. Comparison of our approach (top row), the ISM approach (middle row), and the k-means approach (bottom row) using the GT variable of the Ionization data set at time step 10. (a), (d), and (g) show the distance matrices of the selected isosurfaces. Note that the distances are normalized and all matrices use the same color map. (b), (e), and (h) show all the selected isosurfaces in a single image. (c), (f), and (i) show the central regions of individual isosurfaces in separate images.

that in this, and the following images, the distance values are in $[0, 1]$. We can see that the cells recording the differences between neighboring isosurfaces (i.e., the cells that are next to the diagonal cells) share similar colors. This indicates similar distances between neighbors. Figure 4.7 (b) confirms this observation as the selected isosurfaces distribute evenly in the space. The eight isosurfaces demonstrate a smooth transition of the features at the center of each isosurface, as shown in Figure 4.7 (c).

In contrast, five of the representative isosurfaces selected by the ISM approach and four of the representative isosurfaces selected by the k-means approach are similar. In Figure 4.7 (d), we can see a $5 \times 5$ block at the bottom left corner of the distance matrix of the representative isosurfaces, indicating high similarities among the corresponding isosurfaces. Similarly, Figure 4.7 (h) contains a $4 \times 4$ block.

In Figure 4.7 (e), we can see that the five similar representative isosurfaces collapse in space. Therefore, five of the feature regions in the representative isosurfaces actually corresponds to the nearly identical structure, as shown in Figure 4.7 (f). Although the ISM approach has a scheme to prevent similar isosurfaces to be selected [24], this scheme may be ineffective when the input is biased. As a matter of fact, more than half of the sampled isosurfaces in this volume correspond to the same structure. In general, we find that the distance matrix of the representative isosurfaces identified by the ISM approach often exhibits this kind of blocking effect for the structures captured by more sampled isosurfaces. This echoes that it is critical to producing unbiased isosurfaces as input for the surface-based volumetric data analysis algorithms. In addition, Figure 4.7 (h) shows the same effect for the k-means clustering with four similar isosurfaces selected. In Figure 4.7 (i), a closer inspection reveals that three of the four surfaces are very similar, with the fourth one being closely related to them.

In contrast, Figure 4.8 depicts the representative isosurfaces selected by the ISM [24] and k-means clustering [146] approaches using 128 nearly equally spaced isosurfaces as input instead of the uniformly sampled ones. We use the GT variable of the Ionization

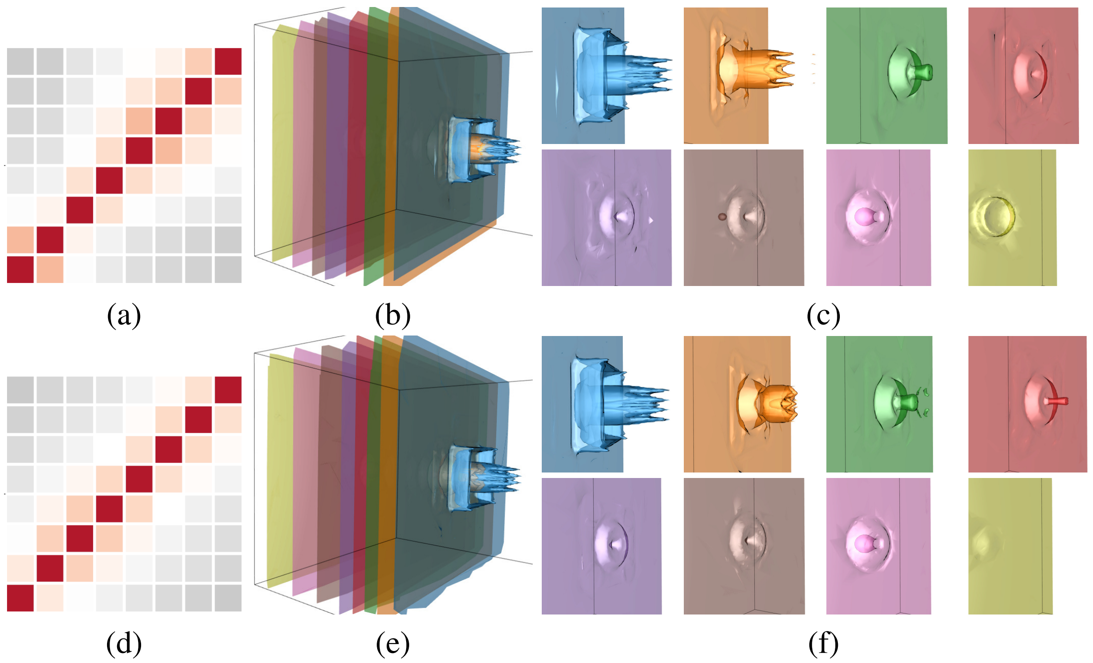
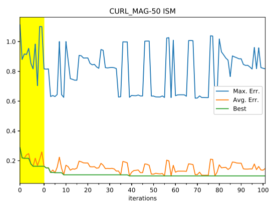


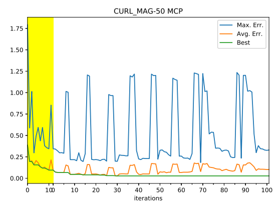
Figure 4.8. Comparison of results using a set of 128 nearly equally spaced isosurfaces as input for the ISM approach (top row) and the k-means approach (bottom row) using the GT variable of the Ionization data set at time step 10. (a) and (d) show the distance matrices, (b) and (e) show the representative isosurfaces rendered together, while (c) and (f) show the individual surfaces.

data set at time step 10. In (a) and (d), the distance matrices do not show the strong blocking effect, which means that the problem of oversampling certain value ranges could be circumvented. Compared to Figure 4.7, we can see that there is a shift between the representative surfaces selected, allowing to further explore previously overseen isosurfaces. For example, the third surface in the second row (pink) in both (c) and (f) has not been discovered previously. This further indicates that an unbiased input may improve the understanding of the underlying surfaces.

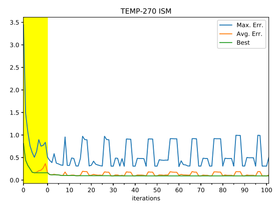
**Comparison of ISM and MCP distances.** For a qualitative study of the impact of distance measures, we first investigate the error curves using the ISM distance and the MCP distance, as shown in Figure 4.9. For each volume, we chose to identify 15 equally spaced



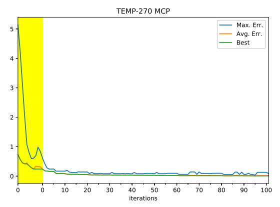
(a)



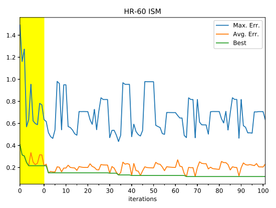
(b)



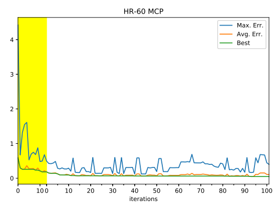
(c)



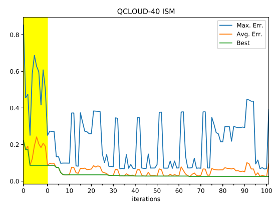
(d)



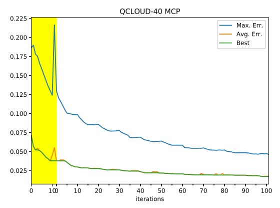
(e)



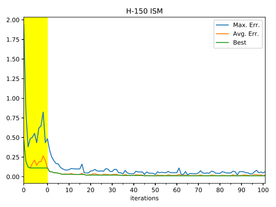
(f)



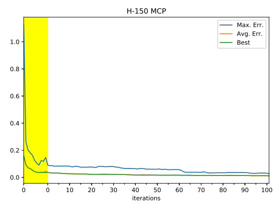
(g)



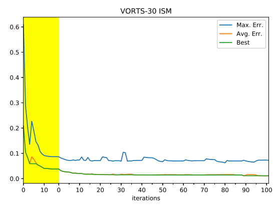
(h)



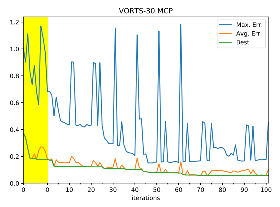
(i)



(j)



(k)



(l)

Figure 4.9. Typical error curves over iterations using the ISM distance (first and third column) and the MCP distance (second and fourth). (a) and (b) show the results of the curl magnitude (CURL_MAG) variable of the Atmosphere data set at time step 50. (c) and (d) show the results of the temperature (TEMP) variable of the Climate data set at time step 270. (e) and (f) show the results of the heat release (HR) variable of the Combustion data set at time step 60. (g) and (h) show the results of the cloud moisture mixing ratio (QCLOUD) variable of the Hurricane data set at time step 40. (i) and (j) show the results of the H mass abundance (H) variable of the Ionization data set at time step 150. (k) and (l) show the results of the Vortex data set at time step 30.

isovalues between the minimum and maximum isovalues. In general, the curves confirm our finding in Table 4.1 that the MCP distance has a slightly smaller average error. The only exception is the Vortex data set. Figure 4.9 (k) and (l) show that the MCP distance converges slower with unstable spikes of the maximum error curve for this data set. Figure 4.10 shows the distance matrix and a set of selected isosurfaces using the Vortex data set for each distance measure. In Figure 4.10 (a), we can see that the distances between

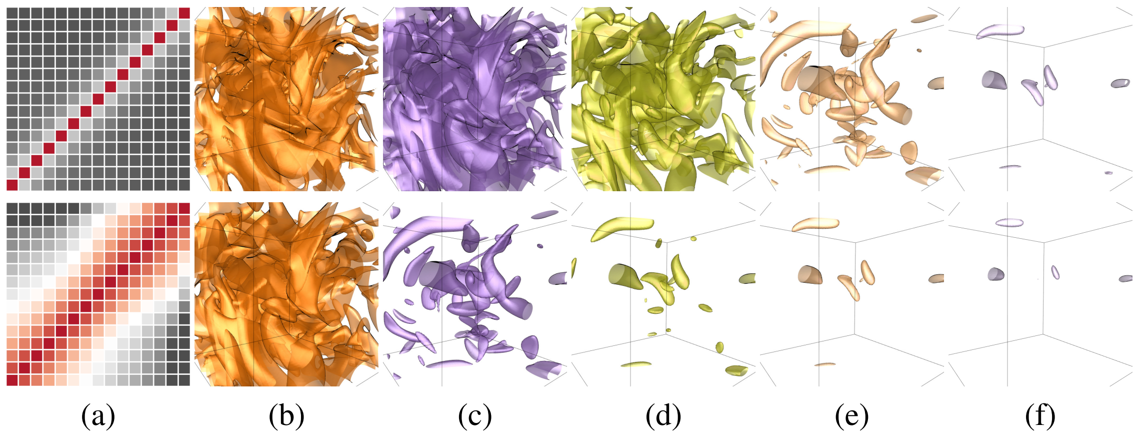


Figure 4.10. Comparison of the ISM distance (top row) and the MCP distance (bottom row) using the Vortex data set at time step 30. (a) shows the distance matrices of the selected isosurfaces. (b) to (f) show the two sets of isosurfaces chosen by the respective distance measures.

neighbors are actually similar for both measures. However, the two distance measures behave differently with this data set: the MCP distance changes in a smoother manner when the isovalues become more different, while the ISM distance seems to better distinguish isovalues in a smaller value range. For each distance measure, we evenly select five isosurfaces (the second, fifth, eighth, 11th, and 14th) from the set of fifteen isosurfaces, as shown in Figure 4.10 (b) to (f). We find that the ISM distance identifies more large-scale isosurfaces while the MCP distance selects more small-scale isosurfaces. This is probably because the ISM distance better distinguishes the large isosurfaces and the MCP distance better differentiates the small ones using this data set.

We then investigate the Combustion data set, for which the average errors are similar using both measures. We show the results of the HR variable in the top two rows of Figure 4.11. For this variable, although both the error curves in Figure 4.9 (e) and (f) and the distance matrices in Figure 4.11 (a) indicate smaller errors using the MCP distance, we find that the isosurfaces selected using the two distance measures are actually similar, as shown in the top two rows of Figure 4.11 (b) to (d). The isosurfaces are the ninth, 12th, and

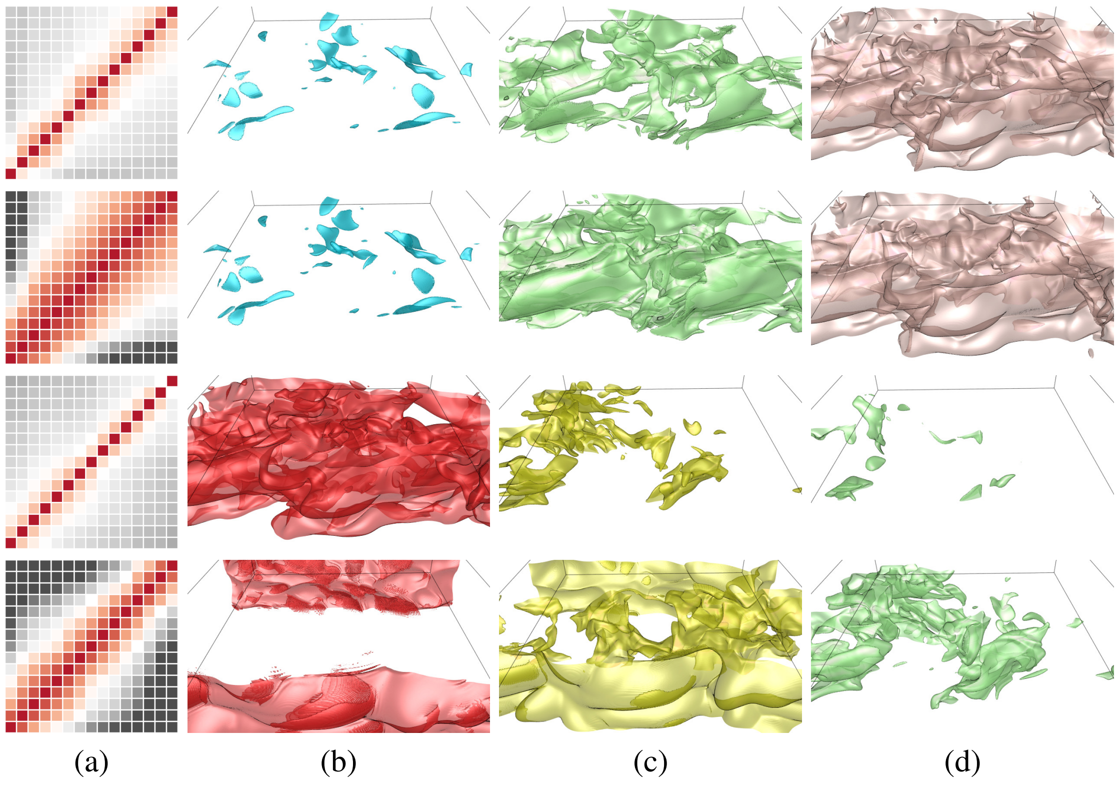


Figure 4.11. Comparison of the ISM distance and the MCP distance using the heat release (HR) and hydroxyl radical mass fraction (YOH) variables of the Combustion data set at time step 60. Rows from top to bottom show the results using HR with ISM distance, HR with MCP distance, YOH with ISM distance, and YOH with MCP distance, respectively. (a) shows the distance matrices of the selected isosurfaces. (b) to (d) show the four sets of isosurfaces chosen by the respective combinations of variable and distance measure.

14th from the fifteen selected ones. The first nine isosurfaces all demonstrate small-scale structures, which are visually similar. For the YOH variable, the distance matrices and the fourth, eighth, 12th isosurfaces (evenly sampled) are shown in the bottom two rows of Figure 4.11. We can see that the ISM distance selects more small-scale isosurfaces, which contradicts our findings with the Vortex data set. We actually find that both measures are sensitive to changes on small-scale isosurfaces in general. This leads to the conclusion: the behavior of the two distance measures heavily depends on the spatial distribution of the isosurfaces. While large isosurfaces usually have stable spatial distributions, changes (even if they are tiny) on small isosurfaces may lead to significantly different spatial distributions. Therefore, due to the nature of these two measures, the differences among small isosurfaces are often emphasized.

### 4.2.3 Discussion

**Time-varying data sets.** We further experiment possible solutions to improve the performance of time-varying data sets. In Chapter 4.2.1, we demonstrate that the refinement stage converges much faster with the use of the estimation stage, which indicates the importance of a good initial set of isovalues. Observing that the structures of volumes usually change gradually over the time steps, we hypothesize that using the isovalues selected for the same variable at the previous time step will speed up the computation. Although this strategy has not been fully studied, we discuss some preliminary findings. We use a cluster with a shared GPU queue to experiment with the six data sets shown in Table 4.2. For each variable, we have a normal run that starts from the uniform sampling of the value range for each time step, and a run that starts from the isovalues selected at the previous time step. Each computation node in the GPU queue performs one run of a variable. Since the cluster contains computation nodes of different configurations, we compare the performance using the number of iterations instead of the computation time.

Our experiment shows that we do not always get better results by starting from the isovalues at the previous time step. Keeping the isovalues produces better results than the normal run for more than 70% of the volumes with four data sets. However, for the Combustion and Ionization data sets, this strategy fails to produce better or even similar results. For the volumes that better results are obtained, it generally takes much fewer iterations to achieve the better results (less than 60% for three of the four data sets). For the Vortex data set, it only takes 21% of the number of iterations compared to the normal run. For five of the data sets, it takes less than 40% of the number of iterations to achieve a

TABLE 4.2

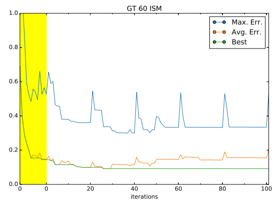
USING THE ISOVALUES SELECTED AT THE PREVIOUS TIME STEP AS INITIAL ISOVALUES FOR THE NEXT TIME STEP

| | | better vol. (%) | | # iter. (%) | |
|---|---|---|---|---|---|
| data set | # vol. | best | 10% diff. | best | 10% diff. |
| Atmosphere | 242 | 87.6 | 91.3 | 57.2 | 38.7 |
| Climate | 151 | 72.9 | 99.3 | 91.4 | 35.20 |
| Combustion | 61 | 0.0 | 1.6 | N/A | 18.93 |
| Hurricane | 19 | 79.0 | 84.2 | 42.6 | 33.7 |
| Ionization | 25 | 16.0 | 24.0 | 57.3 | 57.2 |
| Vortex | 45 | 77.8 | 82.2 | 21.0 | 18.4 |

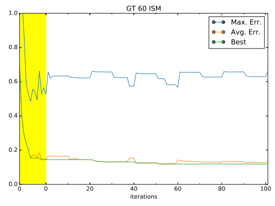
"# vol." shows the number of volumes experimented with. "better vol. (%)" shows the percentage of volumes achieving better results. "# iter. (%)" shows the percentage of iterations spent to achieve better results. "best" indicates that the result is better than the best solution obtained from the normal run, and "10% diff" indicates that the result is within 10% difference from the best solution of the normal run

similar result. However, the conditions under which this strategy will perform effectively are still not clear. We further investigate the impact of the overlap percentage of the value ranges at neighboring time steps and the average error at the previous time step, but none of them exhibits a significant impact on the performance. It seems that the performance of this strategy heavily relies on the nature of the data since for all the data sets shown in Table 4.2, the percentage of volumes with better results is either higher than 70% or less than 20%. If we can determine in advance that a time-varying data set is suitable for this strategy, nearly 2$\times$ speedup can be obtained.

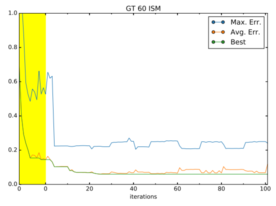
**Spike treatment.** To analyze the impact of the parameter $\delta_s$, we conduct an experiment using the common setting of $< \delta_e, \delta_\tau, \delta_r > = < 10, 5, 20 >$ but varying $\delta_s$ from 0.05 to



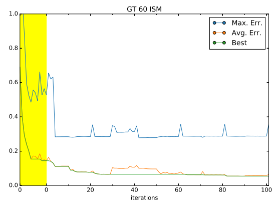
(a)



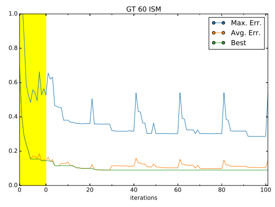
(b)



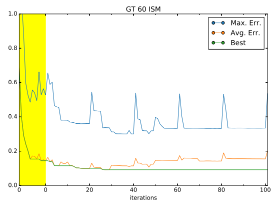
(c)



(d)



(e)



(f)

Figure 4.12. Comparison of the impact of $\delta_s$ on the convergence and the final result using the ISM distance on the GT variable of the Ionization data set at time step 60. (a) shows the original error curve over the set of iterations (i.e., $\delta_s = \infty$). (b) to (f) show the configurations with $\delta_s = \{0.05, 0.1, 0.15, 0.2, 0.25\}$, respectively.

0.25 in steps of 0.05. Note that the original approach without spike treatment can be considered as setting $\delta_s = \infty$, meaning that all spikes are tolerated and will not be explicitly treated. Figure 4.12 shows the results for the GT variable of the Ionization data set at time step 60. The figure represents typical error curves for the ISM distance measure with different $\delta_s$ settings. We can see that using a high tolerance value for $\delta_s$, as seen in (a) the original ($\delta_s = \infty$), (e) $\delta_s = 0.2$, and (f) $\delta_s = 0.25$, yields high spikes in the maximum error, resulting in spikes in the average error. Dampening those instabilities by reducing the threshold translates to fewer negative changes as can be seen in (b) $\delta_s = 0.05$, (c) $\delta_s = 0.1$, and (d) $\delta_s = 0.15$. This further allows us to achieve a lower average error for the two variations shown in (c) and (d). Using $\delta_s = 0.05$ may easily get trapped in a local optimum, since this parameter setting is too strict to allow any drastic changes that could resolve the problem. Setting $\delta_s$ to 0.1 or 0.15 leads to the most stable convergence, showing that those values offer a good balance between allowing too little changes ($\delta_s = 0.05$) and

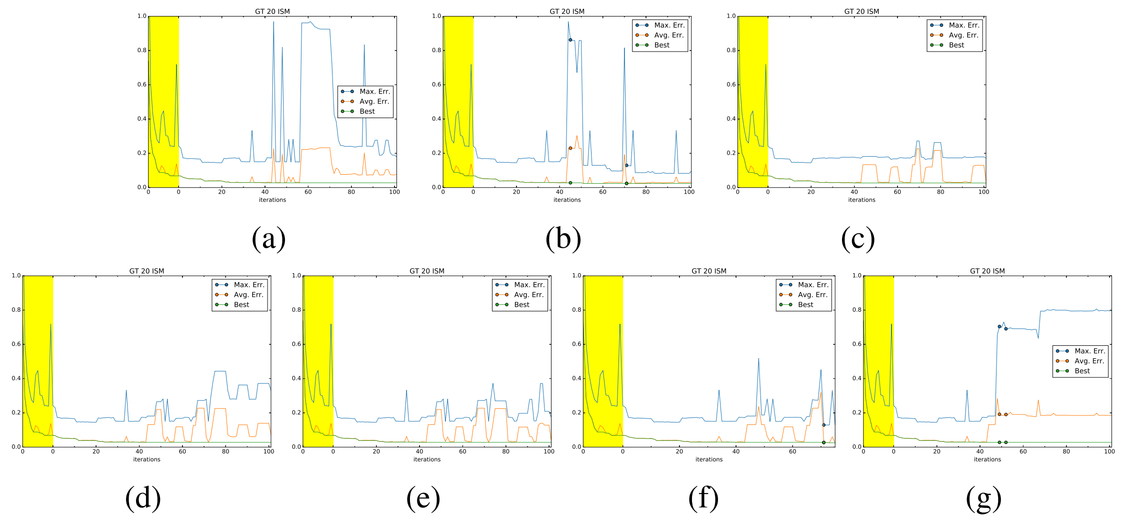


Figure 4.13. Error curves generated with different configurations using the GT variable of the Ionization data set at time step 20. (a) shows the original approach without jump and spike treatments. (b) shows a configuration with only jump treatment. (c) to (g) show configurations with both jump and spike treatments while $\delta_s$ increases from 0.05 to 0.25 in steps of 0.05. Dots on the curves indicate the iterations when jumps are detected.

allowing too much changes ($\delta_s \geq 0.2$). However, the best average errors achieved by setting $\delta_s = 0.1, 0.15, 0.2, 0.25$ are similar. We will study the impact of $\delta_s$ to the best average errors using more data sets later in this chapter.

**Jump treatment.** To analyze the impact of jumps and our treatment, we experiment our approach with several configurations (with and without spike treatment) using the GT variable of the Ionization data set at time step 20. The error curves are shown in Figure 4.13. In (a), we can see the original without jump or spike treatments. The rest shows the different parameter settings for $\delta_s$ to treat spikes while handling jumps at the same time. In our experiments, we encounter at most three jumps during the refinement stage using this data set. In (c) to (e), a strict choice of $\delta_s = 0.05, 0.1, 0.15$ restricts isovalue changes to a degree where it is not possible to detect a single jump in the distance functions. By allowing more drastic changes to the isovalues ((f) $\delta_s = 0.20$, (g) $\delta_s = 0.25$, and (b) $\delta_s = \infty$), we can see that jumps appear around strong fluctuation on the error curve. Treating these jumps helps to reduce the fluctuation from that point on. In (b), although the error curve still has strong spikes after treating the first jump, it finds a better solution than the original approach in a later iteration. An even stronger reduction in fluctuation of the error curve can be seen in (f). Dampened by spike treatment, the first jump is detected at iteration 70 and its treatment allows the curve to come down further, achieving the best solution for this data set at iteration 90. However, jump treatment does not always lead to better convergence or lower average and maximum error values. Figure 4.13 (g) showcases this. Instantly after the jump around iteration 50, the error function cannot recover, resulting in another jump quickly. This results in multiple fixed isovalues, allowing fewer changes in the set of isovalues at every iteration, which hinders the optimization process.

As mentioned previously, it is not always clear beforehand whether or not jump treatment improves the best solution due to multiple reasons. First, we treat each jump when it is detected at the refinement stage without the knowledge of all jumps. Therefore, treating one jump may lead to the appearance of an undiscovered jump. As shown in Figure 4.13 (g), the treatment may not be effective when multiple jumps are encountered. Second, pushing an isovalue over a jump does not guarantee that the distances on the two sides of the jump will be equal, especially when the jump occurs close to the end of the value range. In addition, this situation is often aggravated when additional jumps are encountered.

**Configuration selection.** As we have seen before, treating jumps and spikes can both have a positive or a negative impact on the achieved solution. In order to recommend an appropriate configuration, we run experiments among all time steps of different data sets and variables. Figure 4.14 shows the mean and standard errors of the best average errors with different configurations using four variables from four different data sets. The top row shows the results collected using all the time steps. We can see that in most of the cases, one of our convergence stabilization configurations improves the overall solution or yields a solution similar to the original one. For (a), the CHI variable of the Combustion

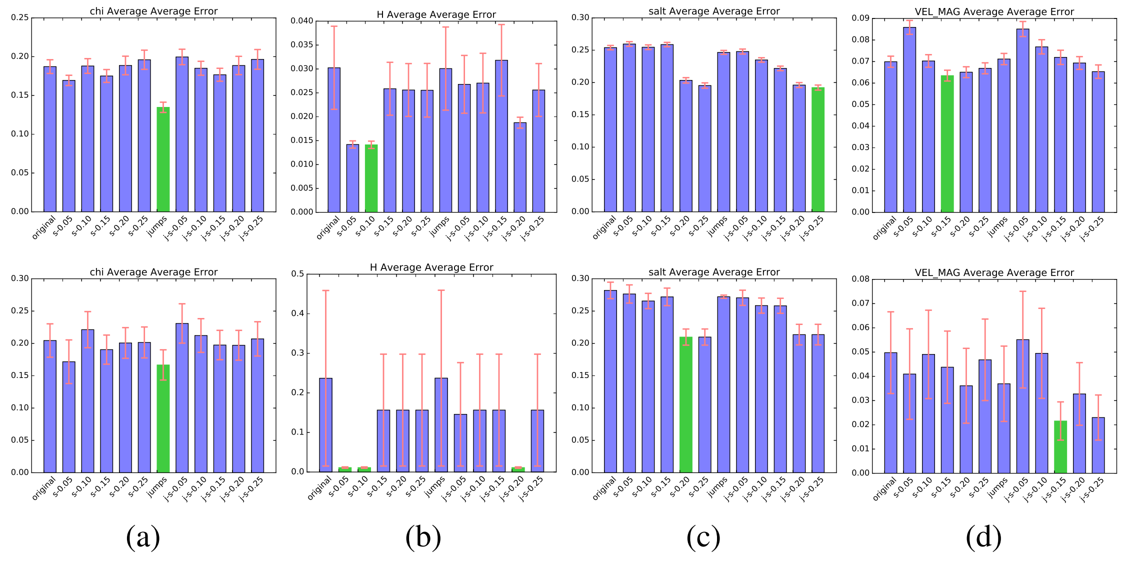


Figure 4.14. Comparison of the best average errors with different settings using (a) the CHI variable of the Combustion data set, (b) the H variable of the Ionization data set, (c) the salinity (SALT) variable of the Climate data set, and (d) the velocity magnitude (VEL_MAG) variable of the Atmosphere data set. The top row shows the results using all the time steps and the bottom row shows the results using only the first five time steps. In each chart, a bar represents the mean of the average errors achieved over the respective time steps of the corresponding data set (lower is better). The bars from left to right correspond to the original ($\delta_s = \infty$), spike treatment with $\delta_s = 0.05, 0.1, 0.15, 0.2, 0.25$, and jump and spike treatments with $\delta_s = \infty, 0.05, 0.1, 0.15, 0.2, 0.25$, respectively. The error bars in red indicate the standard error of the mean. The bars in green represent the best settings.

data set, our experiment shows that treating the jumps but ignoring spikes outperforms the other methods by a huge margin. This is likely due to an initial set of isovalues being stuck in a local optimum and can only escape it by treating jumps while allowing huge spikes. For (b), the H variable of the Ionization data set, we witness that a strict policy for spikes yields the best results. Interestingly, we can see that all configurations, except for $\delta_s = 0.05, 0.1$ without jump treatment, and $\delta_s = 0.2$ with jump treatment, have a very high standard error. This indicates that the structure of the volume differs heavily between time steps. In contrast, a loose spike treatment ($\delta_s = 0.2, 0.25$) has a small standard error among all configurations using the SALT variable of the Climate data set, as shown in (c). Similarly, in (d), the bar chart shows that too strict spike treatment can have a strong negative impact on the performance, using the VEL_MAG variable of the Atmosphere data set.

As there is no clear favorite among all data sets, we sample the first five time steps of a data set to see if we can predict a good configuration for the full run. These results are demonstrated in the bottom row of Figure 4.14. In (a) and (b), our method performs consistently over the time steps, showing the possibility to predict the best variation from the first couple of time steps. In (c), although we mispredict the best configuration, the predicted one still shows significant improvement over the original. However, in (d), the bar chart indicates that a bad configuration is recommended based on the results using the first five time steps. Although the recommended configuration ($\delta_s = 0.15$ with jump treatment) performs similarly as the original approach, we fail to spot the best configuration ($\delta_s = 0.15$ without jump treatment). In this example, the high standard errors among the first five time steps may indicate that the intermediate favorite is not a stable one for all the time steps. Overall, we cannot always predict the exact configuration that yields the best result for a given data set. However, when considering the first couple of time steps, we often identify a configuration of the convergence stabilization that outperforms the original.

## 4.3 Conclusions

We have presented a solution for identifying nearly equally spaced isosurfaces for volumetric data sets. Motivated by finding a small set of isosurfaces to better represent the underlying volume data in the spatial domain, we design a two-stage approach to seeking an approximated solution that maintains a good balance between quality and performance. The resulting surfaces are nearly equally spaced, and the user can freely choose the number of surfaces. Our study demonstrates the effectiveness of the proposed approach and leads to valuable feedback. To conclude, we summarize our key findings and major recommendations as follows.

First, our two-stage strategy is effective for achieving the best solution in a small number of iterations. Our experiment shows that both stages are necessary: without the estimation stage the refinement stage would require a lot more iterations to converge, and the estimation stage may never achieve a solution with a similar error as the refinement stage does.

Second, our approach can produce nearly equally spaced isosurfaces for most of the data sets, although some error may be introduced by jump discontinuities in the distance functions. These points may divide the entire range of isovalues into multiple segments and prevent the isovalues from moving between neighboring segments, resulting in unequal distances among the segments. Our convergence stabilization scheme alleviates this situation by treating spikes and jumps explicitly, although the effectiveness depends on the specific data set.

Finally, our approach is independent of the choice of the distance measure. This provides great flexibility for users to apply a suitable distance measure according to their own needs. Our experiment performs effectively using both the ISM and MCP distance measures with a common parameter setting of $< \delta_e, \delta_\tau, \delta_r > =< 10, 5, 20 >$ for all the data sets. To ensure smoother convergence without a strong restriction, we recommend using $\delta_s = 0.2$ and ignoring jumps, as these settings either outperform or show similar results as the original across all data sets.

# CHAPTER 5

# EXPLORING TIME-VARYING MULTIVARIATE VOLUME DATA USING MATRIX OF ISOSURFACE SIMILARITY MAPS

In this chapter, I present an interface to visualize and analyze large-scale time-varying multivariate volume data. The preprocessing for the MISM consists of computing ISM for each time step and variable, as well as across variables and time steps. Further, representative isosurfaces are fully extracted and simplified to save disk space. Lastly, all these artifacts are assembled into an interface which allows close inspection and guided exploration of a large-scale time-varying multivariate volume data set.[1]

## 5.1 Design Requirements

To investigate the physical phenomenon of the multifaceted time-varying multivariate data sets, experts need to understand the relationships among time steps, variables, and value ranges, which are often complicated and dynamic. The relationships of value ranges of multiple variables may exhibit various patterns even at a single time step, and these relationships may evolve over time and develop into different patterns. Understanding these relationships can hardly be achieved without an effective visual analytics system. We formulate the design requirements of such a system as follows:

**R1. Overview.** The system should provide two levels of overview. First, it should allow users to observe the overall relationships between variables and time steps at the

[1]This work was published as J. Tao, M. Imre, C. Wang, N. V. Chawla, H. Guo, G. Sever, and S. H. Kim. Exploring time-varying multivariate volume data using matrix of isosurface similarity maps. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1236–1245, 2019

*volume* level, helping them answer questions such as *which variables are more similar, which time steps are more similar, and which variables exhibit more frequent changes than the others?* Second, it should allow users to understand the relationships at the *isovalue* level. Similar questions should be answered at this level with more detail: *which ranges of isovalues lead to the relationships discovered at the volume level?*

**R2. Identification of representatives.** The system should quantify the similarities among time steps, variables, and isovalues and derive the representative ones. This provides users more quantitative evidence to verify the relationships they observe from the overview and to reduce the search space for detailed exploration.

**R3. Relationship-centric exploration.** The system should provide multiple modes to investigate various types of relationships, including *the relationships among value ranges for a given variable at a specific time step*, *the relationships among variables at a specific time step*, *the relationships of the same variable at different time step*, *the temporal evolution of the relationships of two variables*, and *the relationships of a specific feature and others (which may not reside in the same variable)*, etc. The system should be able to quantify each type of relationship and the interface should be optimized to allow exploring each type of relationships efficiently.

**R4. Multi-step comparison path.** The dynamic relationships among variables may require comparisons of hundreds of surfaces to be demonstrated. The system should decompose these complicated relationships into a *path* of multiple comparison steps. Each step should reveal a facet of the relationships with a reasonable number of surfaces that can be effectively rendered for clear visual comparison. In addition, the system should also maintain the consistency between surfaces in consecutive steps to preserve the user's mental map.

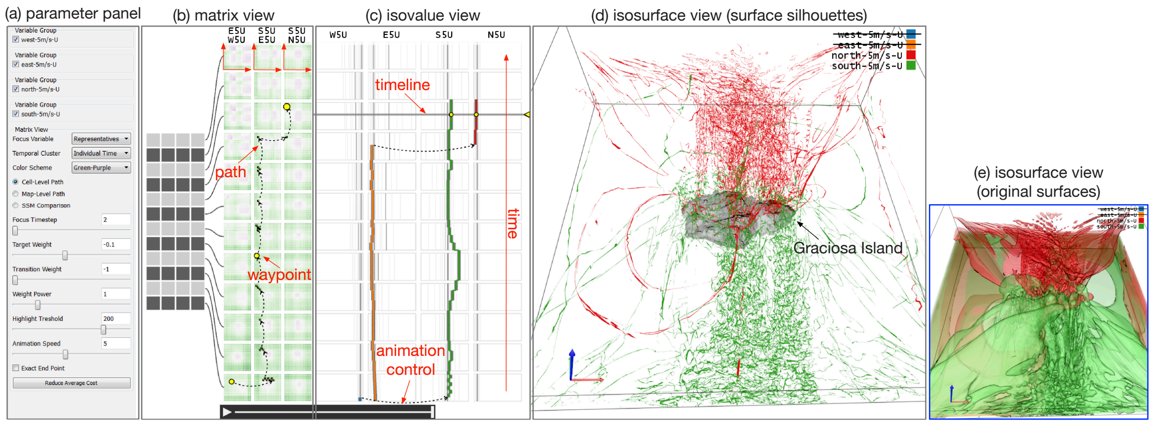
**R5. Path customization.** Users should be able to interact with the system to customize paths focusing on certain variables of interest or a specific type of relationship described in **R3**. The customized path should answer questions such as *how does a feature evolve over* *time*, *what are the surfaces related to the selected one over time*, and *what are the most different surfaces from the selected one over time?*

**R6. Path animation.** The system should produce an animation that concisely describes the comparison path with isosurface rendering. The animation should answer the questions in **R1** with more detail: it not only indicates *which* isosurfaces are (dis)similar, but also describes *how* they are (dis)similar by providing the isosurfaces in the spatial view. For effective visual comparison, the isosurfaces at each animation frame may need to be rendered in different styles, allowing the more important ones to be observed clearly.

## 5.2 Matrix of Isosurface Similarity Maps

### 5.2.1 Overview

Based on our analysis of requirements in Chapter 5.1, we design MISM as the main visual representation and interface for exploring the underlying time-varying multivariate data set. As shown in Figure 5.1, our interface consists of four components: *parameter panel*, *matrix view*, *isovalue view*, and *isosurface view*. The matrix view is the core component of our system. It displays MISM in multiple modes to allows the observation of both an overview of the data set (**R1**) and different types of relationships among variables (**R3**). The matrix of maps allows the relationships to be understood both at map-level (volumes) and cell-level (isosurfaces). With visual hints provided by the MISM display, users can easily identify the variable/isovalues of interest and interact with the matrix view to navigate the related variables/isovalues. The key function we introduce for navigating MISM is *path recommendation* (**R4** and **R5**): users can select two or more similarity maps in MISM and we recommend a traversal path that maximizes certain criteria (e.g., the smoothest, or most surprising path). In conjunction with the isovalue and isosurface views, animating the path in MISM and displaying the corresponding isosurfaces enable users to conveniently tour through the underlying data set (**R6**).



Variable Group
☑ west-5m/s-U

Variable Group
☑ east-5m/s-U

Variable Group
☑ north-5m/s-U

Variable Group
☑ south-5m/s-U

Matrix View

| | |
|---|---|
| Focus Variable | Representatives |
| Temporal Cluster | Individual Time |
| Color Scheme | Green-Purple |

● Cell-Level Path
○ Map-Level Path
○ SSM Comparison

| | |
|---|---|
| Focus Timestep | 2 |
| Target Weight | -0.1 |
| Transition Weight | -1 |
| Weight Power | 1 |
| Highlight Treshold | 200 |
| Animation Speed | 5 |

☐ Exact End Point

Figure 5.1. MISM interface. The cells in the similarity maps are colored purple/white/green for high/medium/low values. A path is specified to reveal the connections between the isosurfaces of variable U corresponding to four different wind directions using the Atmospheric ensemble data set. (a) shows the parameter panel. (b) shows the matrix view with a user-specified path. A summarized view of the matrix is displayed on the left and the representative SSMs and VSMs related to the path are displayed on the right. (c) shows the isovalue view. The blue, orange, green, and red paths correspond to the isovalues of U under the west, east, south, and north wind directions, respectively. (d) and (e) show the isosurface view with surface silhouettes and original surfaces, respectively.

As described in Chapter 5.2.2, we define three types of similarity maps in this work: the *self-similarity map* (SSM) of the isosurfaces from individual volumes, the *temporal similarity map* (TSM) of the isosurfaces from the same variable at different time steps, and the *variable similarity map* (VSM) of the isosurfaces from different variables at the same time step. The definition of similarity follows that of the ISM approach [24]. The matrix view provides four display modes to explore different types of relationships: (1) the *single-variable* mode that displays a single SSM to explore a single volume; (2) the *single-pair* mode that displays a VSM or TSM and the two corresponding SSMs for comparison of two volumes (e.g., Figure 5.14 (a)); (3) the *all-pairs* mode that displays all SSMs and VSMs for selected variables at a given time step (e.g., Figure 5.8 (a)); and (4) the *evolution mode* that displays all SSMs and VSMs for selected variables and time steps through clustering and

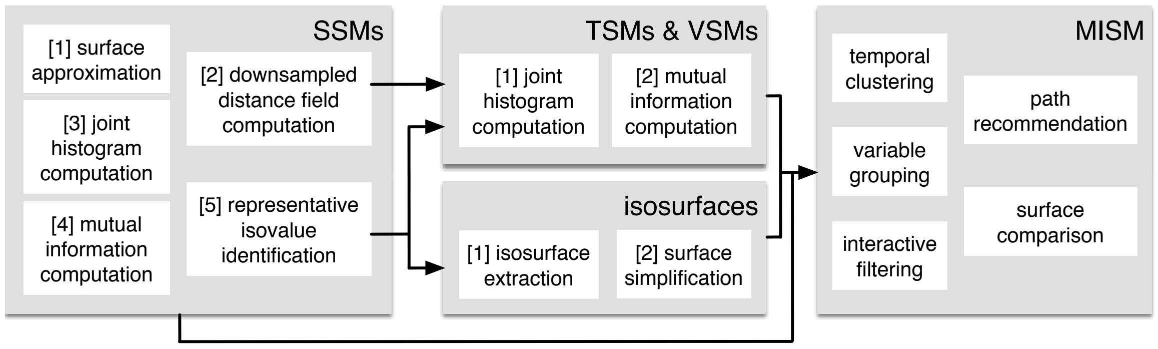

SSMs
[1] surface approximation
[2] downsampled distance field computation
[3] joint histogram computation
[4] mutual information computation
[5] representative isovalue identification
TSMs & VSMs
[1] joint histogram computation
[2] mutual information computation
isosurfaces
[1] isosurface extraction
[2] surface simplification
MISM
temporal clustering
variable grouping
interactive filtering
path recommendation
surface comparison

Figure 5.2. The diagram of our framework. Self-similarity maps (SSMs) are computed for each volume of the data, from which we compute representative isovalues and isosurfaces and further derive temporal similarity maps (TSMs) and variable similarity maps (VSMs). At runtime, SSMs, TSMs, and VSMs are used to construct MISM for users to explore the underlying data set.

filtering (e.g., Figure 5.1 (b)). Only the evolution mode displays multiple time steps, where the horizontal direction represents the selected variables and all related variable pairs, and the vertical direction represents representative time steps. SSMs of the selected variables and all related VSMs are displayed but not TSMs.

**Interaction and typical workflow.** We propose a set of interactive functions for navigating the huge matrix of similarity maps (see Figure 5.2). We cluster temporal sequences and group variables to reduce the exploration space (**R2**). We also highlight representative cells in a similarity map to attract user attention and guide the exploration (e.g., Figure 5.7). A typical workflow to explore a data set using MISM is as follows. Users will start by examining the matrix view in the evolution mode, which shows the SSMs and VSMs of all the representative time steps and variables. This provides users an overall understanding of the data set and guides them to discover the time steps and variables of interest. Then, users can simply click and drag to form a path that reveals the temporal development of the selected variables and time steps. Users can further create waypoints to edit the traversal path so that the connections to additional features can be discovered. In the evolution mode, users can easily click an SSM to enter the single-variable mode or click a VSM to enter the single-pair mode for detailed exploration. They can specify a time step to explore the relationship among all variables in the all-pairs mode as well. Finally, they can create paths to investigate the isovalues in these detailed modes.

First we describe the computation for the needed similarity maps in Chapter 5.2.2. Next, we introduce temporal clustering and variable grouping (Chapter 5.2.3), path recommendation and graph construction (Chapter 5.2.4), and silhouette-based rendering and animation (Chapter 5.2.5).

### 5.2.2 Temporal and Variable Similarity Maps

To fulfill the design requirement **R3**, we create three types of similarity maps to capture different types of relationships among the isosurfaces.

**Self-similarity map.** We refer to the previously introduced ISM (see Chapter 3) as the *self-similarity map* (SSM) because it is only concerned with isosurfaces generated from a single volume.

For a time-varying multivariate data set of $t$ time steps and $v$ variables, we need to compute $t \times v$ SSMs. Since each SSM may take hours to compute given a reasonable number of sampled isosurfaces (e.g., we use $n = 256$), we apply the GPU-accelerated approximation solution introduced in Chapter 3 to speed up the computation while maintaining the fidelity of the resulting SSM. The major steps are listed in Figure 5.2.

For each SSM, we identify $m$ representative isovalues using a greedy strategy that recursively partitions the set of isovalues and selects representative ones based on a priority queue scheme [24]. Typically, $m$ is much smaller than $n$, e.g., we use $m = 16$. We leverage a GPU to compute isosurface approximations, downsampled distance fields, and joint histograms. With that, we are able to reduce the average time to compute a single SSM from hours to a few minutes using a single GPU.

**Isosurface computation.** After the representative isovalues are identified for each volume, we implement a GPU version of the marching cubes algorithm [144] to compute the actual isosurfaces and simplify the resulting surfaces [99]. Both steps are performed on the CPU during a one-time preprocessing stage. The simplification can significantly reduce the space for storing isosurfaces without sacrificing much of the surface quality. This is important as we need to generate a total of $m \times t \times v$ representative isosurfaces for the entire time-varying multivariate data set. Using simplified isosurfaces alleviates I/O burden, making it possible for us to achieve interactive visualization and comparison of a number of isosurfaces while still maintaining good visual quality.

As an example, we can see in Figure 5.3 that our approximation (b) yields a very similar SSM as the exact marching cubes algorithm does in (a). In (d), we show the isosurfaces corresponding to the four most representative isovalues selected in (b). Comparing (a) and (b), although there are slight shiftings of the selected isovalues, those are barely visible in the surface rendering, as shown in the difference image (c). Such minor differences are neglectable as the main goal of selecting the most salient isovalues and depicting their surfaces is still achieved. With our CUDA-accelerated solution, the time cost to compute the SSM is improved by a factor of 43$\times$ (72.30s *vs.* 3,161.70s). The difference image shown in (e) indicates that surface simplification yields a close rendering result with a much less number of triangles ($3,519,952$ *vs.* $235,030$, or nearly 15$\times$ reduction).

**Temporal and variable similarity maps.** Our goal is to investigate not only the SSMs corresponding to individual volumes, but also the similarity maps between the volumes of different time steps and the volumes of different variables. Therefore, we also compute two other kinds of isosurface similarity: *temporal similarity* and *variable similarity*, extending the work of multimodal surface similarity [83] from only a single pair to all pairs of variables, and from steady to time-varying data. These similarities are computed between representative isosurfaces from the same variable at different time steps (temporal similarity), and from different variables at the same time step (variable similarity). We call the resulting similarity maps *temporal similarity maps* (TSMs) and *variable similarity*

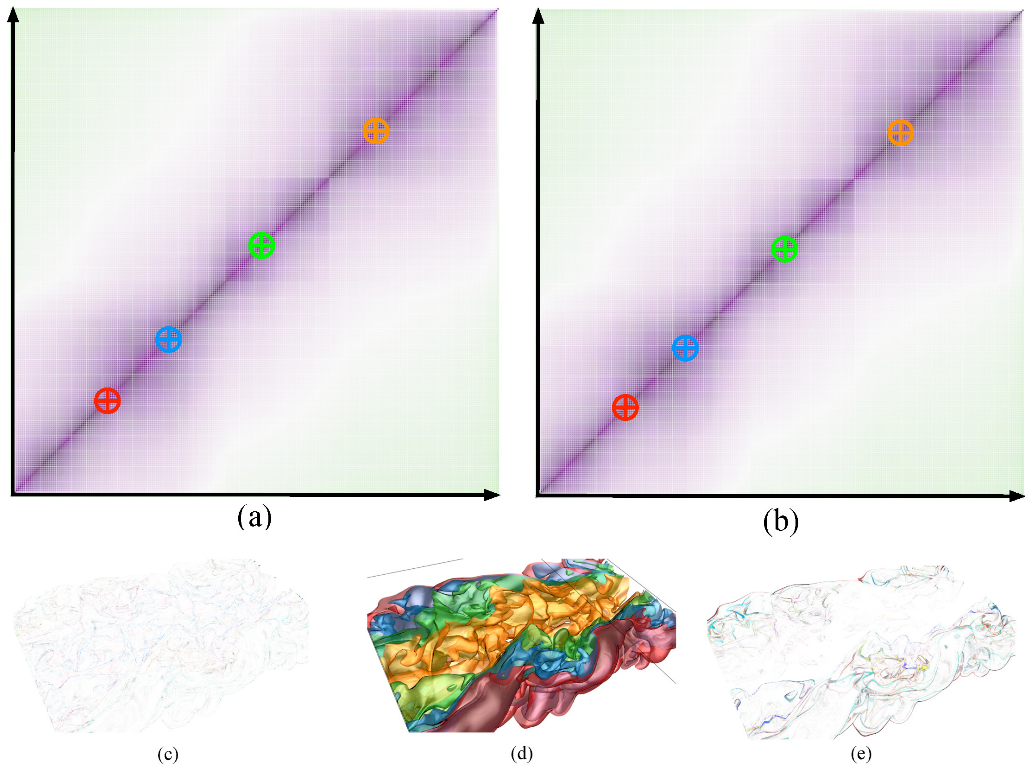


Figure 5.3. Comparing SSMs generated using (a) the actual isosurfaces against (b) our approximation with the Combustion data set (a time step of the MF variable). The marching cubes algorithm is used to extract the surfaces shown in (d) for the four most representative isovalues selected in (b). (c) shows the difference between the surfaces selected in (a) and (b), and (e) shows the difference between the surfaces selected in (b) and their simplified version.

*maps* (VSMs), respectively. A TSM or VSM is not symmetric anymore as the representative isovalues come from different volumes. Each TSM or VSM is much smaller in size compared with an SSM ($m \times m$ *vs.* $n \times n$), but the numbers of TSMs (i.e., $v \times t(t-1)/2$) and VSMs (i.e., $t \times v(v-1)/2$) are much larger than that of SSMs (i.e., $t \times v$) as we need to compute TSMs and VSMs for different pairs of time steps and variables.

### 5.2.3 Clustering and Grouping

We reduce the size of MISM along the temporal and variable dimensions through clustering temporal sequences and grouping variables, respectively. This not only allows users to quickly identify the representative time steps or variables for exploration, but also produces more compact paths and animations for efficient knowledge discovery. We define a similarity measure between volumes using TSMs or VSMs and use it to derive the similarity between two time steps or two variables. We apply affinity propagation [67] to cluster the temporal sequences based on the similarity measure. Affinity propagation automatically determines the number of clusters, which naturally reflects how frequently the variables change along time in a data set. For variables, we use *k*-means clustering to group the variables into the desired number of groups. In the following, we only describe our similarity measure for temporal clustering, as the detail for variable grouping is the same.

**Similarity measure.** We evaluate the similarity of two volumes based on the similarities among their isosurfaces. Two volumes are considered to be similar if for each isosurface in one volume, a similar isosurface can be identified in the other volume. Our similarity measure is analogous to the mean of closest point distances defined on two curves by considering each volume as a curve and each isosurface of the volume as a point on the curve. Formally, the similarity of an isosurface $\mathbf{S}'$ and a volume $\mathbf{V}$ is defined as

$$\mathscr{S}(\mathbf{S}', \mathbf{V}) = \min_{\mathbf{S} \in \mathbf{V}} \mathscr{S}(\mathbf{S}', \mathbf{S}), \tag{5.1}$$

where the similarity between two isosurfaces, $\mathscr{S}(\mathbf{S}', \mathbf{S})$, can be looked up from TSMs or VSMs. The similarity of two volumes $\mathbf{V}_i$ and $\mathbf{V}_j$ is defined as the weighted average of the similarity of each isosurface in one volume to the other volume, i.e.,

$$\mathscr{S}(\mathbf{V}_i, \mathbf{V}_j) = \frac{\sum_{\mathbf{S}_p \in \mathbf{V}_i} w_{\mathbf{S}_p} \mathscr{S}(\mathbf{S}_p, \mathbf{V}_j)}{\sum_{\mathbf{S}_p \in \mathbf{V}_i} w_{\mathbf{S}_p}} + \frac{\sum_{\mathbf{S}_q \in \mathbf{V}_j} w_{\mathbf{S}_q} \mathscr{S}(\mathbf{S}_q, \mathbf{V}_i)}{\sum_{\mathbf{S}_q \in \mathbf{V}_j} w_{\mathbf{S}_q}}, \tag{5.2}$$

where the weight $w_{\mathbf{S}_p}$ for an isosurface $\mathbf{S}_p$ is derived based on its representativeness ranking, which is used for selecting the most representative isovalues Note that the similarity between any pair of volumes is in $[0,1]$, which has the same range as the similarity between any pair of isosurfaces.

**Temporal clustering.** Temporal clustering evaluates the combined similarity between two time steps as the summation of similarities calculated for each of the variables using Equation 5.2. Affinity propagation is applied to cluster the time steps based on their combined similarities. For each cluster, the clustering algorithm identifies one exemplar, which is considered as the representative time step for that cluster.

### 5.2.4 Path Recommendation

MISM has a two-tiered structure: *maps* at the coarse matrix level and *cells* at the fine map level. The map-level captures the overall relationships among volumes, while the cell-level allows the detailed relationships among isosurfaces to be discovered. Given two user-specified maps (cells) as the start and end points, *path recommendation* identifies a series of intermediate maps (cells) to construct a path for traversal. We create an animation for the generated path, showing the isosurfaces corresponding to a map (cell) at one animation frame. Users can adjust the weights of different terms to define the desired path. They may visit the maps or cells to discover affinity relationships or compare distinct features. The differences between frames can be minimized or maximized for generating a smooth animation or increasing information gain. They can specify the start and end points of a path in the matrix view, and drag any point along the path to add waypoints. We identify a path that minimizes the total cost between the user-specified points.

We introduce the following types of paths: (1) *map-level paths* for the evolution mode, (2) *cell-level paths* for all the four modes, and (3) *variable traversal paths* for only the all-pairs mode. All three types of paths are built with similar constructions. In the following, we discuss map-level paths in detail. For the other two, we only explain their differences with respect to map-level paths.

We denote a similarity map between two variables $A$ and $B$ at time step $\tau$ as $\mathbf{M}_{AB,\tau}$ and their corresponding volume as $\mathbf{V}_{A,\tau}$ and $\mathbf{V}_{B,\tau}$. Note that $\mathbf{M}_{AB,\tau}$ can refer to an SSM when $A = B$ or a VSM when $A \neq B$. However, it does not refer to a TSM (which is not displayed in the matrix view) as the indices do not specify two different time steps. We use $\mathbf{M}_{AB,\tau}[i, j]$ to denote a cell in $\mathbf{M}_{AB,\tau}$ at the $i$-th row and the $j$-th column, and $\mathbf{S}_{A,\tau,i}$ and $\mathbf{S}_{B,\tau,j}$ to denote the two isosurfaces corresponding to the $i$-th row and $j$-th column, respectively. We may ignore $\tau$ in the notation for simplicity when time is not discussed. We define the difference $\mathscr{D}(,)$ between two elements as $1 - \mathscr{S}(,)$.

**Map-level path.** We identify the map-level path between two similarity maps as the shortest path in a map-level graph. A *map-level graph* is a directed graph whose nodes are all SSMs and VSMs and whose edges are all the possible transitions from one map to another, as illustrated in Figure 5.4 (a). Specifically, we consider two kinds of transitions: (1) *variable transition* between similarity maps at the same time step sharing at least one common variable, as shown by the blue arrows; and (2) *temporal transition* between similarity maps at neighboring time steps, as shown by the red arrows.

Weighing the edges appropriately is critical to obtaining a path that shows the desired features. In our approach, the weight of an edge corresponding to a transition is a linear combination of the *target cost* $\mathscr{C}_{tg}$ and the *transition cost* $\mathscr{C}_{ts}$ raised to a user-specified power exponent $\alpha$: $(w_{tg}\mathscr{C}_{tg} + w_{ts}\mathscr{C}_{ts})^{\alpha}$, where $w_{tg}$ and $w_{ts}$ are the weights of $\mathscr{C}_{tg}$ and $\mathscr{C}_{tg}$, respectively. We include $\alpha$ to further distinguish the edge costs, so that the shortest path is less likely to end up with a path with a larger average cost but a smaller number of edges. Consider the transition $\mathbf{M}_{AB} \rightarrow \mathbf{M}_{BC}$ (from the yellow node to the green node)

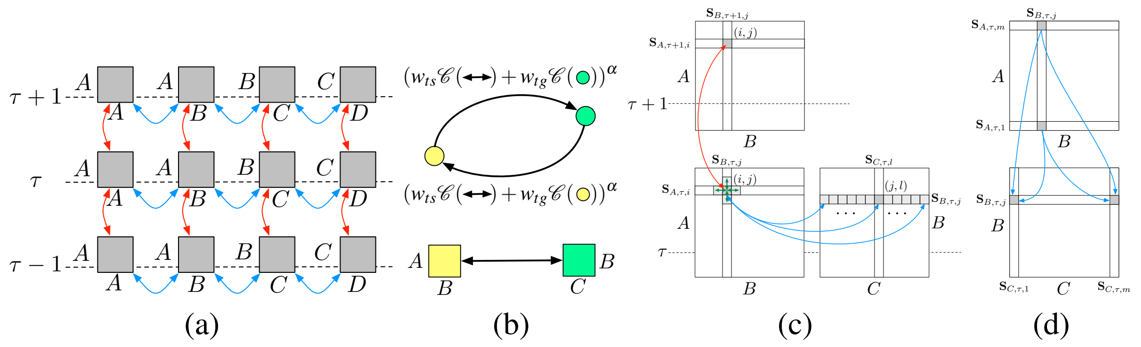


Figure 5.4. Top: map-level graph construction. (a) transitions in map-level graphs. (b) edge weights corresponding to two transitions between maps $\mathbf{M}_{AB}$ and $\mathbf{M}_{BC}$. Bottom: transitions between cells. (c) transitions in cell-level graphs. (d) transitions in variable traversal graphs.

in Figure 5.4 (b). The target cost $\mathscr{C}(\mathbf{M}_{BC})$ is the difference $\mathscr{D}(\mathbf{V}_B, \mathbf{V}_C)$ between the two corresponding volumes $\mathbf{V}_B$ and $\mathbf{V}_C$. When the target cost is weighed positively, the shortest path is more likely to visit the maps corresponding to variables with similar structures by minimizing the cost. When the target cost is weighed negatively (we use $1-\mathscr{C}(\mathbf{M}_{BC})$), the resulting path tends to visit the variables with different behaviors for the most surprise. The transition cost is the penalty when we replace the isosurfaces to display for $\mathbf{M}_{AB}$ to those for $\mathbf{M}_{BC}$. It can be weighed positively in the linear combination, so that the shortest path minimizes the transition cost and maintains a smooth animation between neighboring frames; otherwise, it can be weighed negatively using $1-\mathscr{C}(\mathbf{M}_{AB} \rightarrow \mathbf{M}_{BC})$ to visit the maps with diverse information.

A variable transition from a VSM $\mathbf{M}_{AB,\tau}$ to another $\mathbf{M}_{BC,\tau}$ indicates that the focus of analysis shifts from one pair of variables ($A$ and $B$) to another pair ($B$ and $C$). After the transition, we replace the isosurfaces of $A$ with the ones of $C$ in the isosurface view. Therefore, we weigh the corresponding edge by the difference of their volumes, i.e., $\mathscr{D}(\mathbf{V}_{A,\tau}, \mathbf{V}_{C,\tau})$. The edge weight for transitions between a VSM and an SSM can be defined similarly by letting $A = B$. For coherence, variable transitions should take place only if necessary.

To avoid the paths from jumping back and forth among the variable pairs, we include a sufficiently large *jumping cost* in the variable transition.

A temporal transition between two VSMs $\mathbf{M}_{AB,\tau}$ and $\mathbf{M}_{AB,\tau+1}$ describes the evolution of the two variables $A$ and $B$ over time. The corresponding edge weight is defined as $(\mathscr{D}(\mathbf{V}_{A,\tau}, \mathbf{V}_{A,\tau+1}) + \mathscr{D}(\mathbf{V}_{B,\tau}, \mathbf{V}_{B,\tau+1}))/2$.

**Cell-level path.** Similar to the map-level path, the cell-level path is the shortest path between two cells in a directed *cell-level graph*. Each node in the cell-level graph represents one cell and each edge represents one transition. For the cell-level graph, we consider three kinds of transitions, as shown in Figure 5.4 (c): (1) *variable transition* (the blue arrows) between cells in two similarity maps at the same time step sharing at least one common variable; (2) *temporal transition* (the red arrow) between cells in two similarity maps corresponding to the same pair of variables at different time steps; and (3) *isovalue transition* (the green arrows) between cells in the same similarity map. Similarly, the edge weight is a linear combination of transition cost and target cost. The target cost is given by the difference of the two isosurfaces corresponding to the target cell. The transition cost is defined by the difference between the isosurfaces that are not shared by the two cells. For example, for a variable transition $\mathbf{M}_{AB,\tau}[i,j] \to \mathbf{M}_{BC,\tau}[j,k]$, the transition cost is the difference between $\mathbf{S}_{A,\tau,i}$ and $\mathbf{S}_{C,\tau,k}$, i.e., $\mathscr{D}(\mathbf{S}_{A,\tau,i}, \mathbf{S}_{C,\tau,k})$.

We further provide two options for users to improve the cell-level path. First, we allow users to specify the exact endpoint as cells, or simply select a map that contains the endpoint. When a map is specified, the cell with minimum cost is automatically identified on the map. In this case, we create an edge from each cell in the map to a dummy node with zero weight and compute the shortest path. Second, we design a scheme to further reduce the average cost between cells. The shortest path often tends to minimize the total cost along the path by reducing the number of cells. It is possible to identify a path with a minimum number of undesired transitions. Therefore, finding a path with the smallest average weight between nodes is more desired in our scenario. Since this problem is NP-

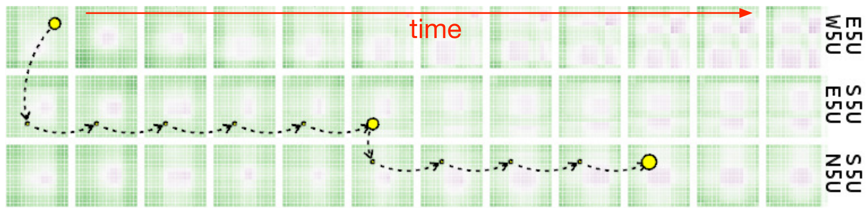


Figure 5.5. The path with the same start point and parameters as Figure 5.1 (b). The example shown here uses the original cell-level graph instead of the reduced weight graph.

hard, we provide an approximate solution here. We start with the shortest path identified by Dijkstra's algorithm and compute the average weight $\bar{w}$ along the path. Then, we create a cell-level reduced weight graph. This graph contains the same edges as the original cell-level graph, but the weight of each edge is reduced by $\bar{w}$. The edges with negative weights are eliminated by assigning a small weight (0.001 in our implementation) to them. Note that the edges with weights smaller than $\bar{w}$ have a minimum cost now. This allows a longer path with more desirable transitions to be identified as the shortest path in the reduced weight graph. Figure 5.1 (b) and Figure 5.5 show two cell-level paths produced by the reduced weight and original graphs, respectively. In Figure 5.5, we can see that without reducing the weights, the path only visits one cell in each map at similar locations, even under the setting of $w_{tg} = 0.1$ and $w_{ts} = -1$, which suppose to show more distinguished isosurfaces. In contrast, with the reduced weight graph, the path usually visits multiple cells in each graph to increase the information gained, as shown in Figure 5.1 (b). For a quantitative comparison, we evaluate the average weight of the two paths in the original graph. The path produced by the reduced weight graph can effectively bring down the average weight from 0.227 to 0.172.

**Variable traversal path in all-pairs mode.** A variables traversal path visits a chain of variable pairs that covers every variable for a complete understanding of relationships among variables at a given time step. For example, this chain can be $< A, B > \rightarrow <$

$B,C> \rightarrow < C,D> \rightarrow < D,A>$ for a data set with four variables. Given a user-specified cell (and therefore a pair of variables to start), we determine the order of variables to visit using a greedy strategy: at each step, we select the unvisited variable that is most similar to the current variable.

The variable traversal path is identified in a *variable traversal graph* which is constructed similarly as the cell-level graph with variable transitions and isovalue transitions. The definitions of transition cost and target cost are the same as the ones described for the cell-level graph. The only difference is that variable transitions are constrained to take place between only boundary cells following the order of the maps being visited. For example, consider the two pairs of variables $<A,B>$ and $<B,C>$ to be neighbors along the path, as shown in Figure 5.4 (d). We only allow transitions from the boundary cells $\mathbf{M}_{AB,\tau}[1,j]$ and $\mathbf{M}_{AB,\tau}[m,j]$ in map $\mathbf{M}_{AB,\tau}$ to the boundary cells $\mathbf{M}_{BC,\tau}[j,1]$ and $\mathbf{M}_{BC,\tau}[j,m]$ in map $\mathbf{M}_{BC,\tau}$ that correspond to the same isosurface $\mathbf{S}_{B,\tau,j}$. This forces the path to enter and leave a similarity map at the boundaries to build a more complete path visiting a large portion of value range for each variable.

### 5.2.5 Surface Rendering and Animation

We render the isosurfaces using an approach based on per-pixel link lists [230] for real-time order independent transparency. The rendering is performed in two passes. The first pass generates a link list for each pixel to store the fragments that are rendered to that pixel. The second pass sorts fragments at the same pixel according to their depth and blends the fragments in the sorted order. Inspired by the silhouette-based rendering [49], we only render the silhouettes for the less important isosurfaces to reduce visual clutter. We combine these two approaches by checking whether a fragment belongs to the silhouettes before storing the fragment in the link lists. The fragments of the less important isosurfaces that are not related to the silhouettes are discarded to reduce the storing and sorting effort. Using our approach, rendering some surfaces as silhouettes not only allows clearer

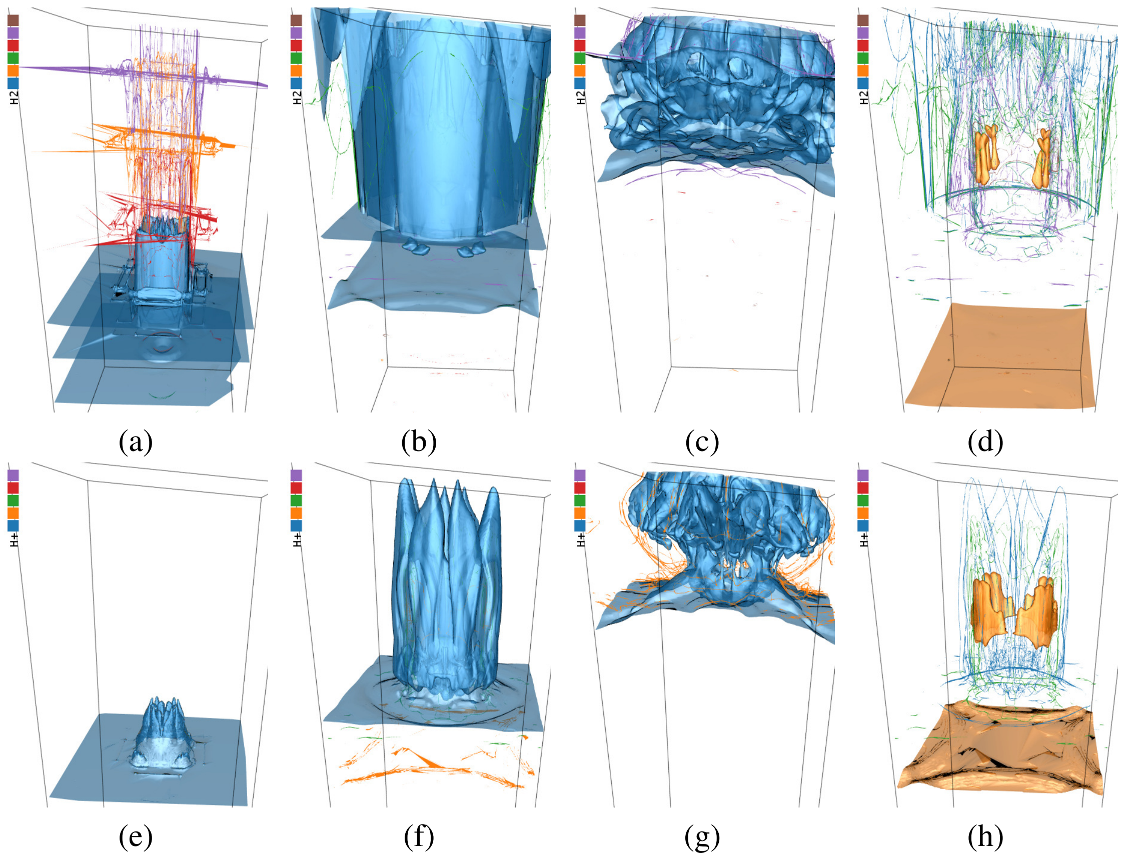


Figure 5.6. Isosurfaces corresponding to map-level paths using the Ionization data set. (a) to (c) show the representative isosurfaces for H2 at time steps 31, 98, and 181, respectively. (d) shows the same isosurfaces as (b) but focusing on the orange isosurface. (e) to (g) show the same as (a) to (c) but for H+.

observation but also has a lower cost than rendering all surfaces in full.

An animation will be produced when a path is specified in the matrix view. As shown in Figure 5.1, with the evolution mode of the matrix view, we also display the isovalue view to show the evolution of isovalues corresponding to the rendered isosurfaces. We synchronize the animation across the matrix, isovalue, and isosurface views. The path is displayed in both the matrix and isovalue views with the current animation step highlighted. We display a timeline in the isovalue view. Users can drag it to play a specific animation step. They can also drag the highlighted isovalues in the isovalue view to create additional waypoints and modify the path. When multiple variables are involved in an animation, the isosurfaces are colored according to their corresponding variables, as indicated by the legends in the isosurface view (e.g., Figure 5.1 (d)). We create an animated transition for each variable transition on the path to gradually transit from one variable to another. When a single variable is used in an animation, we color the isosurfaces according to their representativeness. The representativeness ranking is indicated by the legends shown in the isosurface view, with the most representative one displayed on the left (e.g., Figure 5.6 (a)). By default, we render the complete isosurface for the most representative one and the silhouettes for the others. Users may switch the focus to render any representative isosurface as a complete one.

TABLE 5.1

AVERAGE TIMING RESULTS USING THE CPU/GPU CLUSTER

| data set | dimension $x \times y \times z \times v \times t$ | distance field $x \times y \times z$ | SSM | extraction | simplification | reduction | TSM | VSM | total |
|---|---|---|---|---|---|---|---|---|---|
| Atmospheric | $360 \times 360 \times 112 \times 32 \times 13$ | $45 \times 45 \times 14$ | 26.85s | 8.31s | 94.20s | 7.56× | 0.0953s | 0.0941s | 24.1h |
| Climate | $360 \times 66 \times 27 \times 2 \times 1200$ | $120 \times 33 \times 9$ | 4.85s | 0.18s | 1.82s | 9.39× | 0.0810s | 0.0778s | 10.9h |
| Combustion | $480 \times 720 \times 120 \times 5 \times 122$ | $60 \times 90 \times 15$ | 50.66s | 8.11s | 56.44s | 10.72× | 0.2290s | 0.2275s | 23.5h |
| Ionization | $600 \times 248 \times 248 \times 8 \times 199$ | $75 \times 31 \times 31$ | 25.99s | 4.34s | 14.40s | 12.42× | 0.1936s | 0.1919s | 20.1h |

The average timing results for computing an SSM, an isosurface and its simplified version, a TSM, and a VSM using a CPU/GPU, and the total computation time for the entire data set using the CPU/GPU cluster.

## 5.3 Results and Discussion

### 5.3.1 Data Sets and Timing Performance

We used several time-varying multivariate data sets with different sizes and characteristics for our experiment. The data sets and timing performance are reported in Table 5.1. The timing was collected using a CPU/GPU cluster (8 Quantum TXR231-1000R servers each with a dual Intel E5-2650 12-core CPU @ 2.2 GHz, 128 GB RAM, and 4 NVIDIA TITAN X GPUs). The computation of the distance fields and similarity maps (SSMs, TSMs, and VSMs) was performed in the GPU, while isosurface extraction and surface simplification were performed in the CPU. The timing for SSM ($256 \times 256$) includes the time to compute distance fields, joint histogram, and mutual information. The time costs for surface extraction and simplification mainly depend on the complexity of the surface and the number of triangles involved. The timing for TSM or VSM ($16 \times 16$) includes the time to compute joint histogram and mutual information. The time cost to compute a TSM or VSM mainly depends on the size of distance fields. In our experiment, since the computation of similarity maps or isosurfaces consists of largely independent tasks, we use 8 CPU cores and 8 GPUs of the CPU/GPU cluster to speed up the computation in parallel, reducing the total computation time for each data set to a day or less.

We further examine the accuracy and speed-up of our similarity map computation. Our solution introduces two major variants to speed up the computation: using approximation (APP) to replace GPU-accelerated marching cubes (MC) for surface generation and using downsampled (DS) distance fields instead of the ones with original (ORI) resolution. We evaluate the performance of different combinations of these two variants (i.e., DS-MC, ORI-APP, and DS-APP) by comparing them to the ground truth (ORI-MC). The accuracy is evaluated by the mean of squared error (MSE). Giving two sets of values $\mathbf{X} = \{x_1, \cdots, x_n\}$ and $\mathbf{X}' = \{x'_1, \cdots, x'_n\}$, MSE is computed as

TABLE 5.2

MEAN SQUARED ERROR (MSE) AND SPEED-UP (SU) FOR COMPUTING SSMS USING DIFFERENT CONFIGURATIONS.

| | DS-MC | | ORI-APP | | DS-APP | |
|---|---|---|---|---|---|---|
| data set | MSE | SU | MSE | SU | MSE | SU |
| Atmospheric | 0.0006 | 23.11× | 0.2191 | 1.06× | 0.1637 | 110.67× |
| Climate | 0.0046 | 13.13× | 0.0060 | 1.02× | 0.0052 | 30.50× |
| Combustion | 0.0001 | 48.86× | 0.0584 | 1.23× | 0.0474 | 182.48× |
| Ionization | 0.0134 | 43.69× | 0.0831 | 1.27× | 0.0489 | 299.26× |

$$\mathrm{MSE}(\mathbf{X},\mathbf{X}') = \frac{1}{n}\sum_{i=1}^{n}|x_i - x'_i|^2. \tag{5.3}$$

For the Climate data set, we collect the results using ten sample time steps, and for the others, we use five sample time steps. Note that this already contains 1.3 million to 2 million similarity values per data set so that we can draw a reliable conclusion. In Table 5.2, we can see that using downsampled distance fields produces mostly the same similarity values, with the largest MSE being 0.0134 for the Ionization data set. Surprisingly, we find that DS-APP outperforms ORI-APP in terms of MSE for all the data sets, probably because distance fields of the original resolution may be more sensitive to the errors introduced by the approximation. DS-APP produces reasonable results with MSE smaller than 0.05 except for the Atmospheric data set (0.1637). In terms of efficiency, DS itself speeds up the computation by more than 10×. APP does not lead to large speed boost with ORI, but it can further increase the performance of DS, resulting in 30× to 300× speed-up.

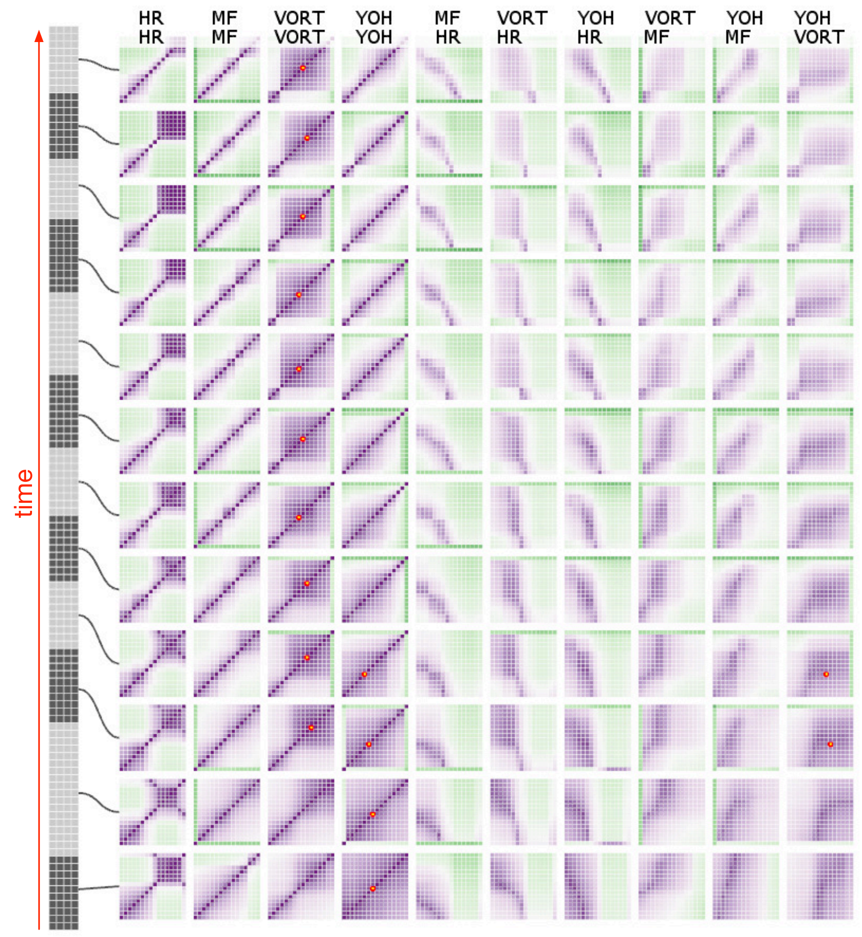


Figure 5.7. Overview of the evolution mode of the matrix view with four variables (HR, MF, VORT, and YOH) of the Combustion data set.

### 5.3.2 Case Studies with Expert Evaluation

**Combustion data set.** This data set comes from direct numerical simulation of temporally evolving turbulent non-premixed flames. The simulation generates five variables: heat release (HR), mixture fraction (MF), vorticity (VORT), mass fraction of hydroxyl radical (YOH), and scalar dissipation rate (CHI). Initially, the thin planar scalar layers are placed in the middle of the computational domain. The layers evolve into complex isosurfaces as they interact with the surrounding turbulence. Combustion reactions occur within the scalar layers.

We invited a domain expert with 20 years of experience in turbulent Combustion modeling and simulation to explore this data set using MISM. The exploration started with the evolution mode of the matrix view with four variables (HR, MF, VORT, and YOH), as shown in Figure 5.7. The expert commented that *the similarity maps at different representative time steps are able to summarize the overall data characteristics*. For each individual variable, at earlier time steps, since the scalar layers are planar, all scalars are well correlated, and most regions of the SSMs in the bottom rows are purple. As time goes on, these scalar layers are distorted by turbulence. Meanwhile, the scalar layers are thickened since small-scale turbulence enhances scalar mixing. As the scalar layers are thickened and interact with turbulence, the isosurfaces corresponding to lower and higher values become dissimilar. Therefore, the positive correlations (purple cells) only appear along the diagonals of the SSMs in the top rows. The expert further pointed out that *the matrix view demonstrates the relationship development of YOH and MF over time*. YOH and MF are perfectly correlated, when Combustion reaction time scales are much shorter than turbulence time scales. This condition is closely satisfied at later time steps, as the YOH-MF VSMs in the top rows show a strong correlation near the diagonal regions. On the contrary, at earlier time steps, the fast chemistry condition is not satisfied. This leads to no evident correlation of these two variables at earlier time steps. The SSMs in the bottom rows confirm this by exhibiting more uniform correlations. In addition, the expert found that *the relationship between VORT and MF is interesting*, since the VORT-MF VSMs gradually develop into a pattern with a purple upper-left triangle and a green lower-right triangle. The expert stated that *the physical origin of this trend could not be fully understood at this moment, but it was likely related to the effects of HR on turbulence*. This demonstrates the potential of our work for revealing physics behind the data set.

Then, the expert analyzed the relationships among four variables (VORT, CHI, MF, and YOH) at time step 111 using the all-pairs mode of the matrix view. Time step 111 is the first time step of the last temporal cluster. The all-pairs mode facilitates better comparison

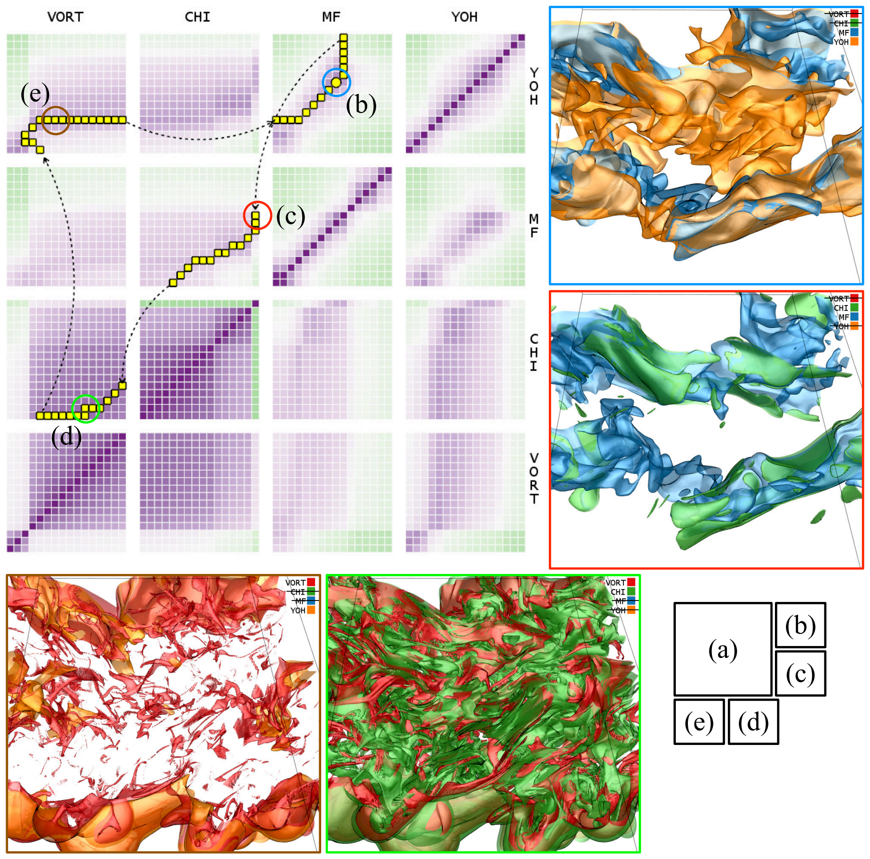


Figure 5.8. Variable traversal paths using time step 111 of the Combustion data set. (a) shows a path with a positive target weight ($w_{tg} = 1$, $w_{ts} = 0.1$, and $\alpha = 4$). (b) to (e) show four animation frames corresponding to the path in (a).

of the VSMs. In Figure 5.8 (a), the expert found that *the pattern of the YOH-MF VSM is different from others*. As already observed in the evolution mode, the diagonal of YOH-MF VSM is mostly purple, indicating *strong correspondence between the value ranges in YOH and MF*. The only exception is the top-right corner corresponding to larger isovalues of YOH and MF, which is mostly green. This means that *the larger isovalues of these two variables are less relevant to each other*. The expert explained that the value of YOH is the highest in the reaction zone and decreases toward the fuel and oxidizer sides, which correspond to the lower and higher values of MF, respectively. Therefore, there is no evident correlation of YOH and MF at higher values of each variable. The expert selected a purple cell in this VSM, as highlighted in the blue circle, to generate a variable traversal path. A positive target weight ($w_{tg} = 1$) was used to identify a path demonstrating similar isosurfaces for each of the variable pairs: YOH and MF, MF and CHI, CHI and VORT, and VORT and YOH. A small transition weight ($w_{ts} = 0.1$) was used to maintain the minimum smoothness between the consecutive animation frames. Figure 5.8 (b) to (e) show four animation frames presenting the similar isosurfaces. The similarity of isosurfaces seen in Figure 5.8 (c) indicates that *the variations of CHI on isosurfaces of MF occur mainly at large scales, which happens mostly for thin scalar layers with high values of CHI*. The regions where the CHI and MF isosurfaces collapse correspond to the thin layers with steep scalar gradients (high values of CHI). In Figure 5.8 (d), while the isosurfaces of CHI and VORT are complex, the expert pointed out that *low CHI regions correspond to high VORT regions, because strong vortical motions tend to mix scalars and reduce their gradients*.

Finally, the expert switched to the evolution mode of the matrix view to further investigate the relationship between YOH and MF. Two purple cells were specified as the starting and ending points in the YOH-MF VSMs. Two paths were generated for comparison: one with a positive target weight ($w_{tg} = 1$) and the other with a negative target weight ($w_{tg} = -1$). The transition weight $w_{ts}$ was set to 0 to eliminate the influence of transition cost, and $\alpha$ was set to 4 to allow more edges with smaller cost. Figure 5.9 (a) shows the path with the positive target weight and the isosurfaces in three animation steps along the path. The path passes only the purple regions near the centers of the VSMs, corresponding to the middle range isovalues of YOH and MF. In the isosurface view, it can be seen that *the blue isosurfaces occupy similar regions as the corresponding orange ones at all the three time steps*, due to the correlation of YOH and MF. In contrast, the path with the negative target weight goes through the green regions at the bottom-right corners of the VSMs, even if it starts and ends at purple cells, as shown in Figure 5.9 (b). This path mostly relates to the high MF isosurfaces and the low YOH ones. In the isosurface view, it can be seen that *the blue and orange isosurfaces are disjoint in the space*. As the wrinkling history of

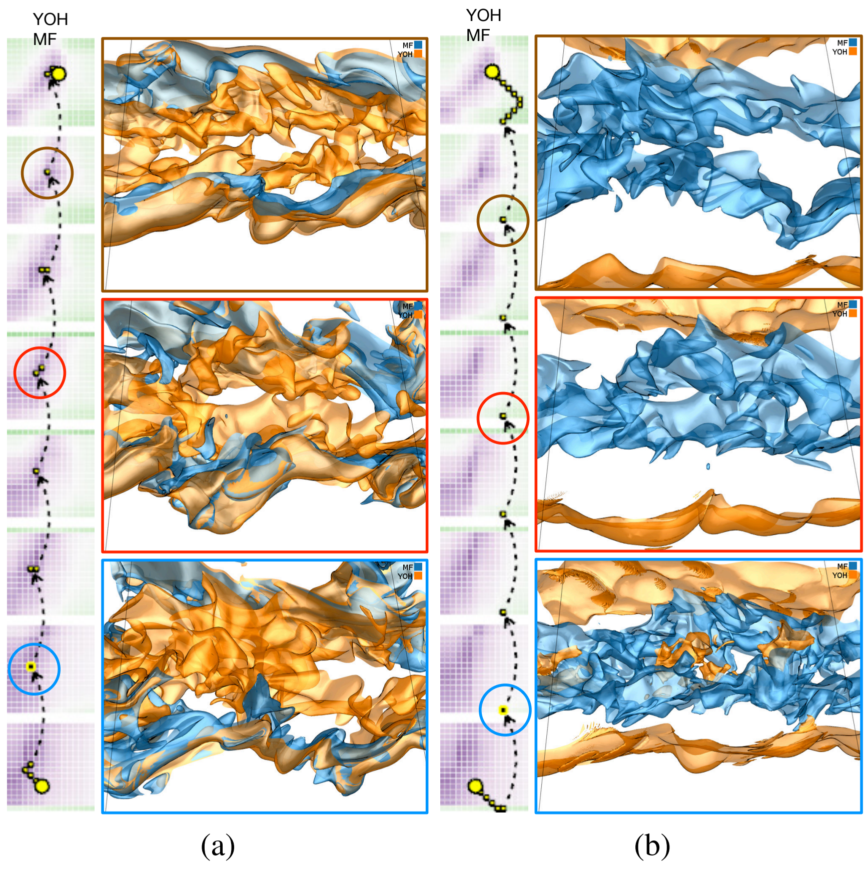


Figure 5.9. Cell-level paths in YOH-MF VSMs using the Combustion data set. (a) shows a path with a positive target weight ($w_{tg} = 1$, $w_{ts} = 0$, and $\alpha = 4$) and three corresponding animation frames. (b) shows a path with a negative target weight ($w_{tg} = -1$, $w_{ts} = 0$, and $\alpha = 4$) and three corresponding animation frames.

isosurfaces due to turbulence is different for isosurfaces present in different regions, the dissimilarity is expected.

**Atmospheric data set.** A set of 3D numerical experiments is performed to explore the mechanical impacts of Graciosa Island on the dry boundary layer evolution under varying wind speed and surface friction setups. The simulations are conducted using the Cloud Model 1 (CM1 release 19 [25]). The full data set has 32 ensemble simulations which are comprised of four basic wind speeds (1, 5, 10, and 20 m/s), four wind directions (west,

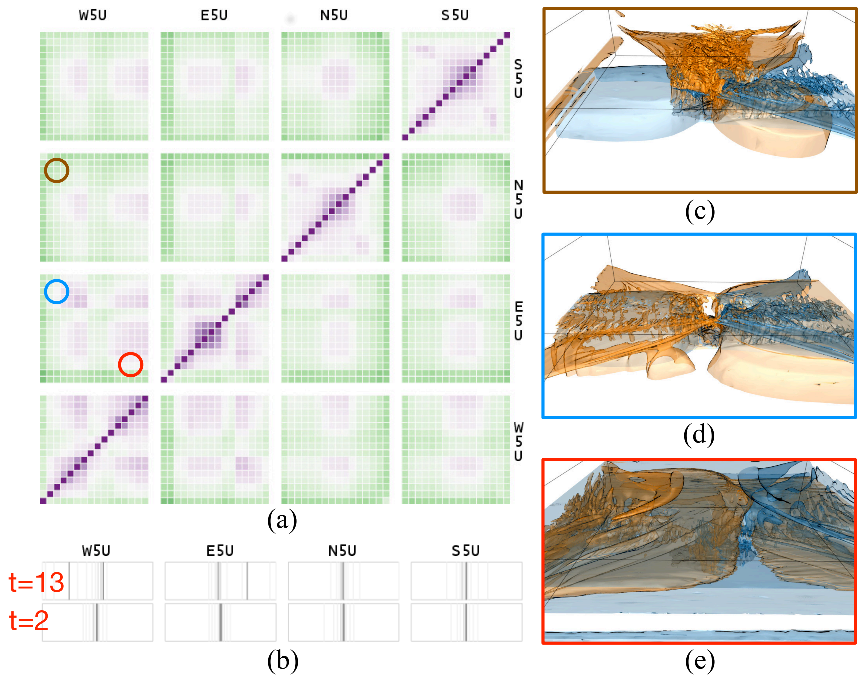


Figure 5.10. Comparison of four wind directions with friction and wind speed 5 m/s using the Atmospheric ensemble data set. (a) shows the all-pairs mode of the matrix view at time step 13. (b) shows the isovalue view at time steps 2 (bottom) and 13 (top). (c) to (e) show two isosurfaces of variable U corresponding to (c) small absolute wind speeds of the west and north, (d) small absolute wind speeds of the west and east, and (e) large absolute wind speeds of the west and east.

east, north, and south), and two bottom boundary conditions (friction/no friction). Two variables, the U and W components of the wind flow, are studied. To distinguish the variables under different simulation setups, we add the initials of the wind directions and speeds to the names of variables. For example, "E5U" stands for variable U with the east wind direction and speed 5 m/s.

We invited a domain expert with ten years of experience in Atmospheric sciences to evaluate MISM using this data set. Since friction effects are more pronounced in lower wind speeds, the expert focused on the 5 m/s case to visualize and analyze flow variability. The expert first investigated the evolution of variable U under different wind directions with friction and 5m/s wind setup. Figure 5.10 (a) shows the matrix view in the all-pairs mode at time step 13. The SSMs at the diagonal show that *the west and east have similar patterns, and so do the SSMs of the north and south.* In Figure 5.10 (b), the isovalue view shows a similar relationship. The representative isovalues distribute similarly for the four wind directions at time step 2 (bottom). In contrast, at time step 13 (top), the distributions form two different patterns. The expert explained that this means from time step 2 to 13, *the west and east show much larger flow variability compared to the north and south, which is attributed to the west-east orientation of the island located at the center of the simulation domain*. The representative isovalues spread out in different directions for the west and east, but the representatives of the south and north are still similar to those in the first time step. Figure 5.10 (c) shows that the isosurface of the north (orange) has quite a different shape from that of the west (blue). Unlike the blue isosurface, the orange one covers both the upper and lower portions of the domain, although the isovalue is small. In contrast, the isosurfaces corresponding to small absolute wind speeds of the west (blue) and east (orange) have similar shapes, as shown in Figure 5.10 (d). Both isosurfaces reside in the lower portion of the domain, although their orientations are opposite. By selecting the isovalues representing larger wind speeds (as highlighted by the red circle in Figure 5.10 (a)), the expert found that the corresponding isosurfaces are also similar, as shown in Figure 5.10 (e). These two isosurfaces occupy mostly the entire domain as the wind gets stronger. The expert commented that *the east wind direction seems to be more interesting. It demonstrates more variation on the surfaces and in the isovalue view,* due to a stronger island effect on incoming flow in this configuration.

Then, the expert compared four variables within the east wind direction: E5U, E5W, E1U, and E1W (i.e., variables U and W with wind speeds 5 m/s and 1 m/s). Only one temporal cluster is produced by affinity propagation, indicating *a consistent pattern over time*. Therefore, the expert used all individual time steps to explore their temporal relationships. Figure 5.11 shows the overview of the evolution mode of the matrix view. In general, the

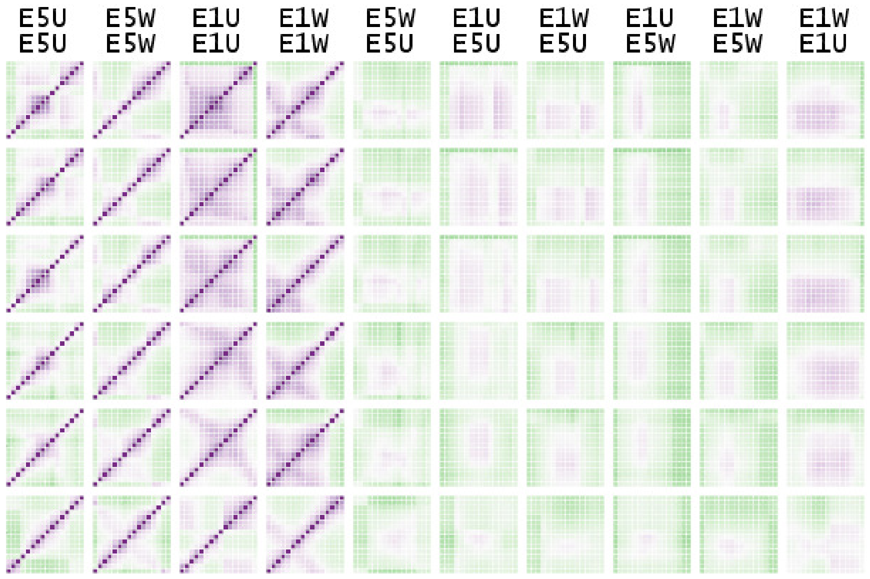


Figure 5.11. Overview of the evolution mode of the matrix view with two variables (U and W) produced under two wind speeds (5 m/s and 1 m/s) using the Atmospheric ensemble data set.

expert found that the patterns of matrices in different rows are similar, which confirms the temporal clustering results. Note that, for E5U and E1U, the purple regions in the middle value range are growing over time. The expert commented that *this is indicative of the non-steady simulation behavior*. By displaying the isosurfaces, the expert found that the purple region corresponds to surfaces covering the entire domain. This trend indicates that, *at the later time steps, the surfaces of different value ranges of U tend to cover more space and become more similar to each other*. Figure 5.12 (a) shows two isosurfaces of a larger isovalue of E1U at time step 2 (blue) and time step 13 (orange), respectively. It can be seen that the orange surface covers almost the entire domain, but the blue one is mainly located at the center of the domain.

The temporal development of variable W with wind speeds 5 m/s and 1 m/s seems to be slightly different. While the pattern in column E1W is more stable, the purple region in E5W is shrinking over time. This indicates that, unlike variable U, *the isosurfaces of*

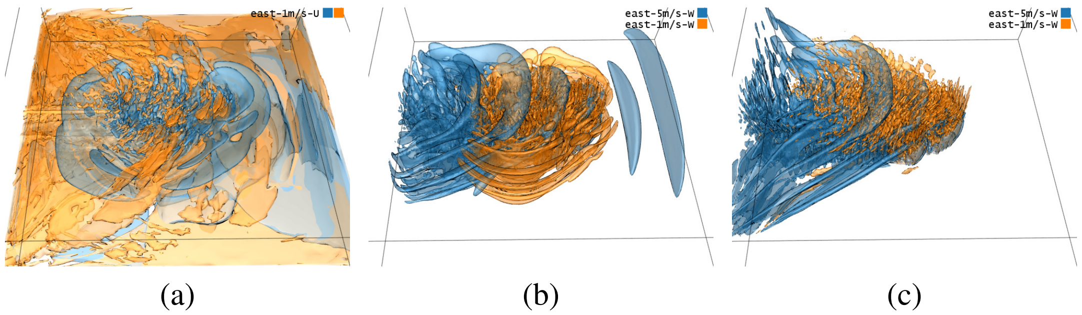


Figure 5.12. Isosurface rendering of (a) E1U at time steps 2 and 13, (b) E5W and E1W at time step 2, and (c) E5W and E1W at time step 13.

*E5W appear to be similar at the beginning, but tend to be more distinguishable over time*. Although the development trend is different, the expert found that *E5W and E1W become more similar in the larger value range*, since the upper-right corners of the E1W-E5W TSMs gradually transit from green to light green. At time step 2, the isosurfaces corresponding to a larger isovalue of E5W (blue) and E1W (orange) occupy different spatial regions, as shown in Figure 5.12 (b); but at time step 13, the two isosurfaces reside in the same region, as shown in Figure 5.12 (c). However, the similarity is still not high, since the distance between different layers of the blue surfaces is much larger.

### 5.3.3 Additional Case Studies

**Climate data set.** This data set was generated from a simulation of salinity (SAL) and temperature (TEM) in the equatorial region from 20°S to 20°N for a period of 100 years. This data set contains 1200 time steps (one month per time step), from which temporal clustering identifies 13 representatives, as shown in Figure 5.13. From the representatives, we can see that SAL exhibits a more stable pattern than TEM. Interestingly, although our temporal clustering does not consider the order of time steps, nine months in the first year are selected as representatives. Overall, the clusters are consistent with the monthly weather change in each year. But we can also notice that for two long periods of time, most

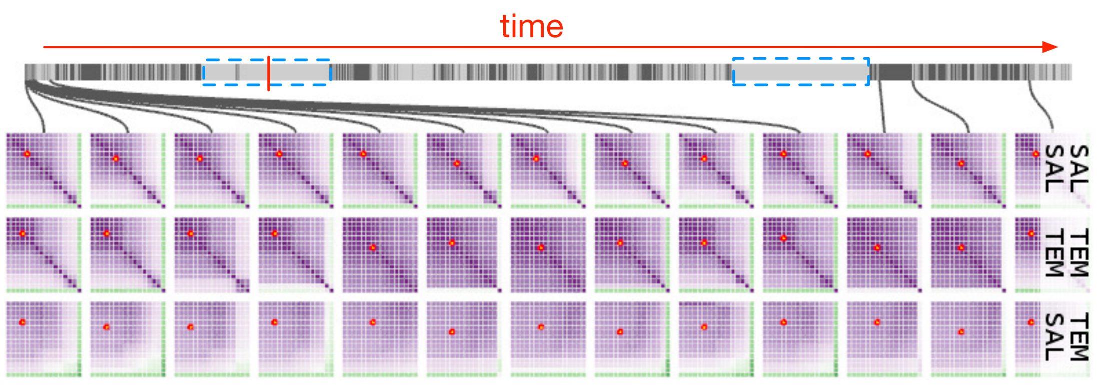


Figure 5.13. Overview of the entire Climate data set of 1200 time steps.

of the time steps are placed in the same cluster, as highlighted in the blue rectangles.

Then, we examine the temperature change with the El Niño condition. We select time step 289 (with El Niño), as highlighted by the red bar in Figure 5.13, and compare it with time step 265 (without). Time step 265 corresponds to the same month as time step 289, but two years before. The single-pair mode of the matrix view displays the two SSMs on the left and the corresponding TSM on the right, as shown in Figure 5.14 (a). The arrows indicate TEM from low to high. We find that *both SSMs share the same pattern*: the low TEM isosurfaces are very similar to each other, indicated by a dark purple block; the medium TEM isosurfaces are moderately similar to each other and the low TEM ones but not that similar to the high TEM ones; and the high TEM isosurfaces are similar to each other but not that similar to the low and medium TEM ones. Similar patterns can be observed in the TSM as well, except that *the high TEM isosurfaces do not share high similarity*.

By selecting the cell highlighted in the blue circle, the isosurface view renders a high TEM isosurface (blue) and a low TEM one (orange) at time step 265 for comparison, as shown in Figure 5.14 (b). The high TEM isosurface is associated with a small region close to the ocean surface and the low TEM one covers the entire oceans at the bottom.

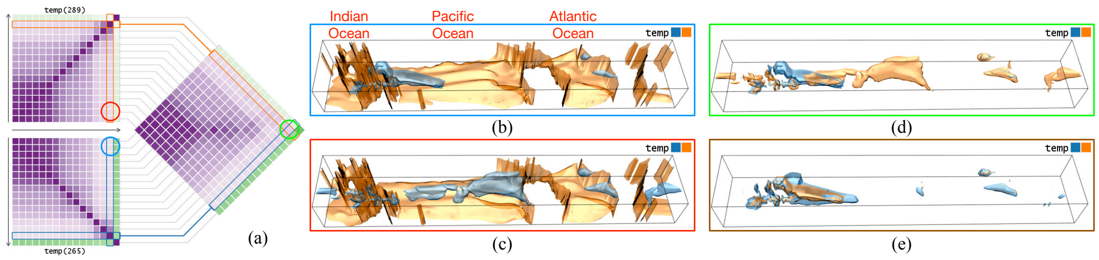


Figure 5.14. The single-pair mode of the matrix view using the Climate data set. (a) shows the TSM comparing time steps 265 and 289. Time step 289 is associated with the El Niño condition. (b) and (c) show a high TEM isosurface (blue) and a low TEM one (orange) at time steps 265 and 289, respectively. (d) shows two high TEM isosurfaces at time steps 265 (blue) and 289 (orange). (e) shows two high TEM isosurfaces at time steps 241 (blue) and 265 (orange).

By selecting the cell highlighted in the red circle, two isosurfaces at time step 289 are displayed, as shown in Figure 5.14 (c). We can see that *the low TEM isosurface is mostly the same as the corresponding one at time step 265; but the high TEM isosurface is quite different, with the central and eastern equatorial Pacific Ocean covered, confirming the existence of El Niño*. By selecting the cell highlighted in the green circle, we compare the two high TEM isosurfaces, as shown in Figure 5.14 (d). The cells related to these two isosurfaces are highlighted by rectangles in the corresponding colors, as shown in Figure 5.14 (a). Figure 5.14 (e) shows two high TEM isosurfaces at time steps 265 and 241 (which is exactly two years before 265). Clearly, *without the El Niño condition, the high TEM regions are similar in the same month across years*.

**Ionization data set.** For this data set, we find that all SSMs show a similar pattern: the purple cells mostly concentrate along the diagonals leaving other regions in green. This indicates *the strongly localized pattern for isosurfaces at different value ranges*. We use map-level paths to trace the movement of the Ionization front for each individual variable and show the corresponding isosurfaces in Figure 5.6. Figure 5.6 (a) to (c) correspond to H2 at three representative time steps at the beginning, middle, and ending stages, respectively. Obviously, *the most representative isosurface shown in blue moves from left to right and demonstrates different shape characteristics*. In Figure 5.6 (a), we can also observe three other representative isosurfaces with a similar shape as the most representative one, displayed as the orange, red, and purple silhouettes. However, in Figure 5.6 (b), the other representative isosurfaces are occluded by the most representative one shown in blue. Therefore, we shift the focus to the orange one, as shown in Figure 5.6 (d). The blue, green, and purple silhouettes indicate that *the isosurfaces form multiple layers of the same structure*. Figure 5.6 (c) shows the isosurfaces of H2 at the ending stage. Figure 5.6 (e) to (h) show a similar process for H+ but with a different shape of the Ionization front.

## 5.4 Conclusions

To provide a convenient mechanism for users to browse through a large time-varying multivariate or ensemble data set in the form of isosurfaces, we present MISM, a visual interface that organizes a huge number of ISMs for navigation and exploration. We design effective solutions to achieve both *computational scalability* (by computing a massive number of self, temporal, and variable similarity maps using GPU-accelerated approximation) and *visualization scalability* (by presenting MISM at different levels of detail via clustering, grouping, and filtering schemes). To the best of our knowledge, both scalabilities have not been demonstrated previously in this context. With advanced features such as path recommendation and surface comparison, MISM is the first of its kind that supports flexible relationship exploration and examination among isosurfaces extracted from the multifaceted data, providing the capability that goes beyond a traditional animation playback could offer.

CHAPTER 6

# CONTOURNET: SALIENT LOCAL CONTOUR IDENTIFICATION FOR BLOB DETECTION IN FUSION PLASMA SIMULATION DATA

In this chapter, I present a deep-learning framework, called ContourNet, for blob detection in XGC plasma fusion simulation data. Blobs are areas of interests close to the boundary of the reactor which can cause damage to the outer wall. Usually, they can be roughly characterized as nearly elliptical and containing a higher than average energy. However, there is no clear definition for blobs and thus they can only be identified heuristically or by domain experts. Based on the discussion with scientists, I transform the problem of *blob detection* to *salient local isocontour detection*. Aiming to detect blobs in plasma fusion, our supervised learning solution is based on expert labels, a label propagation strategy, and a deep CNN.[1]

## 6.1 Background

Formally, the input of blob detection is the scalar field $f_{t,s} : \mathbb{D} \to \mathbb{R}$ on the $s$th 2D cross-section of the tokamak at time step $t$, where $\mathbb{D} \subset \mathbb{R}^2$ is the domain for a 2D cross-section, and the output is a number of regions $\{B^i_{t,s}\}$ that represent blobs. Depending on the context of different experimental and simulation studies, the input scalar field can be temperature, electron densities, scalar derivative of electrostatic potential, and normalized fluctuating density that characterize the fluctuating level. Without loss of generality, we only study

[1]This work was published as M. Imre, J. Han, J. Dominski, M. Churchill, R. Kube, C.-S. Chang, T. Peterka, H. Guo, and C. Wang. ContourNet: Salient local contour identification for blob detection in plasma fusion simulation data. In *Proceedings of International Symposium on Visual Computing*, pages 289–301, 2019

blobs that are associated with local maxima. Assuming the smoothness of the input scalar field $F_{t,p}(\mathbf{x})$, we define

$$L(f_{t,s},\theta) = \{\mathbf{x} \mid F_{t,s}(\mathbf{x}) \geq \theta\} = \bigcup_{k\in\mathbb{I}} R^k_{t,s}(\theta) \tag{6.1}$$

as the *super-levelset* of $f_{t,s}(\mathbf{x})$ with respect to the threshold $\theta$; $\{R^k_{t,s}(\theta)\}$ are disjoint connected components ($R^k_{t,s}(\theta) \cap R^l_{t,s}(\theta) = \emptyset$, for any $k \neq l \in \mathbb{I}$) of the super-levelset and $\mathbb{I}$ is the index set. We further define a *blob candidate*

$$C(f_{t,s},\theta,\mathbf{x}) = \{R^k_{t,s} \mid \mathbf{x} \in R^k_{t,s}\} \tag{6.2}$$

as the simply connected component that contains $\mathbf{x}$ in the super-levelset $L(f_{t,s},\theta)$. The blobs $\{B^j_{t,s}\}$ are a subset of blob candidates that meet the empirical rules defined by scientists, where $j$ is the index of the blob.

In practice, scientists define empirical rules to filter the candidates that connected to a few selected local maxima locations $\{\mathbf{m}^j_{t,s}\}$ ($\mathbf{m}^j_{t,s} \in \mathbb{D}$) with different super-levelset thresholds $\{\theta^j_{t,s}\}$, that is,

$$B^j_{t,s} = C(N,\theta^j_{t,s},\mathbf{m}^j_{t,s}). \tag{6.3}$$

In Kube et al. [127], the threshold $\theta^j_{t,s}$ was set to 60% of the selected maximum value $f_{t,s}(\mathbf{x}_j)$. Davis et al. [47] first found large local maxima in $f_{t,s}(\mathbf{x})$ by thresholding, and then fitted an ellipse to the contour at the 50% maximum level. The same technique was used by Zweben et al. [239] and Churchill et al. [42], but the selection of local maxima is nontrivial and subject to small perturbations of $f_{t,s}(\mathbf{x})$. Based on the statistics of $f_{t,s}(\mathbf{x})$ in a small region of interest, Wu et al. [225] determined the local contour level to only incorporate regions with significantly higher temperatures and densities than the surroundings. A two-phase algorithm was proposed that first selects candidate points and then extracts blobs as the connected components of the candidates.

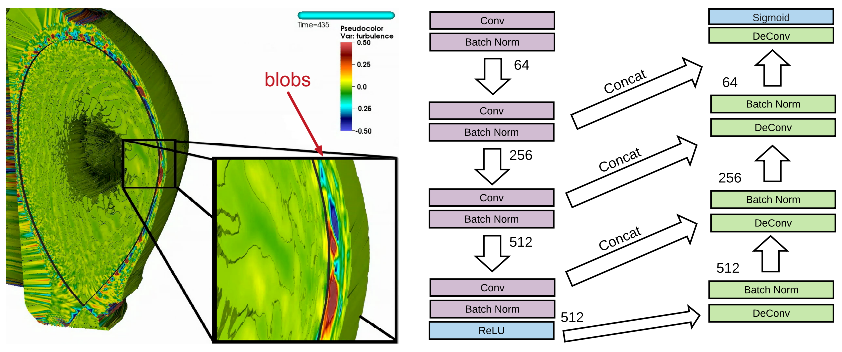
We follow the convention to localize blobs by contouring. Although it is possible to approach blob detection as an object detection problem by segmenting a 2D cross-section into blob and non-blob areas, a local contour has clearer physical meaning than an arbitrary segmentation does. In general, the output segmentation is not necessarily local isocontours and thus could be misleading. Therefore, we aim to segment the image into areas that can be encircled with local isocontours.

For ease of description in terms of image input data, we define $v_{t,s}(i)$ as the $W \times H$ regular 2D image that represents $f_{t,s}(\mathbf{x})$, where $0 \leq i < W \times H$ is the the pixel index. The blob candidate $c_{t,s}(i)$ is defined as the binary (0 or 1) image of the connected component $C(f_{t,s}, f_{t,s}(\mathbf{x}_i), \mathbf{x}_i)$ that contains $\mathbf{x}_i$, the physical coordinates of $i$. We also define the Boolean function $b_{t,s}(i)$ to denote if the blob candidate $c_{t,s}(i)$ is a blob. The purpose of ContourNet is to learn *blob labels* $\widetilde{b}_{t,s}(i)$, in order to predict $b_{t,s}(i)$ for arbitrary $t$, $s$, and $i$, as detailed next.

## 6.2 Workflow and Architecture

**Blob detection workflow.** The input data of our blob detection workflow is the scalar field $f_{t,s}(\mathbf{x})$, and the outputs are a number of detected blobs $\{B^j_{t,s}\}$. The core of our approach is ContourNet, a deep neural network that takes a single slice as input and returns a segmentation for blob and non-blob regions. In this study, ContourNet is based on a modified U-Net [173], which is a CNN originally designed for medical image segmentation. For the ease of data handling using ContourNet, we transform the scalar field $f_{t,s}(\mathbf{x})$ into an image and the ground truth blob candidates into a binary mask.

We tailor both the training and inference routines of ContourNet, in order to detect blobs accurately and efficiently. At the training stage, we gather as many labeled blobs as possible through expert labeling and label propagation. We worked closely with scientists on the user interface design for expert labeling, and co-designed the automatic label propagation strategy, in order to minimize the burden of manual labeling and to maximize the



(a) (b)

Figure 6.1. (a) shows a zoomed view of a blob. The dark red area corresponds to the blob. (b) shows the architecture of ContourNet. Similar to U-Net, ContourNet has two phases. However, there is no change of the image size. The numbers in the figure show the number of channels after a given batch normalization.

blob detection accuracy.

**ContourNet architecture.** The original U-Net is designed for image segmentation: the input is a fixed-size image and the output is a mask that segments the different areas of the image. U-Net contains a downward phase and an upward phase. At each level of the downward phase, a convolutional layer halves the size of the image and doubles the size of the feature maps. Within each level of the upward phase, the exact opposite happens by applying deconvolution operations. Additionally, the features in a given level of the downward phase are copied over as additional input into the same level of the upward phase.

The modified U-Net design for ContourNet is illustrated in Figure 6.1 (b). ContourNet consists of four levels in both the downward and upward phases. The biggest difference to the original U-Net architecture is that ContourNet does not perform any downscaling of the input. We do this as blobs are small-scale features in comparison to the image, which could get missed in an early layer and not be recovered later. The input to each (de-)convolutional layer varies depending on its position in the network. The *i*th level has

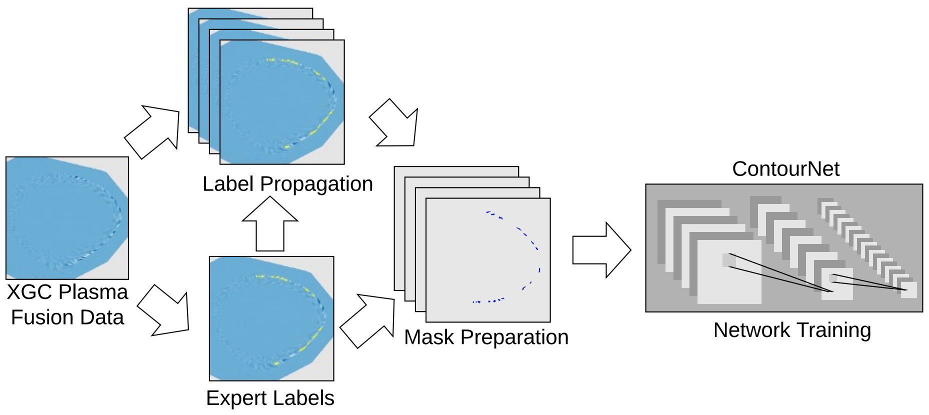


Figure 6.2. The training stage for ContourNet. Expert labels are first propagated to increase the amount of training data. The data slice is then combined with a blob mask as input for network training.

the same size as the input image but with a different number of features in the feature map. To copy over the features from the downward to the upward phase, we concatenate them to the already existing feature map.

The last layer produces a binary segmentation mask that we then perform the following to further transform into a mask of isocontours. First, we compute the connected components for every blob in a segmentation mask. Second, we generate a super-levelset for every point of a given connected component. Third, we compare the given super-levelset to the area from the segmentation using the Dice coefficient and pick the best fit. Finally, we extract the isocontour from the super-levelset and store the seed point for future use.

## 6.3 ContourNet Training

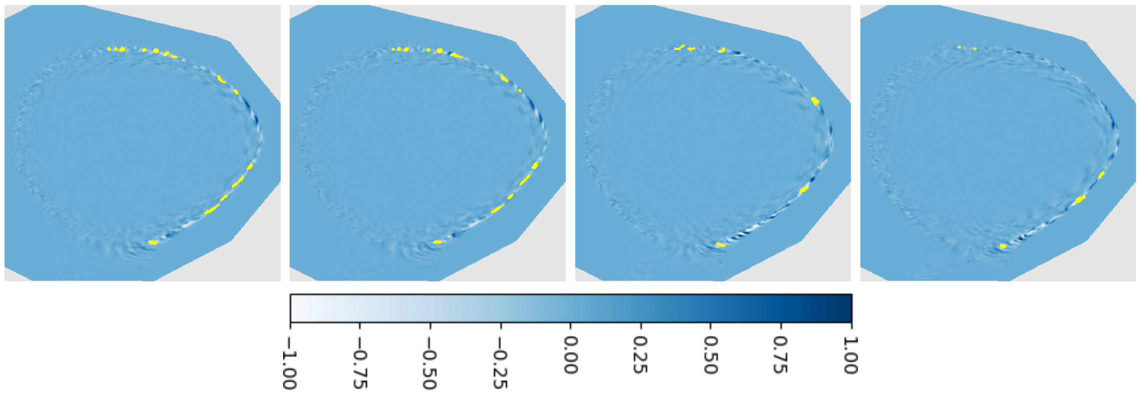
The training stage of ContourNet is shown in Figure 6.2. During training, we preprocess the simulation data into input image $v_{t,s}(i)$ and obtain labeled samples $\widetilde{b}_{t,s}(i)$ from scientists. We further improve the accuracy of ContourNet by introducing propagated labels.

**Data preprocessing.** Because the input image $v_{t,s}(i)$ is not directly available from the simulation, we need to preprocess the simulation data. Based on the instructions from scientists, we first derive $f_{t,s}(\mathbf{x})$ by normalizing the electrostatic potential field with the plasma temperature. The field data, which are defined on a 2D unstructured mesh, are stored in multiple HDF5 files. Because the current CNN implementations do not inherently support unstructured mesh data, we interpolate and convert the data into a $400 \times 400$ regular mesh, which is fine enough to capture blobs for this study. During mask preparation, we collect all the labeled blobs for a time step and slice combination and generate another regular $400 \times 400$ grid with blob areas set to 1 and the rest to 0.

**Label propagation.** We propagate the blob labels $\widetilde{b}_{t,s}(i)$ to a few neighboring time steps to increase the amount of training data and ease the amount of time that scientists need to spend on labeling. Although the propagation is an approximation and cannot be used to detect newly appeared or disappeared blobs, based on the verification from scientists, the propagated blobs are acceptable for the training purpose. A set of propagated labels can be seen in Figure 6.3. In each image, we show blobs as filled regions rather than contours as we used a region-based coefficient to select the best matches. In the figure, (a) shows the expert labels at time step 60, (b) is the propagation to time step 59 which shows very good quality. The propagation to time step 65 is shown in (c). Here we can already see the quality deteriorating as fewer blobs are propagated, but the output is still acceptable. In (d), we can see that the propagation to time step 70 showing even fewer blobs, resulting in a misleading training sample.

Formally, we propagate the labels $\widetilde{b}_{t,s}(i)$ to the next $n$ time steps $\{t+1,\ldots,t+n\}$. The propagation to preceding time steps can be done similarly. Based on the verification from scientists, we set $n = 5$ to achieve a balance between the propagation precision and the number of propagated labels. With this setting, the propagation enables us to generate ten times the labeled blobs that we obtained from the scientists.

The propagation is an iterative process over the consequent time steps. In the $k$th



(a) $(t,s) = (60,10)$ (b) $(t,s) = (59,10)$ (c) $(t,s) = (65,10)$ (d) $(t,s) = (70,10)$

Figure 6.3. Four different time steps showing the same slice with the expert-labeled (a) or propagated (b-d) blobs. $(t,s)$ are for time step ID and slice ID, respectively. We use the underlying density values $\rho_{t,s}$ as the background and overlay them with the blobs in yellow. The density values are clamped to $[-1,1]$. The background is light gray. The images use a $400 \times 400$ pixel resolution.

iteration, we find all blob candidates $c_{t+k,s}(i)$ that are already labeled as blobs, that is, $\widetilde{b}_{t+k,s}(i) = 1$ for all $i$. We then compare each $c_{t+k,s}(i)$ with all possible blob candidates $c_{t+k+1,s}(i')$ using the Dice coefficient [51]. The Dice coefficient for two binary images is defined as

$$\operatorname{dice}(I_1, I_2) = \frac{2 \times |V(I_1) \cap V(I_2)|}{|V(I_1)| + |V(I_2)|}, \tag{6.4}$$

where $I_1$ and $I_2$ are the two blob candidates being compared, and $V(\cdot)$ is the number of nonzero elements in the image. We then find the blob candidate $c_{t+k+1,s}(i')$ with the max Dice coefficient

$$\arg\max_{i'} \operatorname{dice}\left(c_{t+k,s}(i), c_{t+k+1,s}(i')\right), \tag{6.5}$$

and label $\widetilde{b}_{t+k+1,s}(i')$ as 1. If there is no overlap found, we disregard all candidates.

To generate possible blob candidates, we exhaustively sample the area close to the outer wall at a given slice $s$ at time step $t$. To do so, we first filter the ring-like structure of the cross-section. Starting from the remaining area, we inspect every pixel and use it as a seed point to compute an isocontour with its corresponding value and fill the area for propagation.

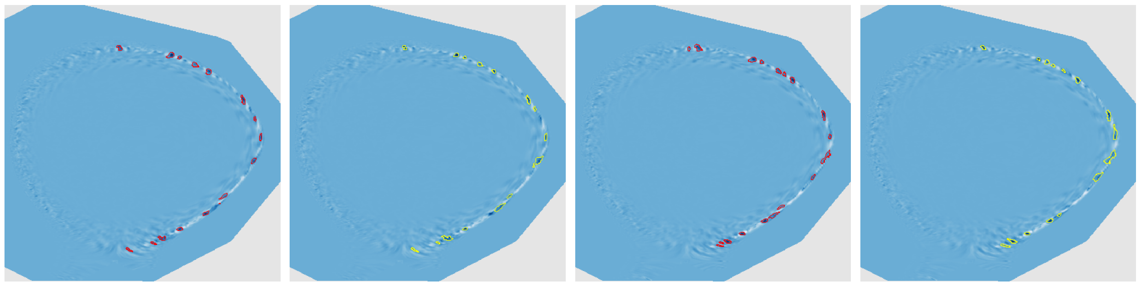
**Training process.** We use all labels $\widetilde{b}_{t,s}(i)$ including both expert labels and propagated labels to train ContourNet. Domain scientists labeled every slice for time step 60, yielding a total of 225 labeled blobs for 15 slices. We extend this set with our label propagation strategy to obtain a total of 165 slices label with about 15 blobs for each slice. We then train ContourNet for 100 epochs on these sets of training data. One epoch took about 52 seconds on a server with an Intel Core i7-7700K 4.2GHz quad-core processor, 32GB main memory, and an NVIDIA Titan Xp GPU. Every input slice comes with a ground-truth mask which is used to compare the weighted cross-entropy loss defined as follows

$$L = -\left(\delta B \log(p) + \gamma(1-B)\log(1-p)\right), \tag{6.6}$$

where $B$ is the ground truth segmentation mask, $p$ is the predicted likelihood for each pixel to be part of a blob, and $\delta$ and $\gamma$ are, respectively, the weighting factors for the blob and non-blob classes. For the results shown in Chapter 6.4, we set $\delta$ to 5 and $\gamma$ to 1.

## 6.4 Results and Discussion

We used 90% of our data for training and 10% for inference. Among the testing set, the prediction yields a Dice score of 0.628. A commonly acceptable Dice score for segmentation tasks is $\geq 0.7$. However, in our scenario, we lower the requirement to a score of $\geq 0.6$ for two reasons. First, we compare inferred results to heuristically generated and noisy "ground truth" (propagated results). Second, small-scale blobs are very difficult to segment and jointly, they only account for a fraction of the image. The average prediction time was 7.56 seconds (segmentation 0.52s, contour extraction 7.04s). In the following,



(a) $(t,s) = (60,6)$ (b) $(t,s) = (60,6)$ (c) $(t,s) = (60,7)$ (d) $(t,s) = (60,7)$

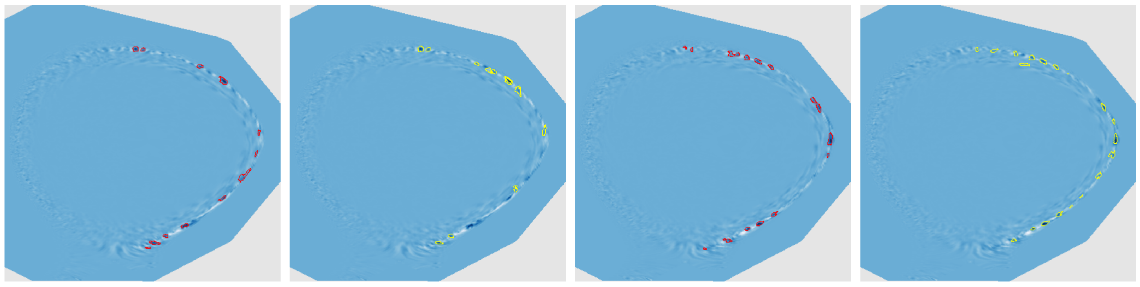
Figure 6.4. Comparing ContourNet's segmentation results to the ground truth. (a) and (c) show the predictions (red), whereas (b) and (d) show the ground truth (yellow).

we discuss selective results in detail.

**Comparison to the ground truth.** Figure 6.4 shows the comparison of our prediction and the ground truth at time step 60. Note that we show the images side by side to avoid the overlap of contours. In (a), we can see that the overall prediction for slice 6 is very good. While some of the blobs differ in the shape or position from the ones shown in (b), the differences are relatively low. In (c), we can see that ContourNet detects more blobs than shown in the ground truth for slice 7. Besides that, some of the blobs have different shapes (bottom right) and there are a few missing ones (top center and right). These results are mostly false positives. We point out that the goal of ContourNet is to assist scientists in identifying blobs, and therefore, false positives are better than a false negatives.

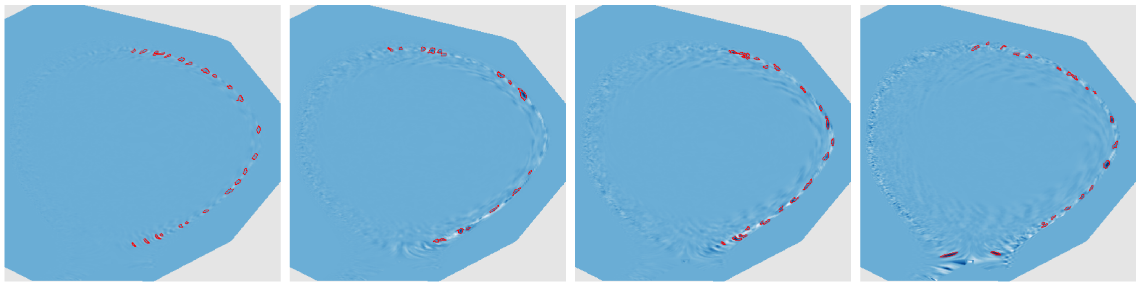
Figure 6.5 shows a comparison of our prediction and the propagated labels at time step 56. (a) and (b) show slice 8 with ContourNet predicting some false positives (center right, bottom). However, the shapes of the detected blobs seem fairly accurate. In (c) and (d), we can see that the prediction for slice 12 is more accurate but misses some of the blobs (top center, right middle).

**Expert evaluation.** We conduct an expert evaluation of ContourNet with fusion scientists. Their feedback is as follows. All in all, ContourNet correctly identifies relevant



(a) $(t,s) = (56,8)$ (b) $(t,s) = (56,8)$ (c) $(t,s) = (56,12)$ (d) $(t,s) = (56,12)$

Figure 6.5. Comparing ContourNet's segmentation results to the propagated ground truth. (a) and (c) show the predictions (red), whereas (b) and (d) show the ground truth (yellow).



(a) $(t,s) = (30,7)$ (b) $(t,s) = (55,9)$ (c) $(t,s) = (70,10)$ (d) $(t,s) = (170,6)$

Figure 6.6. ContourNet segmentation results where contours are highlighted in red.

potential blob candidates across a large spectrum of different simulation data. Figure 6.6 shows toroidal slices of the simulation data at four different time steps. These instances show a variety of situations that typically occur in simulations. In (a), a large number of potential blobs can be seen at the magnetic separatrix. ContourNet identifies the relevant large-amplitude regions in the data and visually, no other region would be a blob candidate. The situation in (b) is more difficult. Again, ContourNet correctly identifies the relevant large-amplitude regions of the image. Note that these regions appear more stretched out along the poloidal direction than in (a). On the other hand, the negative regions in the outboard mid-plane region might be relevant for the physics of the system. In (c), perturbations of the electric potential can be seen all along the separatrix. Here ContourNet identifies contour regions only on the rightmost half, the region where blob dynamics are most relevant to plasma confinement. This inference result is significantly better than the corresponding propagated result shown in Figure 6.3 (d). The simulation data shown in (d) are a difficult test case. Again, small-scale perturbations of the potential appear all along the magnetic separatrix. These stretch out over an X-shaped region at the bottom of the simulation domain. ContourNet identifies multiple blob instances, two of them in the X-shaped bottom region. This area is relevant for the physics underlying blob motion. There are no regions identified in the left half of the image, a region that is not very relevant for blob motion.

## 6.5 Conclusions

We have presented ContourNet, a deep CNN to detect blobs as local isocontours in the XGC plasma fusion simulation. ContourNet learns from expert labeled data and profits from a label propagation strategy that allows us to heuristically label neighboring time steps from the ones labeled by experts. We further present expert evaluation agreeing with the performance of ContourNet. Our work provides a baseline for future improvement that uses only minimal labeling effort.

# CHAPTER 7

# A STUDY OF DEEP LEARNING APPROACHES FOR UNSTEADY FLOW FIELD RECONSTRUCTION

In this chapter, I conduct a study of deep learning frameworks to synthesize vector fields of missing time steps in unsteady VFD. Large-scale unsteady flow simulations can often afford to output the vector fields only at a coarse temporal granularity. Augmenting the temporal resolution during post-processing will lead to higher quality analysis and visualization of the underlying flow field data. In this study, I train three different variants of a deep CNN based on the vector fields of two non-adjacent time steps. During the reconstruction, the network only uses a pair of vector fields as input and synthesizes the missing ones in between the input pair.[1]

## 7.1 Study Description

Flow visualization is a core area of research in scientific visualization. While there are many methods developed for flow visualization, the ones based on integration is the most popular one for three-dimensional VFD. For integration-based methods, seeds are placed in the vector field and traced as *streamlines* (steady flow) and *pathlines* (unsteady flow) for visualization. In this chapter, I focus on unsteady vector fields.

Scientists performing large-scale flow simulations usually run into the problem of having only limited storage possibilities, and therefore, they only save the output sparsely.

[1]This study is part of an ongoing project.

This is especially challenging due to the storage and data movement limitations in high-performance computing systems, which allows scientists to store only the most relevant data for post hoc analysis. As this form of data reduction is an inevitable threat to analysis, data reconstruction and restoration becomes necessary to enable detailed insight into the data. In our scenario, we consider unsteady vector fields, typically stored sparely (e.g., every fifth or tenth time step) during simulation time. Our goal is to restore unsaved intermediate time steps to accurately reconstruct missing output in a post-processing step.

This task poses several challenges. First, the *uvw* channels of a vector field, unlike the *rgb* channels of an image, could have dramatically different ranges. For instance, $u \in [-5,3]$, $v \in [-0.03, 0.05]$, and $w \in [-10^{-4}, 10^{-2}]$. Treating these channels equally using a CNN, RNN, or GAN would not work because components with a larger range will eliminate the smaller ones during convolutional operations.

Second, the change over time in VFD does usually not exhibit linear behavior. Thus linear interpolation (LERP) would not lead to desirable results because it interpolates the vectors based on *local* information rather than considering the *global* evolution and nonlinear changes.

Third, VFD need to be accurate in both *magnitude* and *angle*. Only using one of those measures during optimization could lead to inaccurate flow behavior. To reconstruct missing VFD, we should account for both measures.

To tackle these challenges, we propose *channel CNN* (CCNN), an approach for unsteady VFD reconstruction using a deep neural network. The goal of this study is to let CCNN learn the nonlinear behavior along both the *spatial* and *temporal* dimension to generate accurate intermediate vector fields. Doing so, we employ a combination of losses, considering both vector magnitude and angle. Furthermore, we split the three components into three different networks to solve the value-range problem. Given two vector fields for interpolation, CCNN can synthesize the missing intermediate vector fields after the training converges. We perform quantitative and qualitative evaluation of different variants of

CCNN on data sets with different characteristics.

The main contributions of this study are as follows. First, this is the first attempt that uses a deep neural network for unsteady vector field reconstruction. Second, we combine both magnitude and angle loss for the optimization of CCNN. Third, we propose a new network architecture separating the input channels, which is untypical for other tasks such as image or volume super-resolution. Fourth, this study investigates several hyperparameter settings and their impact on the performance.

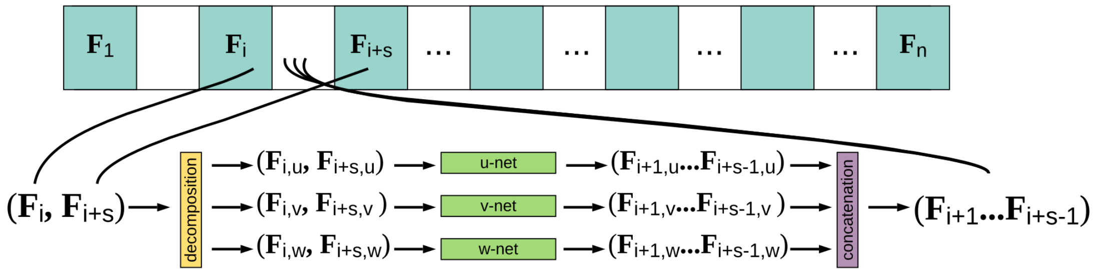


Figure 7.1. Overview of the CCNN framework. A sequence of vector fields is used as input and decomposed into three separate networks. Intermediate vector fields are synthesized.

## 7.2 CCNN

Figure 7.1 shows an overview of our framework. Given a sequence of vector fields $\mathbf{F} = \{\mathbf{F}_1, \cdots, \mathbf{F}_k\}$, we subdivide it into $\mathbf{F}^T = \{\mathbf{F}_1, \cdots, \mathbf{F}_n\}$ as the sequence of vector fields used for *training*, and $\mathbf{F}^I = \{\mathbf{F}_n, \mathbf{F}_{n+s}, \cdots, \mathbf{F}_{n+ms}\}$ as the sequence of vector fields used for *inference*. $s$ is defined as the interpolation interval, which is the granularity at which rate the input data are stored. Each $\mathbf{F}_i$ consists of three different channels $\mathbf{F}_{i,u}$, $\mathbf{F}_{i,v}$, and $\mathbf{F}_{i,w}$.

We denote $L$, $H$, and $W$ as the three spatial dimensions of the vector fields $\mathbf{F}$. For the neural networks ($u$-Net, $v$-Net, and $w$-Net), we denote the learnable parameters as $\theta_U$, $\theta_V$, and $\theta_W$, respectively.

We study an interpolation function $\mathscr{I}$ with the goal of $\mathscr{I}(\mathbf{F}_i, \mathbf{F}_{i+s}) \approx \{\mathbf{F}_{i+1}, \cdots, \mathbf{F}_{i+s-1}\}$, where $i \in [n, n+s, \cdots, n+(m-1)s]$. This means, given the vector fields at both ends of an interpolation interval $s$: $\mathbf{F}_i$ and $\mathbf{F}_{i+s}$, we interpolate $s-1$ intermediate vector fields: $\mathbf{F}_{i+1}$ to $\mathbf{F}_{i+s-1}$.

CCNN accepts $\mathbf{F}_i$ and $\mathbf{F}_{i+s}$ as input and decomposes each vector into its three components: ($\mathbf{F}_{i,u}$, $\mathbf{F}_{i,v}$, and $\mathbf{F}_{i,w}$) and ($\mathbf{F}_{i+s,u}$, $\mathbf{F}_{i+s,v}$, and $\mathbf{F}_{i+s,w}$). Three independent neural networks ($u$-Net, $v$-Net, and $w$-Net) process each of the components to interpolate the intermediate time steps:

$$\{\hat{\mathbf{F}}_{i+1,u}, \cdots, \hat{\mathbf{F}}_{i+s-1,u}\} \approx \mathscr{I}_u(\mathbf{F}_{i,u}, \mathbf{F}_{i+s,u}), \tag{7.1}$$

$$\{\hat{\mathbf{F}}_{i+1,v}, \cdots, \hat{\mathbf{F}}_{i+s-1,v}\} \approx \mathscr{I}_v(\mathbf{F}_{i,v}, \mathbf{F}_{i+s,v}), \tag{7.2}$$

$$\{\hat{\mathbf{F}}_{i+1,w}, \cdots, \hat{\mathbf{F}}_{i+s-1,w}\} \approx \mathscr{I}_w(\mathbf{F}_{i,w}, \mathbf{F}_{i+s,w}). \tag{7.3}$$

Once completed, CCNN concatenates the synthesized components $\{\hat{\mathbf{F}}_{i+1,u}, \cdots, \hat{\mathbf{F}}_{i+s-1,u}\}$, $\{\hat{\mathbf{F}}_{i+1,v}, \cdots, \hat{\mathbf{F}}_{i+s-1,v}\}$, and $\{\hat{\mathbf{F}}_{i+1,w}, \cdots, \hat{\mathbf{F}}_{i+s-1,w}\}$ into $\{\hat{\mathbf{F}}_{i+1}, \cdots, \hat{\mathbf{F}}_{i+s-1}\}$.

The goal is to minimize the difference between the interpolated vector fields $\hat{\mathbf{F}}$ and the ground-truth $\mathbf{F}$. During training, the difference is propagated to $u$-Net, $v$-Net, and $w$-Net to find the best global settings of $\theta_U$, $\theta_V$, and $\theta_W$. Next, we describe the network architecture and loss function of CCNN, followed by the optimization details used during training.

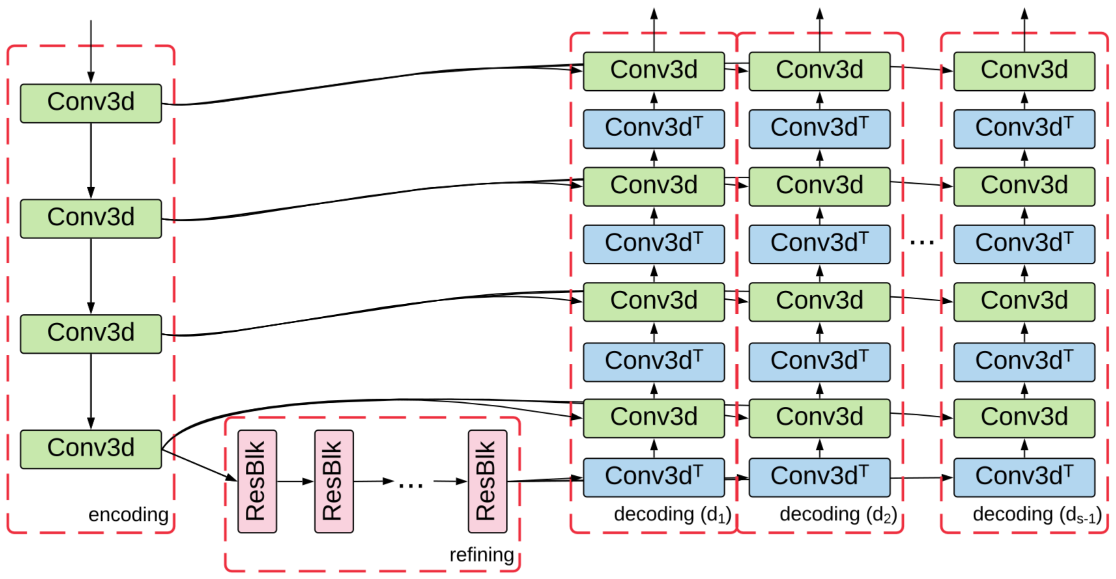


Figure 7.2. Network architecture of CCNN. We omit ReLUs in the figure for brevity. CCNN consists of three paths, an encoding, a refining, and multiple decoding ones.

### 7.2.1 Network Architecture

As shown in Figure 7.1, CCNN uses two sampled vector fields (i.e., $\mathbf{F}_i$ and $\mathbf{F}_{i+s}$) as input and leverages the three separate neural networks to interpolate each of the vectors' components individually to finally produce the intermediate vector fields $\hat{\mathbf{F}}$. We leverage two techniques in CCNN to guarantee the network's performance: (1) we add *residual blocks* (RBs) [92] to circumvent the gradient vanishing problem and (2) we apply *skip connection* [174] to incorporate temporal information to improve the quality of the interpolated vector fields.

RBs connect feature maps directly from earlier layers to later layers, allowing the gradient to be calculated through multiple paths. He et al. [92] has shown to alleviate the gradient vanishing problem during network training. A typical RB consists of multiple paths. One path has several convolutional (Conv) layers that do not change the input dimensions. The other path has only one Conv layer that processes the input. Finally, both paths' outputs are added together. In addition, we use skip connection to incorporate temporal coherence. We do so by allowing information flow from starting and ending vector fields into the generative process for the synthesized vector fields. This allows CCNN to leverage the temporal information rather than only considering deep spatial features.

During this process, more specifically, a feature map is fed into RB to extract and refine higher-level features to be added to the input feature map. Skip connections stack their output feature maps from the encoding stage on top of the input feature maps during the decoding stage at their respective level. A single Conv is used to merge the stacked features into a feature map of the corresponding size.

Figure 7.2 shows the network architecture of each of the three corresponding networks. The network consists of an *encoding* path, a *refining* path, and $s-1$ *decoding* paths ($s$ is the interpolation interval). There are four Conv layers in the encoding path, several RBs in the refining path, and four composites of deconvolutional (DeConv) and Conv layers in the decoding path. Throughout the encoding path, each of the four Conv layers halves the input dimensions. In the refining path, RBs are used to refine the output feature maps from the encoding path. In each decoding path, DeConv layers double the dimensions of the features maps. As previously mentioned, we add skip connection to merge the features maps from the encoding and decoding paths.

After each Conv and DeConv layer, a rectified linear unit (ReLU) [159] is added to help networks learn faster and perform better. Note that there is no activation function after the final Conv layer. Full details of the network's parameters are shown in Table 7.1. We opted to use separated decoding paths, because the dynamic behaviors of unsteady flow could lead to similar flow behaviors in synthesized vector fields from a weight-sharing decoder (i.e., $s-1$ decoders sharing the same weights).

TABLE 7.1

PARAMETER DETAILS FOR THE CCNN ARCHITECTURE.

| type | kernel size | output channels | output size |
|---|---|---|---|
| Input | N/A | 1 | $L \times H \times W$ |
| Conv+ReLU | 4 | 64 | $L/2 \times H/2 \times W/2$ |
| Conv+ReLU | 4 | 128 | $L/4 \times H/4 \times W/4$ |
| Conv+ReLU | 4 | 256 | $L/8 \times H/8 \times W/8$ |
| Conv+ReLU | 4 | 512 | $L/16 \times H/16 \times W/16$ |
| $K\times$RBs | 3 | 512 | $L/16 \times H/16 \times W/16$ |
| DeConv+ReLU | 4 | 256 | $L/8 \times H/8 \times W/8$ |
| Conv+ReLU | 3 | 256 | $L/8 \times H/8 \times W/8$ |
| DeConv+ReLU | 4 | 128 | $L/4 \times H/4 \times W/4$ |
| Conv+ReLU | 3 | 128 | $L/4 \times H/4 \times W/4$ |
| DeConv+ReLU | 4 | 64 | $L/2 \times H/2 \times W/2$ |
| Conv+ReLU | 3 | 64 | $L/2 \times H/2 \times W/2$ |
| DeConv+ReLU | 4 | 32 | $L \times H \times W$ |
| Conv | 3 | 1 | $L \times H \times W$ |

The listed parameters are for a single network of the three $u, v, w$-nets.

### 7.2.2 Loss Function

The synthesized quality of vector fields depends on the speed (magnitude) and direction (angle) of each of their vectors. Specifically, both attributes have a significant influence on streamline and pathline tracing, and errors could accumulate, leading to inaccurate streamline and pathline tracing and rendering results. To optimize CCNN's parameters via backpropagation, the differences in both of these have to be computed between the synthesized intermediate and ground truth vector fields. While the difference in magnitude of a vector can be substituted for $\mathscr{L}_M$, the *mean squared error* (MSE) by omitting the square root operations, the angel difference needs to be computed.

We define the angle loss as

$$\mathscr{L}_A = \frac{1}{k \times L \times H \times W} \sum_{j=1}^{k} \arccos(\mathbf{F}_j, \hat{\mathbf{F}}_j), \tag{7.4}$$

where $k$ is the number of training samples, $\arccos(\cdot)$ is the inverse cosine function, and $\mathbf{F}_j$ and $\hat{\mathbf{F}}_j$ are the ground-truth and synthesized vectors at the $j$th training sample, respectively.

Combining both losses, we define the final loss function as follows

$$\mathscr{L} = \lambda \mathscr{L}_A + (1-\lambda)\mathscr{L}_M, \tag{7.5}$$

where $\lambda \in [0,1]$ is the impact of the angle loss.

### 7.2.3 Optimization

The training process of CCNN is as follows. The training data $\mathbf{F}^T = \{\mathbf{F}_1, \cdots, \mathbf{F}_n\}$ is reorganized as vector field pairs $(\mathbf{F}_1, \mathbf{F}_{s+1}), (\mathbf{F}_2, \mathbf{F}_{s+2}), \cdots, (\mathbf{F}_{n-s}, \mathbf{F}_n)$. With the vector field pairs, $\theta_U$, $\theta_V$, and $\theta_W$, and $\lambda$, we update CCNN iteratively by applying *stochastic gradient descent* (SGD) for a preset number of epochs. During each epoch, CCNN processes all the training sample pairs. Given a vector field pair, CCNN outputs the intermediate vector fields. Using Equation (7.5), the gradients $\bigtriangledown_{\theta_U}\mathscr{L}$, $\bigtriangledown_{\theta_V}\mathscr{L}$, and $\bigtriangledown_{\theta_W}\mathscr{L}$ are computed. Using an optimizer and a predefined learning rate allows us to automatically update the parameters $\theta_U$, $\theta_V$, and $\theta_W$ with the gradients. The inference stage behaves the same, with the exception that gradients are not computed.

TABLE 7.2

UNSTEADY FLOW DATA SETS

| data set | dimension (L×W×H×T) | crop size | $\delta_e$ | training time (s) |
|---|---|---|---|---|
| Square Cylinder | 192×64×48×97 | full | 1.521 | 50.2 |
| Tornado | 128×128×128×43 | full | 2.028 | 167.2 |
| Solar Plume | 128×128×512×25 | 80×80×320 | 3.25 | 63.3 |
| Hurricane | 256×256×56×43 | 224×224×48 | 3.25 | 188.3 |
| Supernova | 256×256×256×121 | 128×128×128 | 3.0 | 105.74 |

The experimented data sets with their dimensions, the used crop size, the entropy value used for entropy-based evaluation and the training time of a single epoch

## 7.3 Preliminary Results

### 7.3.1 Data Sets and Network Training

Table 7.2 shows the simulation data sets we experimented with, including their dimension, crop size, selected entropy threshold $\delta_e$ (see Chapter 7.3.3), and training time, sorted by the volume size of a single vector field. The inference for a given interpolation interval takes between 0.25 and 0.34 second, depending on the size of the data set. From top to bottom, these data sets depict, respectively, the 3D flow around a square cylinder between parallel walls [208], a procedurally generated tornado [46], the compressible downflow of a solar plume [171], a simulation of hurricane Isabel (a strong hurricane in the western Atlantic region in September 2003) made available through IEEE Visualization 2004 Contest, and the flow of core-collapse supernova [16].

We trained three different architectures for our evaluation. CCNN is the network architecture shown in Chapter 7.2.1. LSTM adapts CCNN by adding an ConvLSTM cell between the last RB and the decoders. GAN trains a *general adversarial network*, using CCNN as the generator and a simple CNN (four Conv and LeakyReLU layers) as the dis-

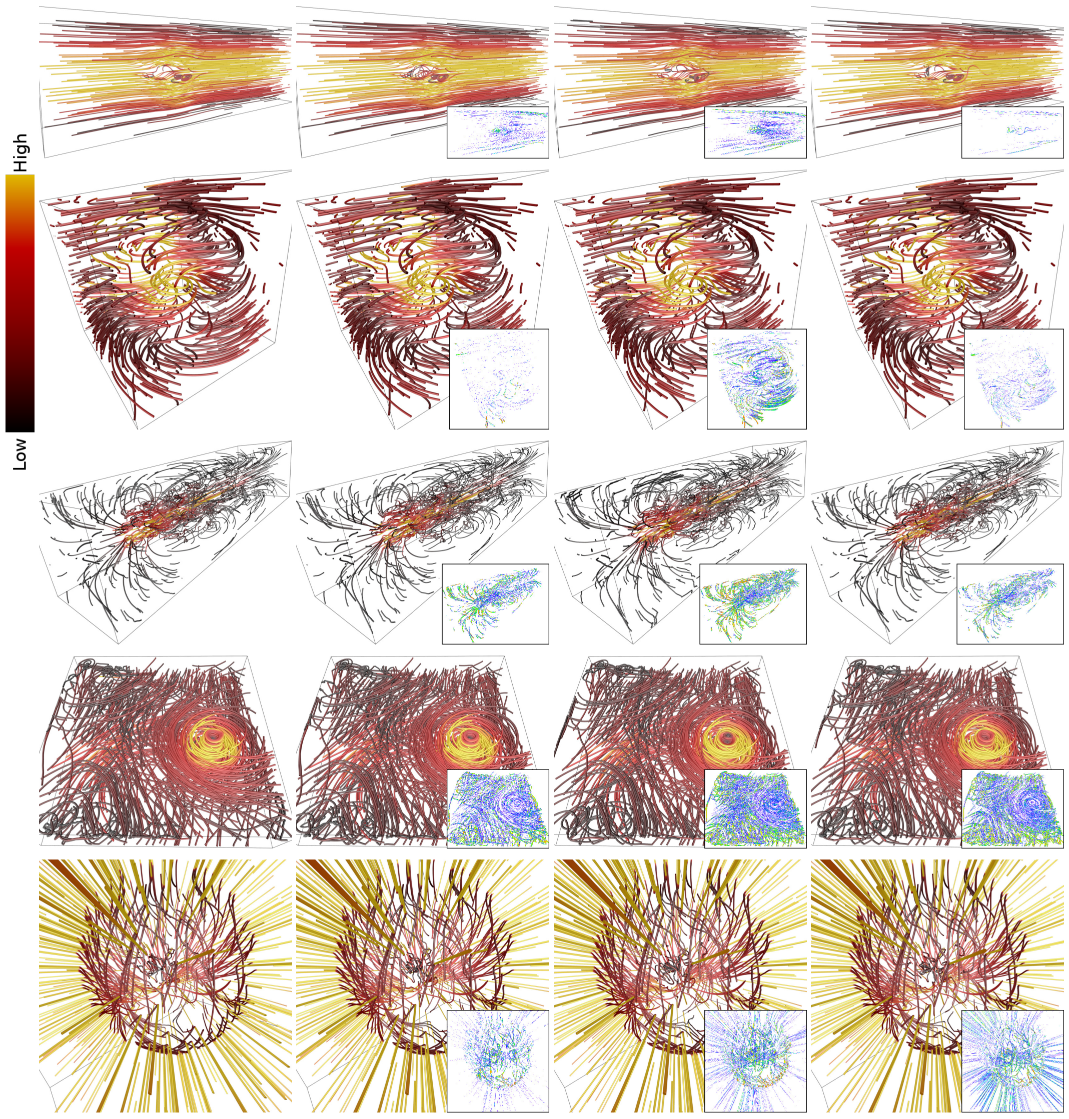


Figure 7.3. Comparison of the traced pathlines. From top to bottom: Square Cylinder, Tornado, Solar Plume, Hurricane, and Supernova. (All variants use the same seeds for pathline tracing. A difference image is shown at the bottom-right corner.

criminator. A single NVIDIA TESLA V100 GPU was used for training and testing except for LSTM, which needed to be split into two NVIDIA TESLA V100 GPUs due to memory limitations. For each epoch, we use a random crop with the size depicted in Table 7.2 from a vector field pair $(\mathbf{F}_i, \mathbf{F}_{i+s})$ due to memory constraints. Note that for the Tornado and Square Cylinder data sets, we use the full vector field size. This cropping mechanism speeds up the training process and allows processing of large vector fields.

For optimization, the parameters are initialized by He et al. [91] and updated through the Adam optimizer [118]. We set one training sample per minibatch. We set the learning rate to $10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for the Adam optimizer. The networks were trained for $1{,}500$ epochs per data set. Note that we stopped training for GAN when the discriminator collapsed.

### 7.3.2 Qualitative Evaluation

For qualitative evaluation, we use different ways to visualize the complete vector field sequences $\mathbf{F}^C = \mathbf{F}^T \cdot \mathbf{F}^I = \{\mathbf{F}_1, \cdots, \mathbf{F}_n, \mathbf{F}_{n+1}, \cdots, \mathbf{F}_{n+ms}\}$ where $\cdot$ denotes concatenation. Let $\mathbf{F}_k$ be $\mathbf{F}_{n+ms}$. First, we trace and render pathlines from $\mathbf{F}_1$ to $\mathbf{F}_k$ (Figure 7.3). Second, we trace and render streamlines in the vector field $\mathbf{F}_{k-\frac{s}{2}}$ (Figure 7.4) as this *should* be the worst possible result for our methods because it is in the middle of the interval furthest away from $\mathbf{F}^T$. Third, we depict the error volume for the vector field $\mathbf{F}_{k-\frac{s}{2}}$ (Figure 7.5) defined as follows

$$\mathscr{E}(\mathbf{F}, \hat{\mathbf{F}}) = \mathscr{E}(\mathbf{F}[i], \hat{\mathbf{F}}[i]), \text{ for each } i \in \mathbf{F}, \tag{7.6}$$

where the voxel error for the $i$th voxel in $\mathbf{F}$ is given by

$$\mathscr{E}(\mathbf{F}[i], \hat{\mathbf{F}}[i]) = \sqrt{\sum_{j \in u,v,w} (\mathbf{F}[i]_j - \hat{\mathbf{F}}[i]_j)^2}, \tag{7.7}$$

and $\hat{\mathbf{F}}$ is the synthesized vector field.

Figure 7.3 shows the traced pathlines for the five data sets. All of the pathlines are traced through the whole sequence ($\mathbf{F}_1$, $\mathbf{F}_k$). The pathlines are colored by velocity magnitude from dark red (low) to yellow (high). For easy comparison, we show the difference image between the ground truth and the corresponding image at the bottom-right corner. The difference image is derived by calculating pixelwise differences (the Euclidean distances) in the CIELUV color space of streamline or pathline rendering images generated from the synthesized and original VFD. We map the noticeable pixel differences (with $\Delta \geq 6.0$) to nonwhite colors (clamping differences larger than 255). For the Square Cylinder data set, we can see that CCNN produces errors mostly in the boundary region, but has a better grasp of the center. Here the LSTM variant seems to perform better, showing fewer errors in the difference image. Similar to this, but not as significant is the difference for the Solar Plume data set. However, the errors spread throughout the whole volume. For the Tornado data set, CCNN performs the best among the three candidates, whereas GAN and LSTM show many errors. For the Hurricane data set, all methods show many errors throughout the whole vector field. However, CCNN has some white region (zero error) closer to the hurricane's eye. This data set depicts the movement of a hurricane. It is quite difficult to reconstruct missing information due to significant changes between time steps. Lastly, the difference images for the Supernova data set shows that all versions have difficulty in capturing the behavior in the boundary region. Overall, we can see that GAN does not perform as well as the other two (CCNN and LSTM).

The next comparison, shown in Figure 7.4, shows the streamlines for all data sets at time step $\mathbf{F}_{k-\frac{s}{2}}$, which is the time step in the middle of the interval furthest away from the training samples. The streamlines are colored by velocity magnitude from blue (low) to red (high). Similar to the pathline images, we show the difference image at the bottom-right corner. The tracing results show very similar quality as those for pathlines. CCNN and LSTM usually perform well at the center of the feature region but struggle at the boundary. For the Tornado data set, we can see that CCNN performs better than LSTM, while the other data sets show very similar quality. Similar to the pathlines comparison,

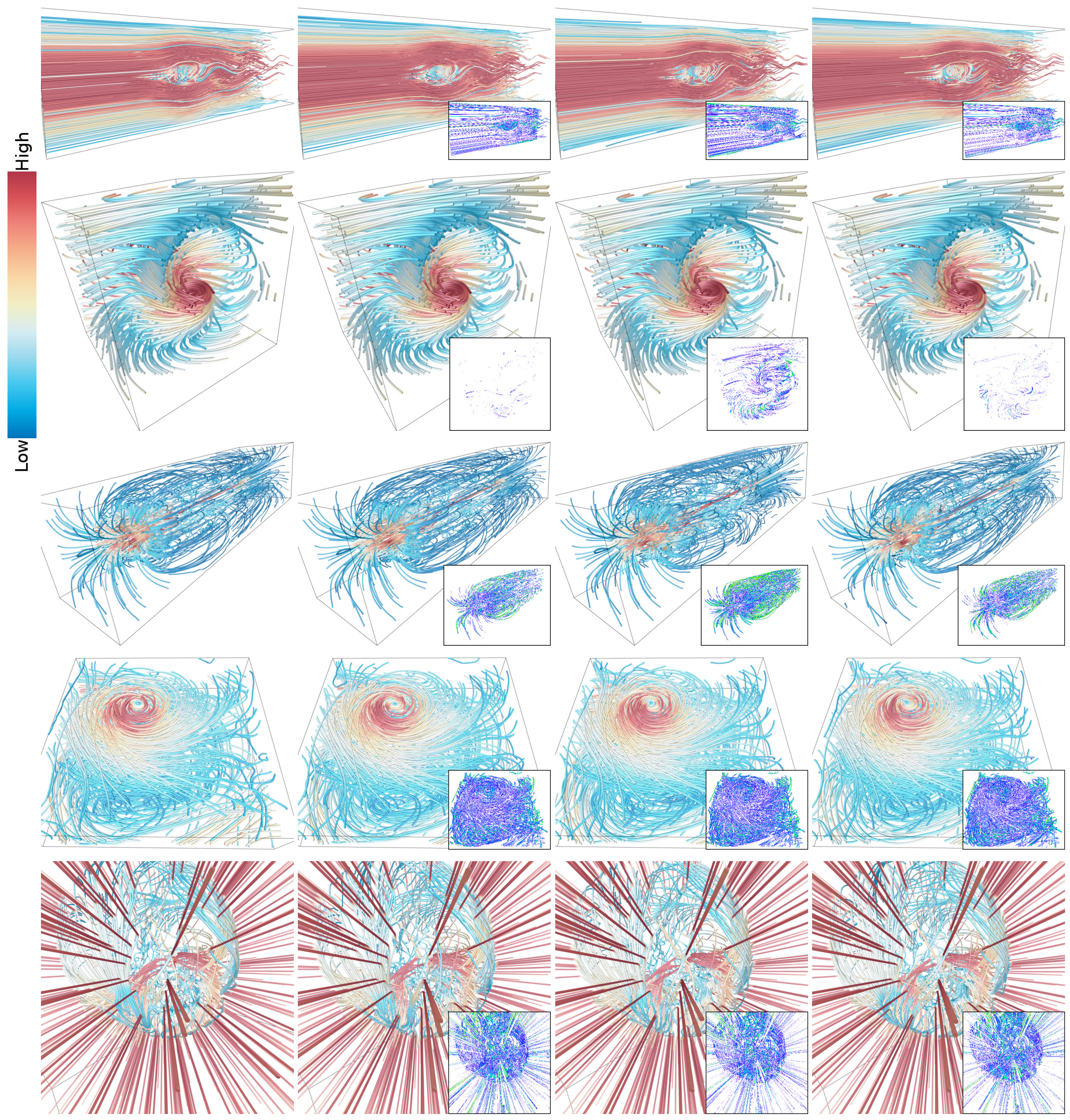


Figure 7.4. Comparison of the traced streamlines. From top to bottom: Square Cylinder, Tornado, Solar Plume, Hurricane, and Supernova. All variants use the same seeds for streamline tracing. A difference image is shown at the bottom-right corner.

GAN performs the worst among the three methods.

Figure 7.5 shows the error volumes between each method and the ground truth for all data sets at time step $\mathbf{F}_{n+ms-\frac{s}{2}}$. The images in the left column show the color map for each data set, respectively, ranging from blue (low) to red (high). For the Square Cylinder and Hurricane data sets, it is immediately visible that there are small errors throughout the whole vector field across all three variants. In the case of the Square Cylinder data set, CCNN only shows a tiny red area at the center, and two small ones along the right border, while GAN and LSTM also have some more high-error regions at the center residing in the feature region. The other case with the Hurricane data set shows that all methods have error regions at the center of the hurricane's eye, and smaller errors to almost no errors further away from it. However, among all three methods, CCNN has the smallest high-error region at the center of the hurricane's eye. For the Tornado data set, we set the lower bound of the error to $2.2 * 10^{-2}$ to avoid having minute errors obscuring the feature region. These tiny errors occur across all methods and vector fields. The center, however, shows that LSTM and CCNN most accurately synthesize the feature region while GAN produces the error in the form of a swirl below the top of the tornado. The error volumes for the Solar Plume data set show that all methods have some errors at the center of the solar flare. GAN has small errors in the boundary region, while LSTM and CCNN show minute errors there. Upon closer inspection of the feature region, one could see that there are almost no red areas in the rendering of the error volume for all methods. This suggests that these methods produce pretty accurate results by keeping the maximum error down. Last, we show the Supernova data set's error volumes, also with a lower boundary to avoid occlusion from non-significant errors in the boundary region. Here we see very similar results for all methods. However, it is worth noting that, compared to other methods, CCNN has fewer but larger high-error regions. This shows that CCNN usually performs very well in the feature region and has small errors spreading throughout the vector field.

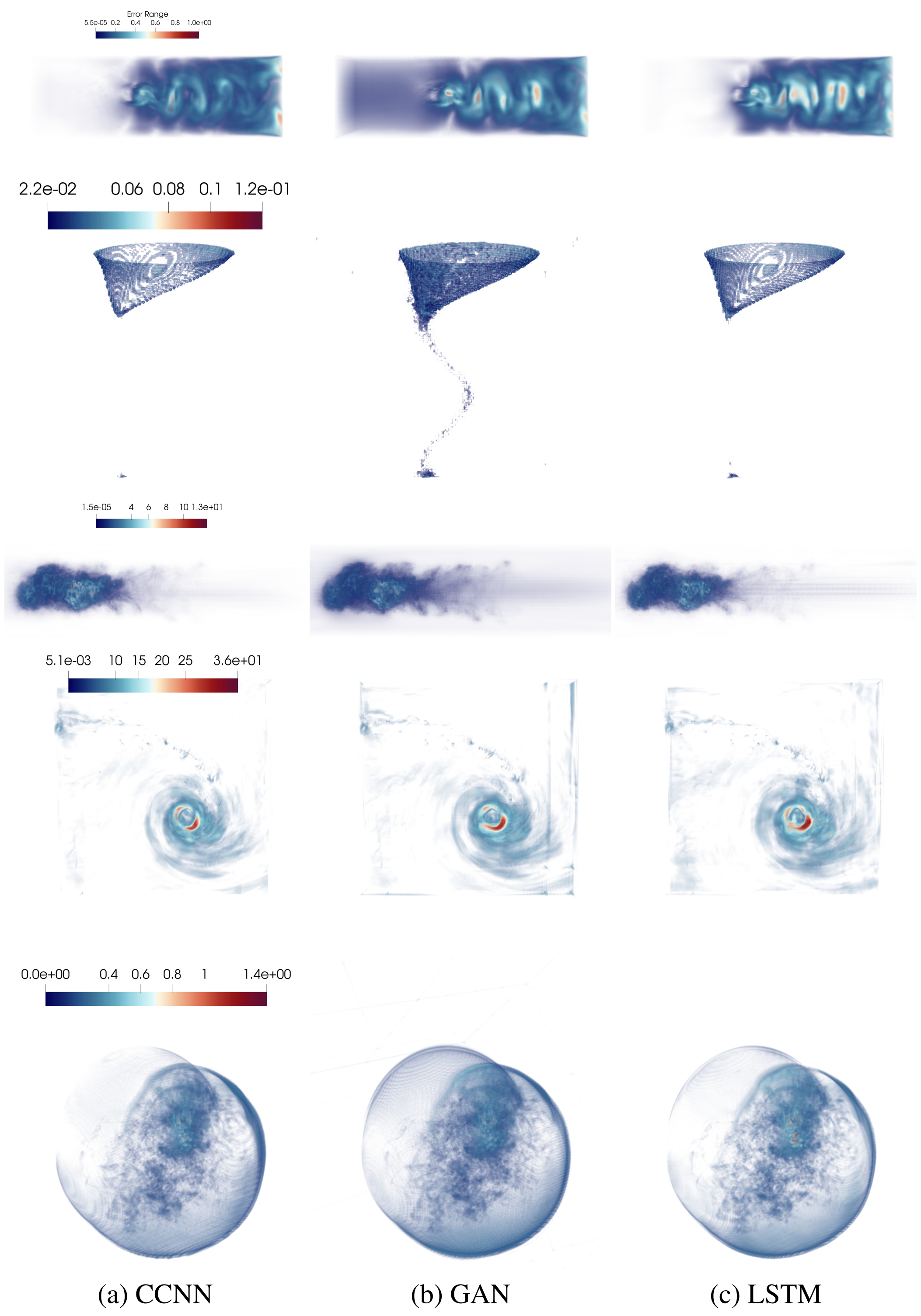

Error Range
5.5e-05 0.2 0.4 0.6 0.8 1.0e+00
2.2e-02 0.06 0.08 0.1 1.2e-01
1.5e-05 4 6 8 10 1.3e+01
5.1e-03 10 15 20 25 3.6e+01
0.0e+00 0.4 0.6 0.8 1 1.4e+00
(a) CCNN
(b) GAN
(c) LSTM

Figure 7.5. Error volumes for $\mathbf{F}_{k-\frac{s}{2}}$. From top to bottom: Square Cylinder, Tornado, Solar Plume, Hurricane, and Supernova.

### 7.3.3 Quantitative Evaluation

For quantitative evaluation, we use the *peak signal-to-noise ratio* (PSNR) and *average angle distance* (AAD) as metrics. PSNR is defined as

$$\mathrm{PSNR}(\mathbf{F},\hat{\mathbf{F}}) = 20\log_{10} I(\mathbf{F}) - 10\log_{10}\mathrm{MSE}(\mathbf{F},\hat{\mathbf{F}}), \tag{7.8}$$

where $\mathbf{F}$ and $\hat{\mathbf{F}}$ are the original an reconstructed vector fields, respectively, $I(\mathbf{F})$ describes the value range of the vector field as

$$I(\mathbf{F}) = \max(\mathbf{F}_u,\mathbf{F}_v,\mathbf{F}_w) - \min(\mathbf{F}_u,\mathbf{F}_v,\mathbf{F}_w), \tag{7.9}$$

where $\mathbf{F}_u$, $\mathbf{F}_v$, and $\mathbf{F}_w$ are the velocities for the components $u$, $v$, and $w$ in $\mathbf{F}$, respectively. $\mathrm{MSE}(\mathbf{F},\hat{\mathbf{F}})$ is the mean squared error between $\mathbf{F}$ and $\hat{\mathbf{F}}$, defined as

$$\mathrm{MSE}(\mathbf{F},\hat{\mathbf{F}}) = \frac{1}{3\times L\times W\times H}\sum_{i=1}^{L\times W\times H}\sum_{j\in u,v,w}(\mathbf{F}[i]_j - \hat{\mathbf{F}}[i]_j)^2, \tag{7.10}$$

where $\mathbf{F}[i]$ and $\hat{\mathbf{F}}[i]$ are the vectors of $\mathbf{F}$ and $\hat{\mathbf{F}}$ at voxel $i$, respectively.

AAD averages the difference of the angles of all the corresponding vectors in both $\mathbf{F}$ and $\hat{\mathbf{F}}$ as follows

$$\mathrm{AAD}(\mathbf{F},\hat{\mathbf{F}}) = \frac{1}{L\times W\times H}\sum_{i=1}^{L\times W\times H}\arccos\frac{\mathbf{F}[i]\cdot\hat{\mathbf{F}}[i]}{||\mathbf{F}[i]||\,||\hat{\mathbf{F}}[i]||}/\pi, \tag{7.11}$$

where $\cdot$ denotes the dot product, $||\cdot||$ denotes the magnitude of a vector. Note that if either of the vectors is the zero vector, then the value of arccos is undefined. In this case, we set it to the maximal error of $\pi$.

Table 7.3 shows PSNR $\uparrow$ (higher is better) and AAD $\downarrow$ (lower is better) values for all data sets and methods we have experimented. We highlight the best value for each data set and metric in bold. As we can see, CCNN dominates the PSNR rating, showing that

TABLE 7.3

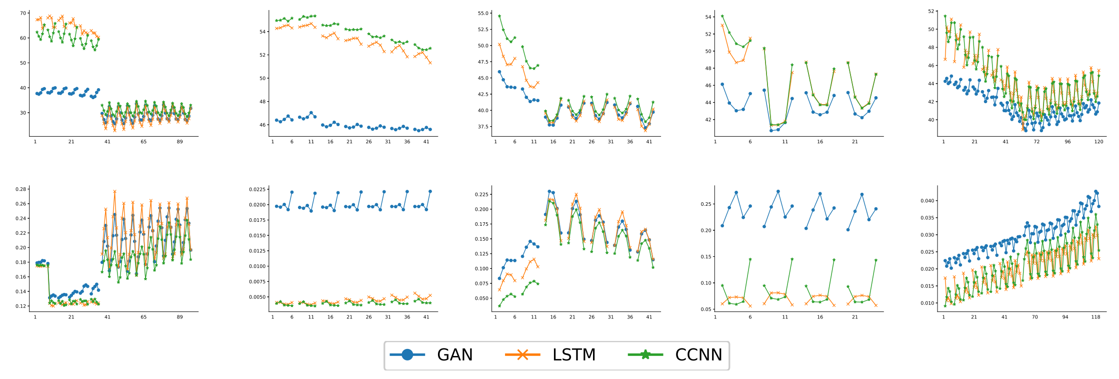
PSNR ↑ AND AAD ↓ FOR ALL DATA SETS AND METHODS

| data set | method | PSNR ↑ | AAD ↓ |
|---|---|---|---|
| Square Cylinder | GAN | 32.401 | **0.027** |
| | LSTM | 41.883 | 0.187 |
| | CCNN | **42.049** | 0.170 |
| Tornado | GAN | 46.013 | 0.020 |
| | LSTM | 53.250 | 0.004 |
| | CCNN | **54.045** | **0.004** |
| Solar Plume | GAN | 43.506 | 0.237 |
| | LSTM | 46.548 | **0.069** |
| | CCNN | **46.930** | 0.087 |
| Hurricane | GAN | 40.509 | 0.156 |
| | LSTM | 41.381 | 0.148 |
| | CCNN | **42.999** | **0.128** |
| Supernova | GAN | 41.664 | 0.0307 |
| | LSTM | **44.326** | 0.0218 |
| | CCNN | 44.324 | **0.0217** |

The best method is highlighted in bold.

the network can adequately learn to synthesize missing vector fields with high precision in the magnitude of the three vector components. The results for AAD are mixed. From our experiments, we cannot select a clear favorite. However, CCNN performs either the best or is close to the best method except for the Square Cylinder data set.

Figure 7.6 shows PSNR and AAD values plotted over time. What we can see from the plots is that for every data set except the Solar Plume and Supernova data sets, there is a



(a) Square Cylinder (b) Tornado (c) Hurricane (d) Solar Plume (e) Supernova

Figure 7.6. PSNR (top) and AAD (bottom) plots over all time steps for five different data sets.

clear split between the intervals $\mathbf{F}^t$ and $\mathbf{F}^I$. While this is expected, it shows the difficulty in learning the temporal development in unsteady flow. The plots also make it quite visible that there is a big difference between the time steps within the interpolation interval. Typically PSNR (AAD) tends to be lower (higher) in the middle of the interval, as it is further away from the ground truth input. The only exception to this is CCNN for the Solar Plume data set. What goes hand in hand with this is a sharp fall in PSNR. This shows a direct correlation between AAD and PSNR for the Solar Plume data set and the CCNN method. Besides that, the plots do not consider feature regions and outside regions, in which the Solar Plume data set has a clear distinction.

To further investigate which method has better performance overall or in feature regions, we employ an *entropy-based evaluation*.

**Entropy-based evaluation.** For a more detailed evaluation of the feature regions, we compute the entropy field for every data set at every time step and then incorporate these fields into the metric computation. More specifically, let $E$ be the entropy field of $\mathbf{F}$, then we update the PSNR computation to

$$\hat{\text{PSNR}}(\mathbf{F},\hat{\mathbf{F}},E) = 20\log_{10} I(\mathbf{F}) - 10\log_{10}\hat{\text{MSE}}(\mathbf{F},\hat{\mathbf{F}},E), \tag{7.12}$$

where $\hat{\text{MSE}}$ is the entropy-based filtered MSE defined as

$$\hat{\text{MSE}}(\mathbf{F},\hat{\mathbf{F}},E) = \frac{1}{L\times W\times H}\sum_{i=1}^{L\times W\times H}\frac{\hat{E}[i]}{3}\sum_{j\in u,v,w}(\mathbf{F}[i]_j - \hat{\mathbf{F}}[i]_j)^2, \tag{7.13}$$

and

$$\hat{E}[i] = \begin{cases} 1,\ if\ E[i] \geq \delta_e \\ 0,\ if\ E[i] < \delta_e \end{cases}$$

where $\delta_e$ is the entropy threshold as listed in Table 7.2.

Similarly, we adapt AAD to

$$\hat{\text{AAD}}(\mathbf{F},\hat{\mathbf{F}},E) = \frac{1}{L\times W\times H}\sum_{i=1}^{L\times W\times H}\hat{E}[i]\arccos\frac{\mathbf{F}[i]\cdot\hat{\mathbf{F}}[i]}{\|\mathbf{F}[i]\|\|\hat{\mathbf{F}}[i]\|}/\pi. \tag{7.14}$$

Table 7.4 shows the entropy-based metrics for all data sets and methods. Note that the values are lower due to evaluating fewer non-zeros, but averaging based on the size of the vector field. We can see that the PSNR value for the Square Cylinder and Tornado data sets decreases, while it increases for other data sets. This means that the first two data sets are reconstructed worse in the feature region than on the boundary, whereas the latter three experience a better synthesis in the feature region. The main contributor to this is the size of the vector field. A larger vector field has more voxels in the boundary region compared to the ones in the feature region. Another interesting finding was that, among the three methods, GAN gains most from only considering high-entropy regions. That being said, the relative changes were almost the same across all data sets and methods, hinting that these methods were already performing well at the center, but not in the outer regions. For the entropy-based AAD, all methods improve vastly, showing that errors in

TABLE 7.4

ENTROPY BASED PSNR $\uparrow$ AND AAD $\downarrow$ FOR ALL DATA SETS AND METHODS

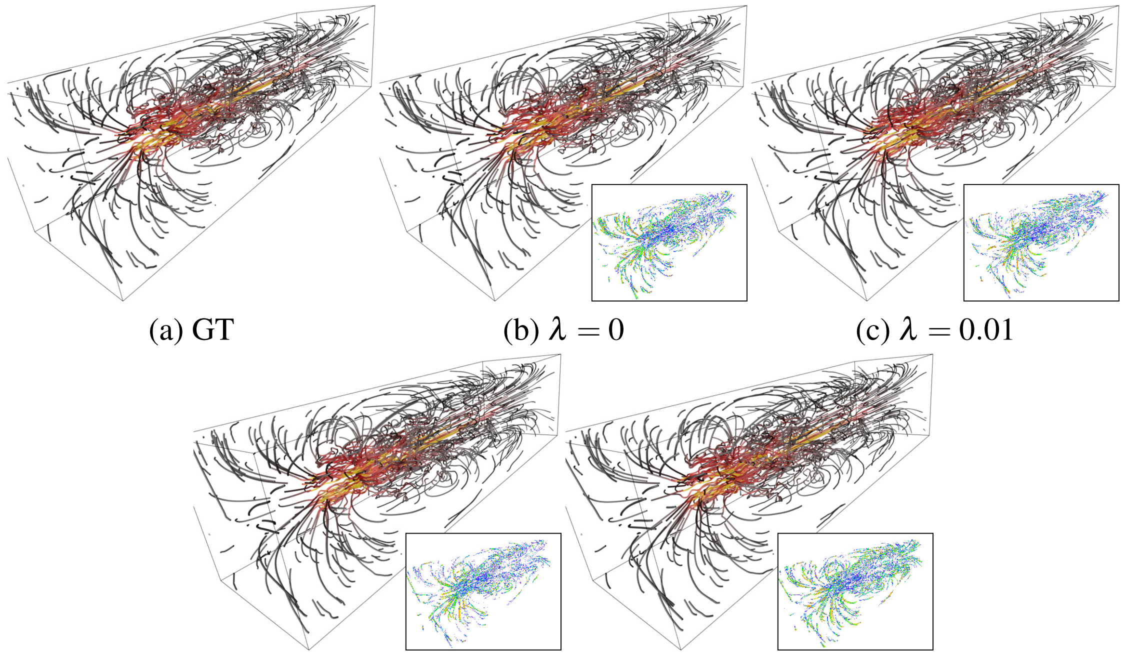
| data set | method | PSNR $\uparrow$ | AAD $\downarrow$ |
|---|---|---|---|
| Square Cylinder | GAN | 30.684 | $8.33*10^{-3}$ |
| | LSTM | **36.115** | $8.37*10^{-3}$ |
| | CCNN | 35.867 | $\mathbf{7.37*10^{-3}}$ |
| Tornado | GAN | 42.470 | $8.38*10^{-5}$ |
| | LSTM | 49.877 | $2.12*10^{-5}$ |
| | CCNN | **50.242** | $\mathbf{1.91*10^{-5}}$ |
| Solar Plume | GAN | 54.687 | $4.04*10^{-3}$ |
| | LSTM | 58.518 | $9.17*10^{-4}$ |
| | CCNN | **58.753** | $\mathbf{9.09*10^{-4}}$ |
| Hurricane | GAN | 62.754 | $1.08*10^{-3}$ |
| | LSTM | 63.208 | $1.04*10^{-3}$ |
| | CCNN | **64.045** | $\mathbf{9.14*10^{-4}}$ |
| Supernova | GAN | **45.983** | $2.84*10^{-3}$ |
| | LSTM | 30.267 | $2.19*10^{-2}$ |
| | CCNN | 30.681 | $\mathbf{2.55*10^{-3}}$ |

The best method is highlighted in bold.

vector angles are mainly present in the boundary region, which occupies the largest part of the vector field. Although all methods have a significant change in AAD, CCNN gains the most, hinting that it synthesizes vectors in feature regions with a better angle accuracy than outside those regions.

As we can see, CCNN is performing the best among the three versions in terms of

PSNR, visual quality, and almost always in AAD as well.


(a) GT (b) $\lambda = 0$ (c) $\lambda = 0.01$

(d) $\lambda = 0.05$ (e) $\lambda = 0.1$

Figure 7.7. Comparison of different $\lambda$ settings using the Solar Plume data set. Images show the pathlines.

### 7.3.4 Hyperparameter Study

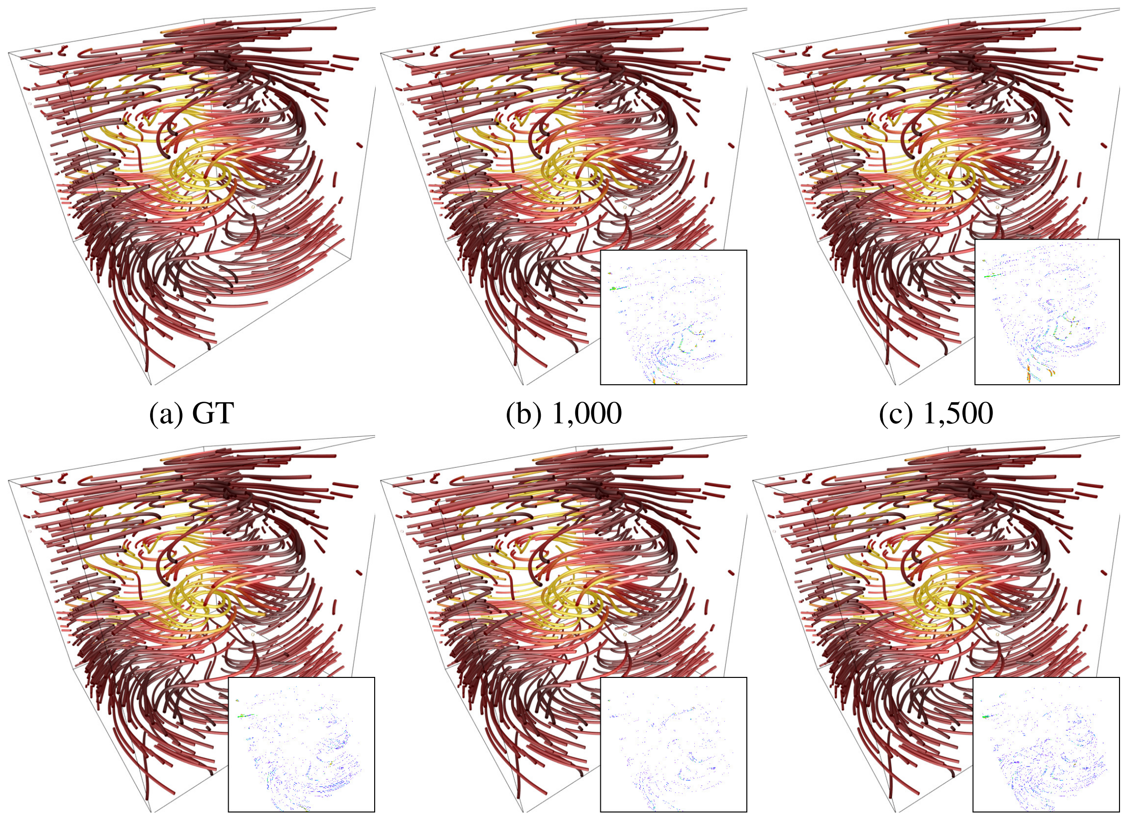
To evaluate CCNN, we analyze the following hyperparameter settings: the choice of $\lambda$, the number of training epochs, the crop size, the interpolation interval $s$, and whether a decomposition into the three channels is necessary.

**Loss composition factor $\lambda$.** Figure 7.7 shows different settings of $\lambda$ for the Solar Plume data set. Although not instantly visible, the quality of the synthesized vectors at the center of the vector field tends to be higher. This is visible on close inspection of the difference images, showing that $\lambda = 0.05$ is overall clearer and has more white space than the other two. While this has not a huge impact on PSNR (46.93 (b), 46.42 (c), and 46.67 (d)) the impact on AAD is remarkable. With $\lambda = 0$ having an AAD of 0.087, the slightest increase for $\lambda$ to 0.01 improves AAD to 0.053 already. Increasing it even further to $\lambda = 0.05$ shows a smaller yet still important impact, resulting in an AAD of 0.042. Although expected, it is generally not easy to pick the right factor for $\lambda$, as a too-large one harms PSNR. This can be seen when increasing $\lambda$ to 0.1 (e). Not only does the visual quality of the traced pathlines decreases, but also PSNR decreases to 46.06, and AAD increases to 0.08. This indicates an over-correction during training based on the AAD, making the overall result worse in terms of both PSNR and AAD. We generally recommend using $\lambda = 0.05$ as a safe choice.

**Training epochs.** We investigate the impact of the number of epochs trained on visual quality, PSNR, and AAD values. For this investigation, we use the Tornado and Solar Plume data sets.

Figure 7.8 shows the pathlines for the Tornado data set after different training epochs. We can see that there is no drastic change between the results of the different epochs. However, the difference images show that there is an improvement in quality at the bottom of the vector field. Although the rendering shows steady improvement, PSNR fluctuates from $51,55$ (b) to 54.04 (c), followed by 51.99 (d), and then increasing to 52.13 (e) and 52.16 (f). AAD, on the other hand, shows a steady decreasing trend (0.0042, 0.0039, 0.0034, 0.0030, and 0.0032 for (b) (c) (d) (e), and (f), respectively) leading to the steady improvement in visual quality.

Similar to this, in Figure 7.9, we can see the same behavior for the Solar Plume data set. While there are only small differences between 1,000 (b) and 1,500 (c) epochs, the core structure of the plume clears up at 2,000 (d) epochs. After that, it first gets worse at 2,500 (e) and then clears up again at 3,000 (f) epochs. Here, however, PSNR (AAD) increases


(a) GT (b) 1,000 (c) 1,500

(d) 2,000 (e) 2,500 (f) 3,000

Figure 7.8. Comparison of different training epochs using the Tornado data set. Images show the pathlines.

(decreases) steadily from 45.49 (0.193) at 1,000 epochs to 48.23 (0.044) at 3,000 epochs. This shows that there is no apparent convergence, and it might be necessary to train for further epochs. However, with an average training time of roughly a minute per epoch, training $3,000$ epochs already takes more than two days, making it infeasible to continue training. Furthermore, the example of the Tornado data set shows that for visual quality, both PSNR are AAD are essential metrics for a comprehensive evaluation.

**Crop size.** As big data sets cannot entirely fit into memory, we introduce cropping. Besides reducing the memory requirements, cropping also increases the training speed. To investigate the impact of the crop size on the quality of the synthesized vector fields, we

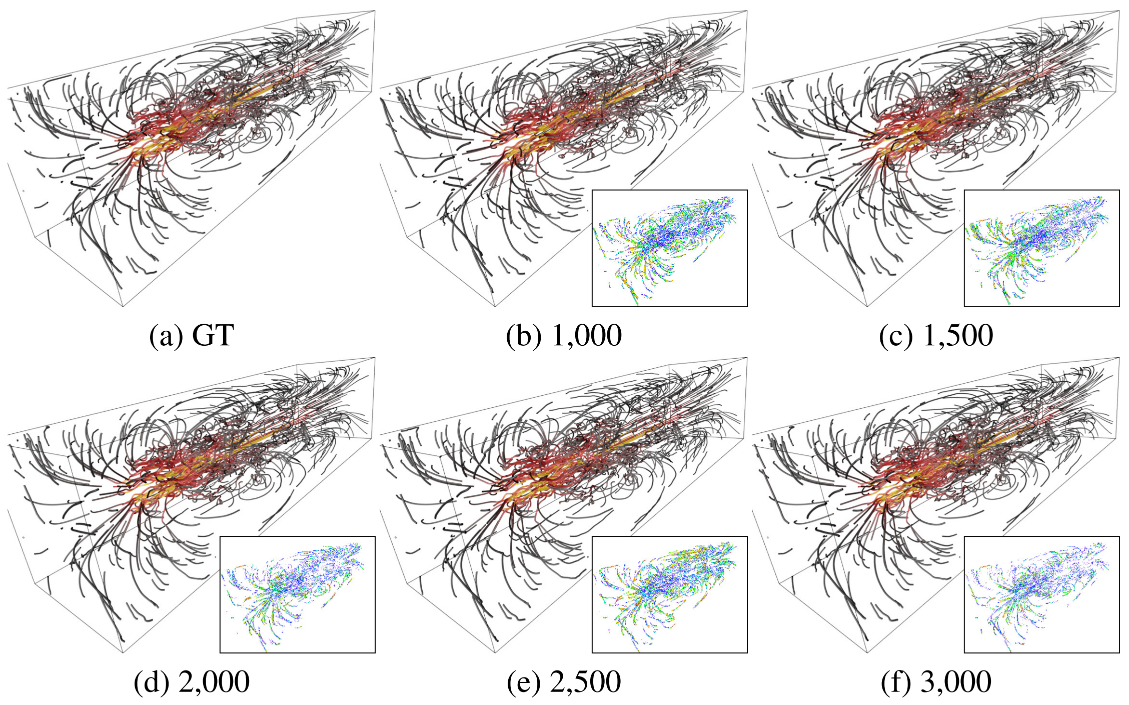


Figure 7.9. Comparison of different training epochs using the Solar Plume data set. Images show the pathlines.

used the Supernova data set with a crop size of $32 \times 32 \times 32$, $64 \times 64 \times 64$, and $128 \times 128 \times 128$. Figure 7.10 shows the traced pathlines for the three different crop sizes. From the difference image at the bottom-right corner of each image, it is easily visible that $128 \times 128 \times 128$ (d) shows fewer errors than the other two versions. Interestingly, $32 \times 32 \times 32$ (b) also shows fewer errors than $64 \times 64 \times 64$ (c). When it comes to PSNR, the same is visible comparing 39.6 (b), 43.67 (c), and 44.32 (d). AAD values reveal the same trend with 0.042 (b), 0.025 (c), and 0.022 (d). Although AAD did not change as drastically between $64 \times 64 \times 64$ and $128 \times 128 \times 128$, the minor change in both PSRN and AAD are enough to argue for a bigger crop size. As expected, average training per epoch time favors the $32 \times 32 \times 32$ version (10.32 seconds) over $64 \times 64 \times 64$ (22.36 seconds), and $128 \times 128 \times 128$ (105.74 seconds). This means, if the quality is not the highest priority, one should opt for a smaller crop size, allowing much faster training.

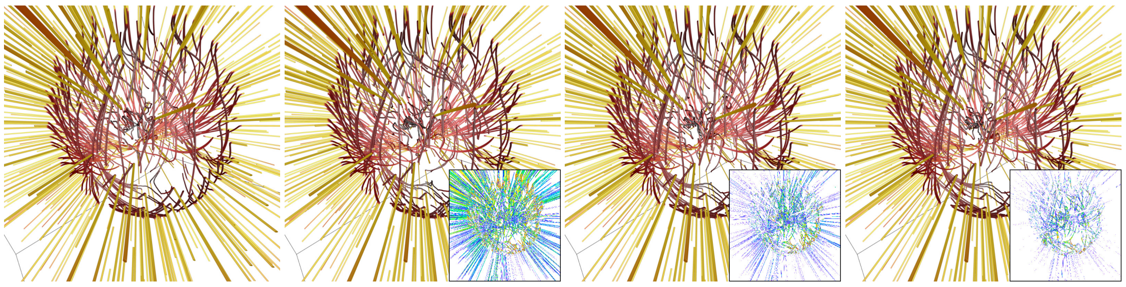

(a) GT (b) $32 \times 32 \times 32$ (c) $64 \times 64 \times 64$ (d) $128 \times 128 \times 128$

Figure 7.10. Comparison of different crop sizes using the Supernova data set. Pathlines are traced over all time steps.

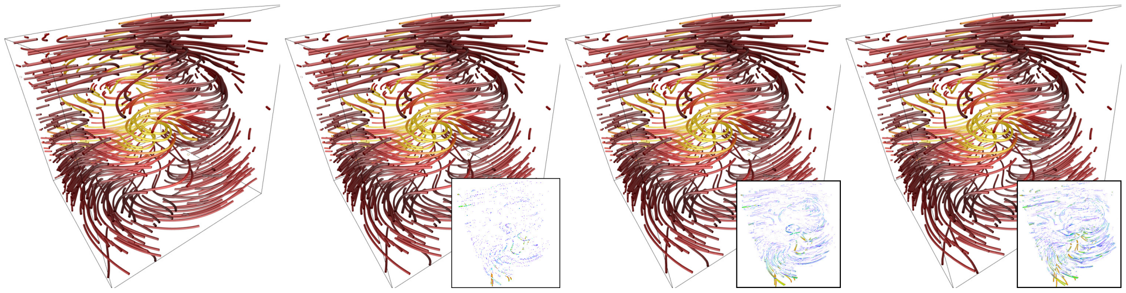

(a) GT (b) $s = 6$ (c) $s = 8$ (d) $s = 10$

Figure 7.11. Comparison of different interpolation interval $s$ using the Tornado data set. Pathlines are traces over all time steps.

**Interpolation interval.** Another important parameter for our framework is the interpolation interval $s$. Figure 7.11 shows a comparison of three different settings for $s$. As expected, a short interval of $s = 6$ (b) performs much better than longer intervals. That being said, there is a big difference between $s = 8$ (c) and $s = 10$ (d), showing that after $s = 6$, there is a sharp decline in quality. This can also be seen with a decrease in PSNR with 54.04 ($s = 6$), 50.39 ($s = 8$), and 48.88 ($s = 10$), as well as an increase in AAD 0.004 ($s = 6$), 0.009 ($s = 8$), and 0.011 ($s = 10$). This means that $s = 6$ is a good choice.

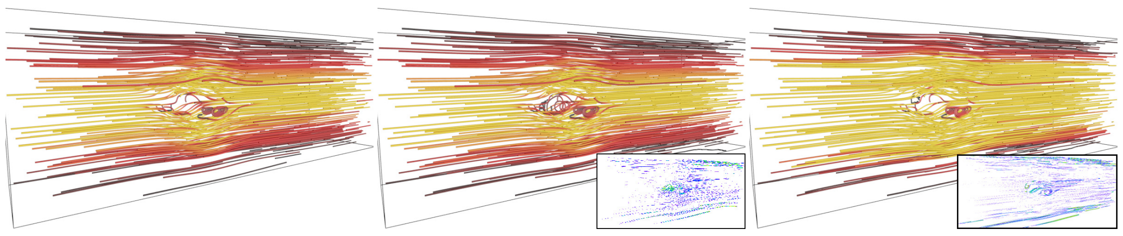

(a) GT (b) decomposed (c) combined

Figure 7.12. Comparison of decomposed and combined approaches using the Square Cylinder data set. Pathlines are traces over all time steps.

**Channel decomposition.** We use the Square Cylinder data set to investigate whether the decomposition does have a significant impact. The version without decomposition has the same network architecture as CCNN, except that instead of splitting the three channels in *u*, *v*, and *w*-net, it is a single network with three input channels instead of one. Figure 7.12 shows the pathlines traced over all time steps. As we can see, the decomposed variant (b) synthesizes the feature region with much higher accuracy than the variant without decomposition (c). Even in the boundary regions, the decomposition helps improving reconstruction accuracy. All this is reflected in PSNR (42.05 and 36.71) and AAD (0.17 and 0.19). Although the decomposition presumably should take a longer time to train, both variants take the same amount (about 50.17 seconds per epoch).

## 7.4 Conclusions and Future Work

We have studied three deep learning methods (CCNN, LSTM, GAN) to reconstruct intermediate vector fields for unsteady flow data. We design neural nets that take a sequence of vector fields for training. Once trained, the networks can synthesize intermediate vector fields from two vector fields at both ends of an interpolation interval as input. Compared with GAN and LSTM, CCNN yields synthesized vector fields of better visual quality, both qualitatively and quantitatively.

In the future, we would incorporate CCNN into an in-situ scenario. During pre-processing, we will train the network with pre-run data. Then at simulation time, we will output temporally sparsely sampled VFD for storage saving. During post-processing, we will interpolate unsaved intermediate time steps. With a recommended interpolation interval of $s = 6$, we can synthesize missing vector fields with high accuracy after training CCNN for only $1,500$ epochs.

# CHAPTER 8

# SPECTRUM-PRESERVING SPARSIFICATION FOR VISUALIZATION OF BIG GRAPHS

In this chapter, I present spectrum-preserving sparsification (SPS), a spectrum-preserving framework for sparsification and visualization of big graph data. This is achieved by first doing edge sparsification followed by iterative node reduction to generate multiple levels of a given graph. Following this, the layout on the coarsest level is computed and then mapped back to denser levels using an eigenmap procedure. This allows to quickly compute spectrum-based layouts and clustering for large graphs, as well as interactive level-of-detail exploration.[1]

## 8.1 Background

Consider a graph $G = (N, E, w)$ where $N$ and $E$ are the node set and edge set respectively, and $w$ is a weight function that assigns positive weights to all edges. The symmetric diagonally dominant Laplacian matrix of $G$ can be constructed as follows

$$\mathbf{L}_G(n_i, n_j) = \begin{cases} -w_{ij} & \text{if } e_{ij} \in E, \\ \sum_{e_{ik} \in E} w_{ik} & \text{if } n_i = n_j, \\ 0 & \text{otherwise.} \end{cases} \tag{8.1}$$

[1]This work has been accepted as M. Imre, J. Tao, Y. Wang, Z. Zhao, Z. Feng, and C. Wang. Spectrum-preserving sparsification for visualization of big graphs. *Computers & Graphics*, 87:89–102, 2020

where $n_i$ is a node, $e_{ij}$ is the edge between $n_i$ and $n_j$, and $w_{ij}$ is the weight of $e_{ij}$. Graph sparsification aims to find $G' = (N, E', w')$, a subgraph or *sparsifier* of $G$ that maintains the same set of nodes but fewer edges. To tell if two graphs have similar spectra, we usually use the following Laplacian quadratic form

$$\mathbf{x}^T \mathbf{L}_G \mathbf{x} = \sum_{e_{ij} \in E} w_{ij} \big(\mathbf{x}(n_i) - \mathbf{x}(n_j)\big)^2, \tag{8.2}$$

where $\mathbf{x} \in \mathbb{R}^N$ is a real vector. Two graphs $G$ and $G'$ are $\sigma-$*spectrally similar* if the following condition holds for all real vectors $\mathbf{x} \in \mathbb{R}^N$

$$\frac{\mathbf{x}^T \mathbf{L}_{G'} \mathbf{x}}{\sigma} \leq \mathbf{x}^T \mathbf{L}_G \mathbf{x} \leq \sigma \mathbf{x}^T \mathbf{L}_{G'} \mathbf{x}. \tag{8.3}$$

Defining the relative condition number to be $\kappa(\mathbf{L}_G, \mathbf{L}_{G'}) = \lambda_{\max}/\lambda_{\min}$, where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest nonzero generalized eigenvalues satisfying

$$\mathbf{L}_G \mathbf{u} = \lambda \mathbf{L}_{G'} \mathbf{u}, \tag{8.4}$$

where $\mathbf{u}$ is the generalized eigenvector of $\lambda$. It can be further shown that $\kappa(\mathbf{L}_G, \mathbf{L}_{G'}) \leq \sigma^2$, which indicates that a smaller relative condition number or $\sigma^2$ corresponds to a higher spectral similarity.

The state-of-the-art nearly-linear time spectral sparsification algorithm leverages an edge sampling scheme that sets sampling probabilities proportional to edge *effective resistances* [189]. However, it becomes a chicken-and-egg problem since even approximately computing edge effective resistances by leveraging the Johnson-Lindenstrauss Lemma still requires solving the original graph Laplacian matrix $\log |N|$ times and thus can be extremely expensive for very large graphs, not to mention directly computing the Moore-Penrose pseudo inverse of graph Laplacians. For example, a recent work on graph drawing using spectral sparsification shows the major computational bottleneck is due to estimating edge effective resistances (by computing the Moore-Penrose pseudo inverse): even for a relatively small graph with $|N| = 7,885$, $|E| = 427,406$, the spectral sparsification procedure can take several hours to complete [58].

## 8.2 Our Approach

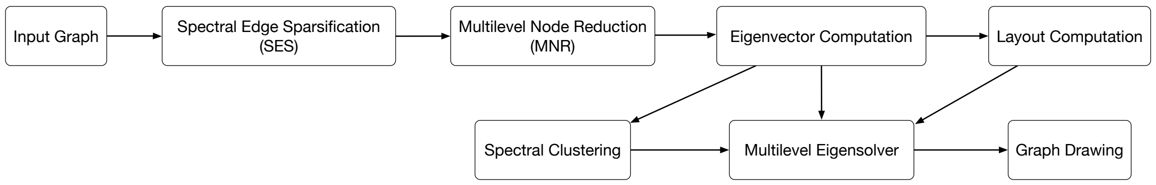


Figure 8.1. The diagram of our SPS framework. Layout computation could use the eigenvector-based layout, t-SNE-based layout, or any other graph drawing algorithm.

Figure 8.1 shows an overview of our SPS framework. Given the input graph, we first perform SES (Chapter 8.2.1) to reduce the number of edges. Next, based on the edge sparsification results, we perform MNR (Chapter 8.2.2) to further produce multiple levels of node simplification. This leads to a fairly small graph that preserves spectrally-important nodes and edges, allowing us to compute the eigenvectors of the graph Laplacian in an efficient manner. We then use these eigenvectors as input for dimensionality reduction using t-distributed stochastic neighbor embedding (t-SNE) [204, 205] and for spectral clustering using k-means. For spectral graph drawing (Chapter 8.2.3), we can layout the most simplified level of the graph based on the eigenvectors, t-SNE, and clustering results, where node positions are determined by either the leading eigenvectors or t-SNE projection and node colors are determined by spectral cluster labels. To obtain the graph drawing at a finer level, we can compute positions for newly-added nodes based on a multilevel eigensolver without recomputing the layout. Note that our SPS framework can readily work with other graph drawing algorithms by replacing the layout based on eigenvectors, t-SNE, or with another one.

### 8.2.1 Spectral Edge Sparsification

We outline the key steps of the proposed method for spectral graph sparsification of a given undirected graphs as follows: (1) low-stretch spanning tree extraction based on the original graph [2, 62]; (2) spectral embedding and criticality ranking of off-tree edges using approximate generalized eigenvectors leveraging the recent spectral perturbation analysis framework [64]; (3) subgraph densification by recovering a small portion of the most “spectrally critical” off-tree edges to the spanning tree; and (4) subgraph edge weight scaling via stochastic gradient descent (SGD) optimization.

In the following, we assume that $G = (N, E, w)$ is a weighted, undirected, and connected graph, whereas $G' = (N, E', w')$ is its graph sparsifier. The descending generalized eigenvalues of $\mathbf{L}_{G'}^{+}\mathbf{L}_G$ are denoted by $\lambda_{\max} = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$, where $\mathbf{L}_{G'}^{+}$ denotes the Moore-Penrose pseudoinverse of $\mathbf{L}_G$.

**Spectral distortion of spanning-tree sparsifiers.** Spielman [193] showed that there are not too many large generalized eigenvalues for spanning tree sparsifiers: $\mathbf{L}_{G'}^{+}\mathbf{L}_G$ has at most $k$ generalized eigenvalues greater than $\mathrm{st}_{G'}(G)/k$, where $\mathrm{st}_{G'}(G)$ is the total stretch of the spanning-tree subgraph $G'$ with respect to the original graph $G$ that can be considered as the spectral distortion due to the spanning tree approximation. Recent research shows that every graph has a *low-stretch spanning tree* (LSST) such that the total stretch $\mathrm{st}_{G'}(G)$ can be bounded by [188]

$$O(|E|\log|N|\log\log|N|) \geq \mathrm{st}_{G'}(G) = \mathrm{tr}(\mathbf{L}_{G'}^{+}\mathbf{L}_G) = \textstyle\sum_{i=1}^{|N|}\lambda_i \geq \sigma^2, \tag{8.5}$$

where $\mathrm{tr}(\mathbf{L}_{G'}^{+}\mathbf{L}_G)$ is the trace of $\mathbf{L}_{G'}^{+}\mathbf{L}_G$. As a result, it is possible to construct an ultra-sparse yet spectrally similar sparsifier by recovering only a small portion of *spectrally critical* off-tree edges to the spanning tree. For example, $\sigma$-similar spectral sparsifiers with $O(|E|\log|N|\log\log|N|/\sigma^2)$ off-tree edges can be constructed in nearly linear time [64].

**Edge embedding with generalized eigenvectors.** To identify the off-tree edges that should be recovered to the spanning tree to dramatically reduce spectral distortion (the total stretch), Feng [64] introduced an off-tree edge embedding scheme using generalized eigenvectors, which is based on the following spectral perturbation framework. Considering the following first-order eigenvalue perturbation problem

$$\mathbf{L}_G(\mathbf{u}_i+\delta\mathbf{u}_i) = (\lambda_i+\delta\lambda_i)(\mathbf{L}_{G'}+\delta\mathbf{L}_{G'})(\mathbf{u}_i+\delta\mathbf{u}_i), \tag{8.6}$$

where a perturbation $\delta\mathbf{L}_{G'}$ is applied to $\mathbf{L}_{G'}$, which results in perturbations in generalized eigenvalues $\lambda_i+\delta\lambda_i$ and eigenvectors $\mathbf{u}_i+\delta\mathbf{u}_i$ for $i=1,\ldots,n$, respectively. The first-order perturbation analysis shows that [64]

$$-\frac{\delta\lambda_i}{\lambda_i} = \mathbf{u}_i^T\delta\mathbf{L}_{G'}\mathbf{u}_i, \tag{8.7}$$

which indicates that the reduction of $\lambda_i$ is proportional to the Laplacian quadratic form of $\delta\mathbf{L}_{G'}$ with the generalized eigenvector $\mathbf{u}_i$. Consequently, if the eigenvector $\mathbf{u}_1$ is applied, a significant reduction of the largest generalized eigenvalue $\lambda_1$ can be achieved. Once all large generalized eigenvalues are dramatically reduced, the subgraph $G'$ can serve as a very good spectral sparsifier of $G$.

To achieve effective reductions of large generalized eigenvalues, we exploit the following two key steps: (1) recover a small portion of most spectrally-critical off-tree edges into the spanning tree; (2) scale up edge weights in the subgraph $G'$ to further improve the approximation. Additionally, the scaling factor obtained for each edge can be treated as its *spectral importance* in the subgraph: a larger scaling factor may indicate a more important role that the edge plays in mimicking the original graph.

**Subgraph densification.** If we denote $\mathbf{e}_{jk} \in \mathbb{R}^N$ the vector with only the $j$-th element being 1, the $k$-th element being $-1$, and others being 0, then the eigenvalue perturbation due to the inclusion of all off-tree edges can be expressed as follows

$$-\frac{\delta\lambda_i}{\lambda_i} = \mathbf{u}_i^T \delta\mathbf{L}_{G',\max}\mathbf{u}_i = \sum_{e_{jk}\in E\setminus E'} w_{jk}\left(\mathbf{e}_{jk}^T\mathbf{u}_i\right)^2 = \sum_{e_{jk}\in E\setminus E'} H_{jk}(\mathbf{u}_i), \tag{8.8}$$

where $\delta\mathbf{L}_{G',\max} = \mathbf{L}_G - \mathbf{L}_{G'}$ denotes the Laplacian including all off-tree edges, $H_{jk}(\mathbf{u}_i)$ denotes the Joule heat (power dissipation) of edge $e_{jk}$ by considering the undirected graph $G$ as a resistor network and $\mathbf{u}_i$ as the voltage vector. Equation (8.8) can also be considered as a spectral off-tree edge embedding scheme using generalized eigenvectors. It indicates that when using the first few dominant generalized eigenvectors for off-tree edge embedding, the top few generalized eigenvalues can be dramatically reduced by recovering the most spectrally-critical off-tree edges back to the spanning tree. In practice, we can leverage approximate eigenvectors computed via a few steps of generalized power iterations for good efficiency [64]:

- **Step 1:** Compute an approximate generalized eigenvector $\mathbf{h}_t$ from an initial random vector $\mathbf{h}_0$ via $t$-step generalized power iterations

$$\mathbf{h}_t = \left(\mathbf{L}_{G'}^{+}\mathbf{L}_G\right)^t\mathbf{h}_0 = \left(\mathbf{L}_{G'}^{+}\mathbf{L}_G\right)^t\sum_{i=1}^{|N|}\alpha_i\mathbf{u}_i = \sum_{i=1}^{|N|}\alpha_i\lambda_i^t\mathbf{u}_i; \tag{8.9}$$

- **Step 2:** Compute the Joule heat of all off-tree edges with $\mathbf{h}_t$ by

$$\begin{aligned}\mathbf{h}_t^T\delta\mathbf{L}_{G',\max}\mathbf{h}_t &= \textstyle\sum_{i=1}^{|N|}(\alpha_i\lambda_i^t)^2(\lambda_i-1)\\ &= \textstyle\sum_{e_{jk}\in E\setminus E'} w_{jk}\sum_{i=1}^{|N|}\alpha_i^2\lambda_i^{2t}\left(\mathbf{e}_{jk}^T\mathbf{u}_i\right)^2 = \sum_{e_{jk}\in E\setminus E'} H_{jk}(\mathbf{h}_t).\end{aligned} \tag{8.10}$$

Similar to Equation (8.8), Equation (8.10) also allows embedding generalized eigenvalues into the Laplacian quadratic form of each off-tree edge and thus ranking off-tree edges according to their spectral criticality levels: recovering the off-tree edges with the largest edge Joule heat values will most significantly decrease the largest generalized eigenvalues.

In practice, using a small number (e.g., $0 < t < 3$) of power iterations suffices for the embedding purpose.

**Subgraph edge scaling via SGD iterations.** Once a sufficient number of off-tree edges ($O(|E| \log |N| \log\log |N| / \sigma^2)$) are selected and recovered to the spanning tree, the subgraph can already well mimic the original graph by approximating its first few Laplacian eigenvectors. To further mitigate the accuracy loss due to the missing edges in the subgraph, we introduce a novel edge scaling procedure that scales up edge weights in the subgraph so that $\lambda_1$ can be substantially reduced. To this end, we express the dominant eigenvalue perturbation $\delta\lambda_1$ in terms of edge weights perturbation $\delta w$ as

$$-\frac{\delta\lambda_1}{\lambda_1} = \mathbf{u}_1^T \delta\mathbf{L}_{G'} \mathbf{u}_1 = \sum_{e_{jk} \in E'} \delta w_{jk} \left(\mathbf{e}_{jk}^T \mathbf{u}_1\right)^2, \tag{8.11}$$

which directly gives the sensitivity of $\lambda_1$ with respect to each edge weight $w_{jk}$ as

$$\frac{\delta\lambda_1}{\delta w_{jk}} = -\lambda_1 \left(\mathbf{e}_{jk}^T \mathbf{u}_1\right)^2 \approx -\lambda_1 \left(\mathbf{e}_{jk}^T \mathbf{h}_t\right)^2. \tag{8.12}$$

With the weight sensitivity expressed in Equation (8.12), SGD iterations can be performed for scaling up edge weights: during each iteration of SGD, a random vector is first generated and used to compute the approximate dominant eigenvector ($\mathbf{h}_t$) using Equation (8.9) as well as edge weight sensitivities using Equation (8.12) for the following edge scaling step; when the edge weight sensitivities are small enough, we can terminate the SGD iterations. Since edge weights in $G'$ will be updated during each SGD iteration, we need to solve a new subgraph Laplacian matrix $\mathbf{L}_{G'}$ for updating the approximate eigenvector $\mathbf{u}_1$ in Equation (8.12). This can be achieved by leveraging recent graph-theoretic algebraic multigrid algorithms that have shown highly scalable performance for solving large graph Laplacians [125, 141, 236]. Since the subgraph structure remains unchanged with only edge weights adjusted during the SGD iterations, it is also possible to incrementally update graph Laplacian solvers for achieving better computation efficiency.

### 8.2.2 Multilevel Node Reduction

To generate the reduced graph based on the original graph (in our case, the graph after SES), our MNR framework applies a spectrum-preserving node aggregation scheme where the node affinity metric is considered [237]. Given neighboring nodes $p$ and $q$, the node affinity between them is defined as [35, 141]

$$a_{p,q} = \frac{\|(\mathbf{X}_p, \mathbf{X}_q)\|^2}{(\mathbf{X}_p, \mathbf{X}_p)(\mathbf{X}_q, \mathbf{X}_q)}, \ (\mathbf{X}_p, \mathbf{X}_q) = \sum_{k=1}^{K} \left( \mathbf{x}_p^{(k)} \cdot \mathbf{x}_q^{(k)} \right), \tag{8.13}$$

where $\mathbf{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(K)})$ is a vector set with $K$ test vectors which are computed by applying a few Gauss-Seidel (GS) relaxations to the linear system of equations $\mathbf{L}_G \mathbf{x}^{(i)} = 0$ for $i = 1, \ldots, K$, starting with $K$ random vectors that are orthogonal to the all-one vector $\mathbf{1}$. If we consider $\tilde{\mathbf{x}}^{(i)}$ to be the approximate solution of $\mathbf{L}_G \mathbf{x}^{(i)} = 0$ after a few GS relaxations, and $\mathbf{x}^{(i)}$ to be the true solution, the error between $\tilde{\mathbf{x}}^{(i)}$ and $\mathbf{x}^{(i)}$ can be expressed as $\mathbf{e}_s^{(i)} = \mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}$. Due to the smoothing property of GS relaxation, $\mathbf{e}_s^{(i)}$ will only contain the smooth (low-frequency) modes of the initial error, while the oscillatory (high-frequency) modes of the initial error will be effectively removed [23]. Based on these $K$ smoothed vectors in $\mathbf{X}$, we are able to embed each node into a $K$-dimensional space such that nodes $p$ and $q$ are considered spectrally-close to each other if their low-dimensional embedding vectors, $\mathbf{x}_p \in \mathbb{R}^K$ and $\mathbf{x}_q \in \mathbb{R}^K$, are highly correlated. Thus, spectrally-similar nodes $p$ and $q$ can be aggregated together for node reduction purpose.

The node affinity metric $a_{p,q}$ also reflects the distance or strength of the connection between nodes $p$ and $q$. For example, the algebraic distance $d_{p,q}$ can be expressed by $d_{p,q} = 1 - a_{p,q}$, which can be used to represent the geometric distance in grid-structure graphs. Nodes with large affinity or small algebraic distance should be aggregated together to form the nodes in the reduced graph. Based on this node aggregation scheme, we can generate the next coarser-level graph by applying it to the original graph. To further reduce its size, we leverage a multilevel procedure by repeatedly applying the above

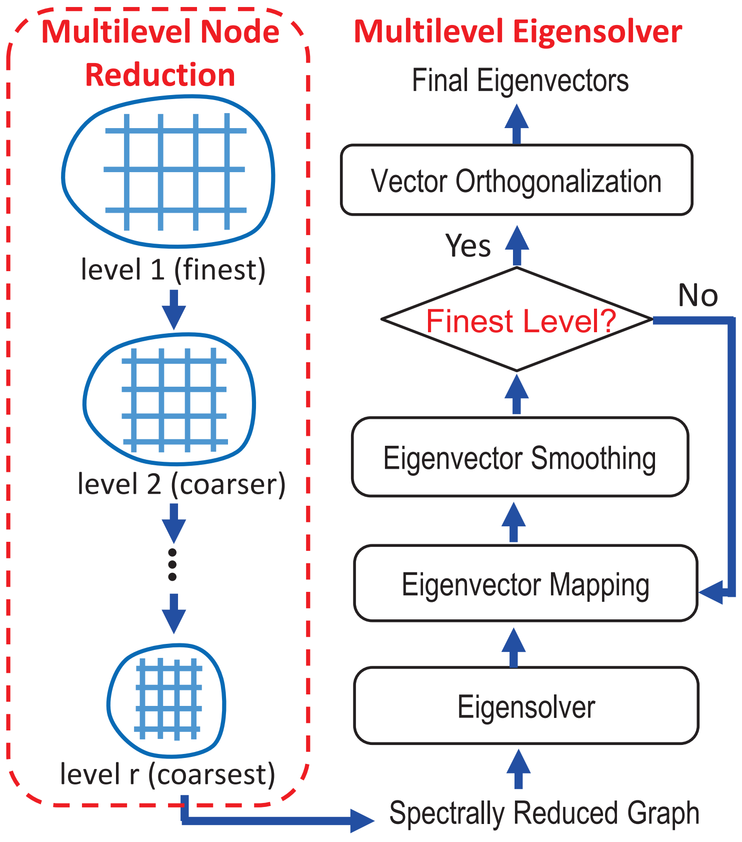


Figure 8.2. The framework of multilevel node reduction and multilevel eigensolver.

node reduction procedure to the current-level graph until the desired size of the reduced graph at the coarsest level is reached, as shown in Figure 8.2. Once the node aggregation scheme for each level is determined, we can define the graph mapping operators $\mathbf{H}_i^{i+1}$ (fine-to-coarse) and $\mathbf{H}_{i+1}^{i}$ (coarse-to-fine), which can be further leveraged for constructing the spectrally-reduced graph. For example, given the graph Laplacian $\mathbf{L}_G$ and the defined mapping operators from the finest level 1 to the coarsest level $r$, we can always uniquely compute the final reduced Laplacian by $\mathbf{L}_R = \mathbf{H}_G^R \mathbf{L}_G \mathbf{H}_R^G$, where $\mathbf{H}_G^R = \mathbf{H}_1^2 \mathbf{H}_2^3 \cdots \mathbf{H}_{r-1}^r$ and $\mathbf{H}_R^G = \mathbf{H}_2^1 \mathbf{H}_3^2 \cdots \mathbf{H}_r^{r-1}$.

The computational cost of node reduction scheme based on the above spectral node affinities is linear. This allows us to preserve the spectral properties of the original graph in a highly efficient and effective manner: the node aggregation scheme will preserve the smooth components in the first few Laplacian eigenvectors well, which is key to preserving the first few eigenvalues and eigenvectors of the original graph Laplacian in the reduced graphs.

Since only the first few nontrivial eigenvectors of the original graph Laplacian are needed for graph visualization tasks, they can efficiently and effectively be calculated by leveraging a multilevel eigensolver procedure [237], as shown in Figure 8.2. Instead of directly solving the eigenvalue problems on the original graph $G$, we will first reduce $G$ into a much smaller graph $R$ such that the eigenvectors of the reduced graph can be easily calculated. Once we get the eigenvectors of graph $R$, we will map them back to next finer level using the mapping operators defined during the MNR process. To further improve the solution accuracy of the mapped eigenvectors, a weighted-Jacobi-iteration-based eigenvectors smoothing (refinement) scheme is applied. The eigenvector mapping and smoothing procedures are recursively applied until the finest level graph is reached. Finally, all the eigenvectors for the finest level will be orthonormalized using the Gram-Schmidt process.

Note that the MNR process taken by SPS generates pseudo-nodes that are not in the node set of the original graph. As shown in Figure 8.3 (a), at each level of node reduction, our process essentially aggregates nodes into groups and creates a pseudo-node to represent each group. On the contrary, other node reduction methods do not create pseudo-nodes. As shown in Figure 8.3 (b), at each level of simplification, they simply sample the graph and output a subset of nodes from the node set of the original graph. In this process, no pseudo-nodes are created.

### 8.2.3 Spectral Graph Drawing

SPS is a practically efficient solution for spectrum-preserving graph sparsification and Laplacian eigenvalue computation. This enables us to tackle much bigger graphs previ-

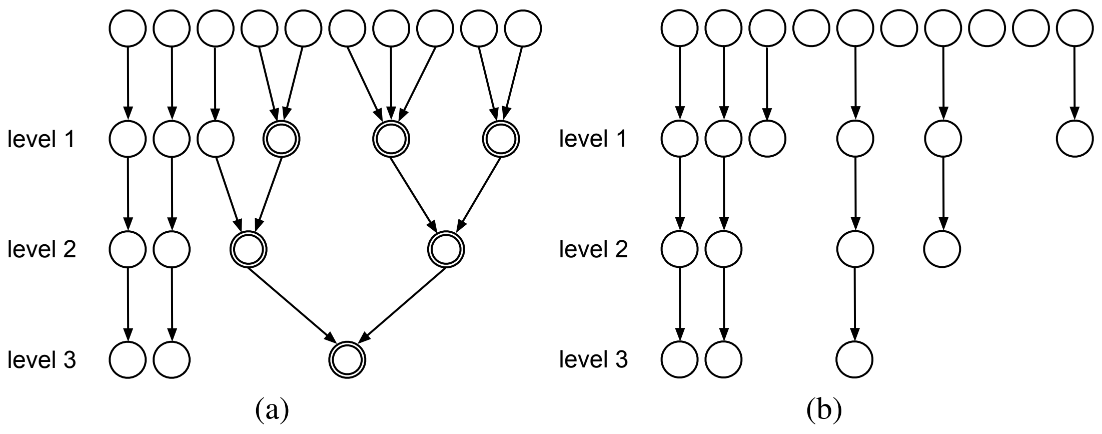


Figure 8.3. Comparison of different node reduction processes. (a) shows the node reduction process taken by our SPS (or METIS) method where double-circled nodes are pseudo-nodes newly created. (b) shows the node reduction process taken by graph sampling methods (such as DSS and FF, refer to Chapter 8.3.1).

ously impossible by creating spectrally-simplified graphs at various levels of detail for graph drawing and interaction. We present two different layouts to use in conjunction with SPS: the eigenvector-based (EIGEN) and t-SNE-based (t-SNE) layouts. The EIGEN layout lays out the graph vertices using certain eigenvectors of the related matrices (we use the two leading eigenvectors). The t-SNE layout employs t-SNE to create a 2D embedding based on the leading eigenvectors (we empirically use the first 50 dominant eigenvectors).

**Layout generation.** To generate a layout for visualizing a given graph, we propose the following steps as outlined below:

- **Step 1:** Apply SPS to simplify the graph $G_0$, yielding the sparsified graphs $G_1, G_2, \ldots, G_r$ and their associated sparse Laplacian matrix $\mathbf{L}_{G_i}, i \in \{0, 1, \ldots, r\}$ of size $|N_i|^2$. Note that $G_0$ is the original graph and $G_r$ is its most simplified form.
- **Step 2:** Perform an eigenanalysis [134, 195] on $\mathbf{L}_{G_r}$ to obtain the first $k'$ leading eigenvectors and their associated eigenvalues (we set $k' = 50$). Each of these eigenvectors is $|N_r|$-dimensional and every graph node has a $k'$-dimensional representation.
- **Step 3:** Identify the largest eigengap, i.e., the largest difference of two neighboring

eigenvalues, among the first $k'$ eigenvalues to determine the desired number of clusters $k$. Perform spectral clustering using k-means to obtain cluster labels for the $k$ different clusters.

- **Step 4:** Either use the two leading eigenvectors as 2D positions of the nodes (for the EIGEN layout), or perform dimensionality reduction, which maps the graph's node positions from $k'$D to 2D using t-SNE (for the t-SNE layout).
- **Step 5:** Map the cluster labels, eigenvectors, and t-SNE results from $G_r$ to $G_{r-1}$, and repeat this iteratively until the mapping from $G_1$ to $G_0$ is obtained.

After these steps, we hold all the data needed (2D coordinates, cluster labels, Laplacian matrix) to display the graph in 2D for the various levels of detail from $G_0$ to $G_r$. Nodes are colored to show their cluster memberships where neighboring clusters shown in the layout use different colors. To draw the graph at a given level of detail $i$, we position the nodes of $G_i$ according to the selected layout and draw a straight line for each edge present in $\mathbf{L}_{G_i}$. Note that our SPS algorithm is independent of the choices of graph layout. Although our layout algorithm is not interactive, the timing results in Table 8.1 show that the SPS algorithm allows efficient layout generation for large graphs.

**Graph interaction.** For graph interaction, we allow users to change the graph layout, the level of detail, and turn on or off edge bundling. Edge bundling is computed in real time as we avoid its pre-computation for every graph level by implementing FFTEB, the state-of-the-art edge bundling technique using the fast Fourier transform (FFT) [137].

TABLE 8.1

TIMING RESULTS (IN SECONDS) FOR ALL DATA SETS

| data set | nodes | edges | SPS | | | | | | $FF_E$ | | | | | $FF_N$ | | | | | DSS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | SES | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | |
| **small data sets (DSS compatible)** | | | | | | | | | | | | | | | | | | | |
| Facebook | 4,039 | 88,234 | 0.30 | 1.19 | 0.11 | 0.16 | 0.05 | 0.15 | 0.27 | 0.20 | 0.13 | 0.11 | 0.10 | 0.30 | 0.31 | 0.11 | 0.16 | 0.10 | 802.8 |
| Airfoil | 4,253 | 12,289 | 0.07 | 2.94 | 0.24 | 0.19 | 0.11 | 0.27 | 0.15 | 0.11 | 0.05 | 0.03 | 0.02 | 0.11 | 0.13 | 0.05 | 0.05 | 0.07 | 947.4 |
| ND3k | 9,000 | 3,279,690 | 5.93 | 1.18 | 0.17 | 0.19 | 0.07 | 0.17 | 4.02 | 4.04 | 4.22 | 3.90 | 2.68 | 2.98 | 2.43 | 2.43 | 2.19 | 2.29 | 12,774 |
| USPS10NN | 9,298 | 136,762 | 0.92 | 1.33 | 0.15 | 0.17 | 0.07 | 0.15 | 0.89 | 1.07 | 0.65 | 0.35 | 0.35 | 0.36 | 0.23 | 0.20 | 0.17 | 0.17 | 11,448 |
| Mycielskian14 | 12,287 | 3,695,512 | 6.42 | 1.23 | 0.11 | 0.17 | 0.07 | 0.19 | 3.36 | 3.19 | 3.11 | 2.90 | 2.90 | 3.35 | 3.15 | 3.33 | 2.81 | 2.69 | 27,181 |
| Appu | 14,000 | 1,839,104 | 11.60 | 1.33 | 0.14 | 0.21 | 0.06 | 0.14 | 2.86 | 2.30 | 2.79 | 1.96 | 2.69 | 5.00 | 6.32 | 7.55 | 5.66 | 4.78 | 34,236 |
| **big data sets (DSS incompatible)** | | | | | | | | | | | | | | | | | | | |
| Vsp | 21,996 | 1,221,028 | 5.36 | 1.30 | 0.17 | 0.20 | 0.08 | 0.15 | 8.85 | 5.57 | 5.40 | 3.90 | 2.72 | 2.38 | 2.14 | 2.09 | 2.09 | 1.94 | |
| Protein_DB | 36,417 | 2,154,174 | 11.19 | 1.36 | 0.18 | 0.20 | 0.08 | 0.15 | 16.30 | 9.15 | 6.43 | 4.77 | 4.06 | 6.38 | 4.04 | 3.09 | 3.01 | 3.05 | |
| Mesh | 40,000 | 79,600 | 0.64 | 1.31 | 0.15 | 0.21 | 0.08 | 0.16 | 6.43 | 3.10 | 1.86 | 1.09 | 0.49 | 4.47 | 2.18 | 0.90 | 0.62 | 0.26 | |
| CFD | 70,656 | 878,854 | 10.48 | 1.59 | 0.24 | 0.24 | 0.08 | 0.16 | 20.74 | 14.00 | 9.38 | 4.84 | 2.60 | 28.69 | 13.58 | 4.77 | 3.04 | 1.58 | |
| DBLP | 317,080 | 1,049,866 | 19.08 | 3.63 | 0.50 | 0.26 | 0.18 | 0.11 | 306.12 | 164.19 | 63.47 | 31.07 | 19.03 | 63.87 | 19.48 | 7.58 | 5.55 | 5.50 | |
| ND | 325,729 | 1,469,679 | 19.11 | 2.74 | 0.63 | 0.44 | 0.15 | 0.25 | 501.65 | 245.22 | 144.24 | 61.28 | 17.25 | 129.88 | 23.77 | 11.46 | 4.89 | 2.51 | |
| IL2010 | 451,554 | 1,082,232 | 9.47 | 3.06 | 0.91 | 0.53 | 0.20 | 0.20 | 855.67 | 388.49 | 202.90 | 97.24 | 51.84 | 773.92 | 300.86 | 130.06 | 60.46 | 25.30 | |

The data sets are ordered according to the number of nodes in the original graphs, and split into two groups (small and big data sets). The five levels of simplification under SPS is for the MNR step.

## 8.3 Results and Discussion

### 8.3.1 Data Sets and Methods

We experimented our approach with the graph data sets from different application domains as listed in Table 8.1. Among them, Facebook and DBLP are from the social network domain, recording a friend network (Facebook) and co-authorship relations (DBLP). Airfoil is a mesh graph from finite element analysis, ND3k is a graph generated from a 3D mesh problem, and Mesh is a $200 \times 200$ mesh graph with uniform edge weights. USPS10NN is a k-NN network for handwritten digit recognition. Mycielskian14 represents a triangle-free graph with the chromatic number of 14. Appu and Vsp are random graphs, representing the app benchmark from NASA Ames Research Center and a graph with a star-like structure. CFD is from computational fluid dynamics application representing a symmetric pressure matrix. ND is a web graph of the webpages of Notre Dame. IL2010 is a geographic network of the census blocks of Illinois. Protein_DB is the protein databank of an enzyme found in HIV.

To compare different graph sparsification methods, we evaluated the results of four methods: SPS (ours), *deterministic spectral sparsification* (DSS) [58], and two variants of a traversal-based sampling method named *forest fire* (FF) [135]. DSS picks edges with the largest effective resistances. Note that Eades et al. [58] also introduced a second variant of spectral sparsification, *stochastic spectral sparsification* (SSS). However, DSS has been shown to perform better than SSS. Hence, we only use DSS in our comparison, where the pseudoinverse is computed using OpenIMAJ [88]. As a probabilistic version of *snow-ball sampling* (SBS) [203], FF randomly selects a seed node with incident edges and adjacent nodes getting "burned" away recursively with a probability. In this work, we continue FF sampling until a desired number of edges ($\mathrm{FF_E}$) or nodes ($\mathrm{FF_N}$) are reached.

Besides DSS, the only other implementation publicly available is provided by Spielman, which is based on the effective-resistance sampling approach [189] and has been recently available for download [194]. However, such an implementation needs to set up input parameters carefully for each individual input graph and thus does not allow effective control of spectral approximation levels, such as the spectral similarity. In other words, it is impossible to control the approximation quality or sparsity of the sparsified graph using a common set of input parameters. In contrast, our SPS allows precise control of spectral similarity or graph sparsity, thereby enabling effective trade-offs between approximation quality and graph complexity. Our latest extensive experiments carried out on a series of public-domain graphs show that it is almost impossible to compare the sparsified graphs obtained by using our SPS method and Spielman's approach due to the above reasons.

For multilevel graph drawing, we compare MNR against METIS [114], a fast and high-quality multilevel scheme for graph partitioning. The version of METIS provided by Karypis and Kumar [114] is used, where we set the number of clusters METIS should produce to the number of nodes of the equivalent level of MNR. We merge a cluster $i$ into a new node $i'$ and add an edge between two new nodes $i'$ and $j'$ if there exists an edge from any node in cluster $i$ to any node in cluster $j$. In order to make fair comparisons, we only use the graph after SES as input for METIS.

The graph data sets experimented are split into two groups: small data sets ($< 15,000$ nodes) and big data sets ($> 15,000$ nodes). This is due to the fact that DSS is not able to handle the big data sets on the machines we used. Given a data set, after edge sparsification, we produced five levels of node reduction for SPS and used the resulting numbers of edges and nodes as the targets to obtain the sparsification results for DSS (small data sets only) and the two variants of FF.

### 8.3.2 Sparsification Timings

Table 8.1 reports the timing results in seconds for graph sparsification. For SPS, $\mathrm{FF_E}$, and $\mathrm{FF_N}$, we show the computation time to achieve five different levels of sparsification. As the MNR step of SPS is an iterative algorithm, the results only show the time it takes from level $i$ (finer) to level $i+1$ (coarser), while the entries for either $\mathrm{FF_E}$ or $\mathrm{FF_N}$ always show the total computation time starting from the original graph. For DSS, only a single computation time is reported for each data set, as the algorithm computes the effective resistance for every edge and then uses a desired number of edges with the highest resistance values as the result. All the reported timing results were collected from runs on Lenovo NeXtScale nx360 M5 Servers with dual 12 core Intel Xeon CPU E5-2680 v3 @ 2.50GHz Haswell processors and 256GB RAM.

**Small data sets.** The upper part of Table 8.1 shows that DSS cannot keep up with the speed of the other algorithms. Even for the smallest data sets (Facebook and Airfoil), it already takes more than 10 minutes to compute the effective resistance value for the entire graph. In contrast, SPS and the two FF methods, always complete the computation under 20 seconds, with most of the cases below 10 seconds. When comparing SPS against $\mathrm{FF_E}$ and $\mathrm{FF_N}$, we can see that either $\mathrm{FF_E}$ or $\mathrm{FF_N}$ outperforms SPS for all the data sets, due to the time spent by SPS on SES. However, the performance gap decreases with increasing graph size.

**Big data sets.** The lower part of Table 8.1 shows the timing results for the bigger data sets. Besides the spectral sparsification, SPS stays consistent with its low computation time. On the contrary, the computation time for $\mathrm{FF_E}$ and $\mathrm{FF_N}$ drastically increases along with the input graph's size. The first three data sets (Vsp, Protein_DB, and Mesh), still show a similar timing performance for all three methods, due to the time spent by SPS on the SES step. After that, starting with CFD, the difference in computation time between SPS and the FF methods increases drastically to more than 10 folds (DBLP and ND), and about 70 folds (IL2010) at the finest level. At the coarsest level, however, the difference between SPS and $\mathrm{FF_E}$ vanishes for DBLP and ND, and decreases to about five folds for IL2010. At this sparse level, $\mathrm{FF_N}$ outperforms any method except for the IL2010 data set. This demonstrates the competitive advantage of our SPS method in terms of computational scalability.

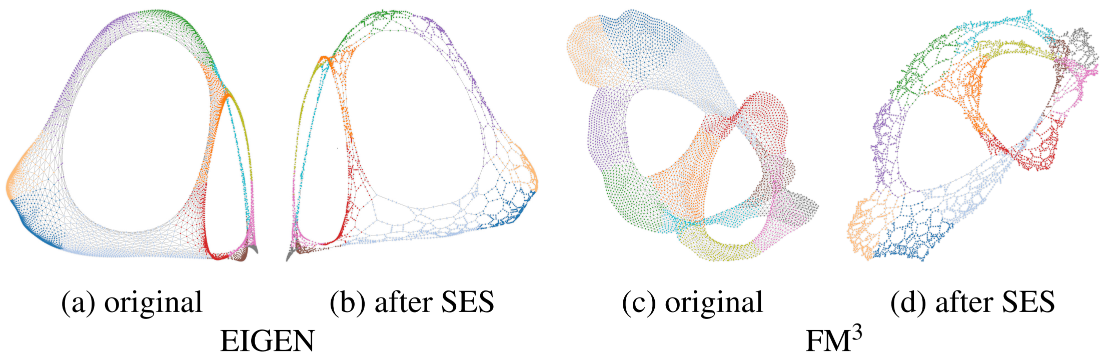


Figure 8.4. Graph drawings of the Airfoil data set using the EIGEN ((a) and (b)) and FM$^3$ ((c) and (d)) layouts. The drawings show the original graph ((a) and (c)) and the reduced graph after SES ((b) and (b)).

## 8.3.3 Graph Visualization

For graph drawing, we used the following methods: (1) EIGEN (refer to Chapter 8.2.3), (2) t-SNE (refer to Chapter 8.2.3), and (3) fast multipole multilevel method (FM$^3$) [81]. Note that we leveraged MATLAB for computing EIGEN and t-SNE, and OGDF [40] for computing FM$^3$. We chose FM$^3$, a force-directed layout for large graphs, because it has an efficient time complexity of $O(|N|\log|N|+|E|)$ and was recently applied to graph drawing with spectral sparification [58]. We did not draw the original graphs, but only their sparsified or sampled versions, as it is often not possible to draw the full-size graph due to the computational costs of EIGEN and t-SNE for large graphs. To circumvent the problem of not drawing the original big graph, we used the proxy quality metric [162] to evaluate the quality of the graph's proxy drawing. Our work demonstrates the capability of drawing graphs with spectral sparsification on data sets much larger than recently attempted by Eades et al. [58].

We implemented FFTEB to reduce visual clutter. Based on the spectral clustering result, we colored the nodes in different clusters with different colors. To allow easier visual comparison, for the FF and DSS sampling results, we kept the coloring based on the

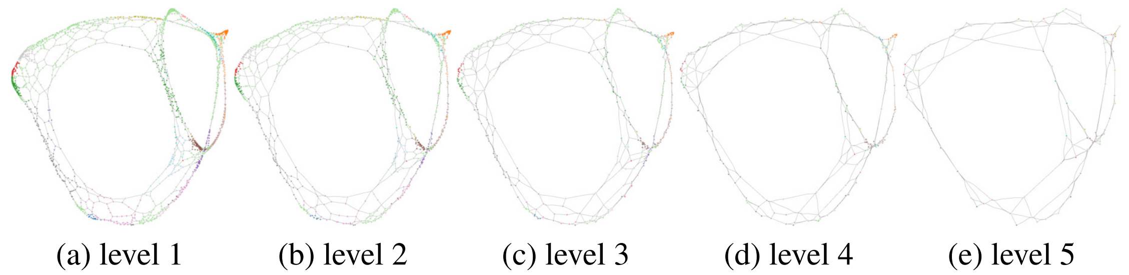


Figure 8.5. Graph drawings of the Airfoil data set using the EIGEN layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification using the SPS algorithm.

SPS clusters and used black for all the nodes that do not exist in the SPS results at the same sparsification level.

**Edge sparsification.** Figure 8.4 shows the Airfoil data set before and after SES. Ignoring the flip that occurred, we can see that in (a) and (b), the graph structure remains the same using the EIGEN layout, with (b) showing fewer edges. The drawings in (c) and (d) reveal the same using the FM$^3$ layout. This indicates that SES can successfully keep edges relevant for the graph structure while removing non-essential edges. Additionally, the spectral clusters are also well preserved in the drawing.

**Comparison across simplification levels.** In Figure 8.5, we compare the five levels of sparsification using the Airfoil data set using the EIGEN layout. We can see that although the number of nodes halves at each level of simplification, the overall graph structure remains the same. Even the coarsest level (Figure 8.5 (e)) shows the two big circle-like structures as the most distinguishable features of this data set. Similarly, the first two rows of Figure 8.6 show the graph drawings for the five sparsification levels of the CFD data set using the t-SNE layout. Again we can see that the structure from the layout at the fifth level is preserved through the multilevel eigensolver. For this much bigger and denser data set, we do not observe an almost bone-like structure like for the Airfoil one at the coarsest level. However, we can still witness how the number of nodes in each cluster reduces

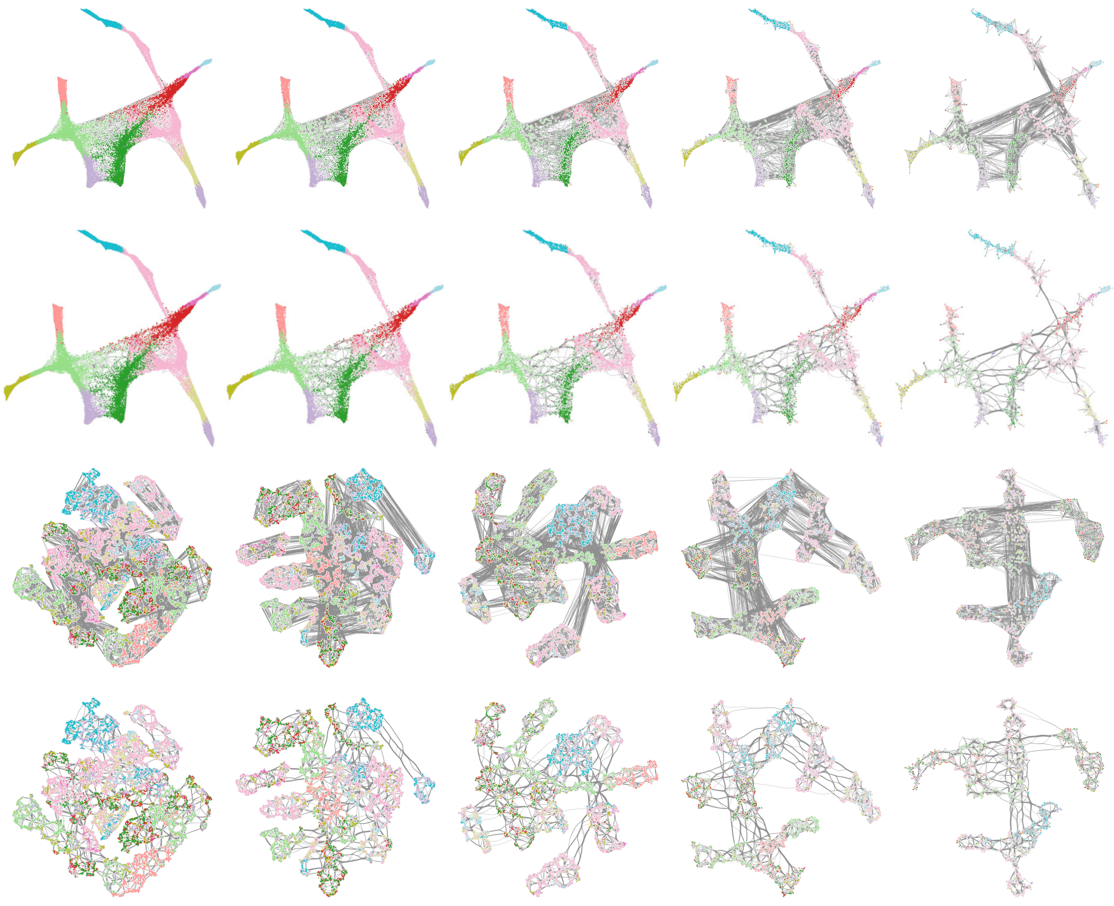
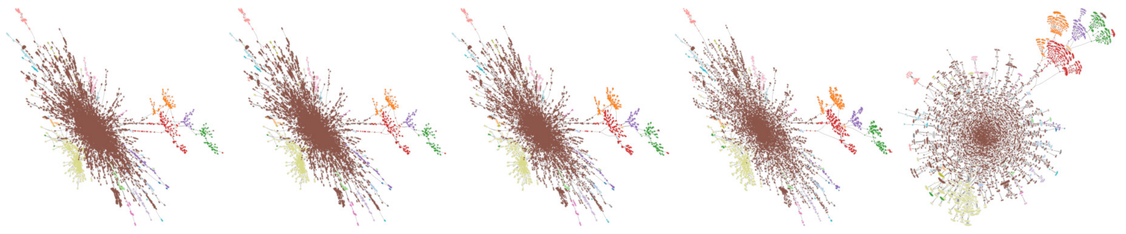


Figure 8.6. Graph drawings of the CFD data set using the t-SNE layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification. Top two rows: MNR. Bottom two rows: METIS. For either method, the upper or lower row shows the drawing without or with edge bundling.

successively between the neighboring levels without losing the inter-cluster connectivity. Figure 8.7 shows the drawing of the ND data set at its five sparsification levels using the FM$^3$ layout. This even bigger graph does not show much difference at the first four levels as the numbers of nodes at these four levels remain pretty high. At the last level (Figure 8.7 (e)), however, we can see a drastic skew in the layout. Although this represents a strong change in the layout, the graph features, especially the clusters, still remain easily



(a) level 1 (b) level 2 (c) level 3 (d) level 4 (e) level 5

Figure 8.7. Graph drawings of the ND data set using the FM$^3$ layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification using the SPS algorithm.

distinguishable. These three examples show how well the multilevel eigensolver allows using the layout from a coarser level and map it back to the original one without changing the overall graph structure.

**Comparison across sparsification methods.** Figure 8.8 shows a comparison of the three sparsification methods for the ND3k (top row) and Facebook (bottom row) data sets. We use the t-SNE layout and the third level of sparsification. In Figure 8.8 (a) and (c), we can see that the two spectrum-based methods do a better job at preserving the underlying graph structure compared to the $FF_E$ result shown in Figure 8.8 (b). The drawing of the $FF_E$ method seems rather random and contains a large number of small node clusters (shown in black) that do not exist in the SPS result. It is worth noting that the two spectrum-based methods mostly agree on the chosen nodes, while the $FF_E$ method contains many nodes that do not exist in the SPS variant. In the second row of Figure 8.8, we can see that the $FF_E$ method needs more nodes than the other two methods for the Facebook data set to achieve the desired number of edges. This shows that spectrum-based methods are better suited to give an overview of the most important nodes of the graph than the $FF_E$ sampling.

In Figure 8.9, we show similar comparisons for the Protein_DB and IL2010 data sets using the FM$^3$ layout. Protein_DB shows the finest level of sparsification while IL2010 shows the coarsest level. For the Protein_DB data set, we can see that the layout produced

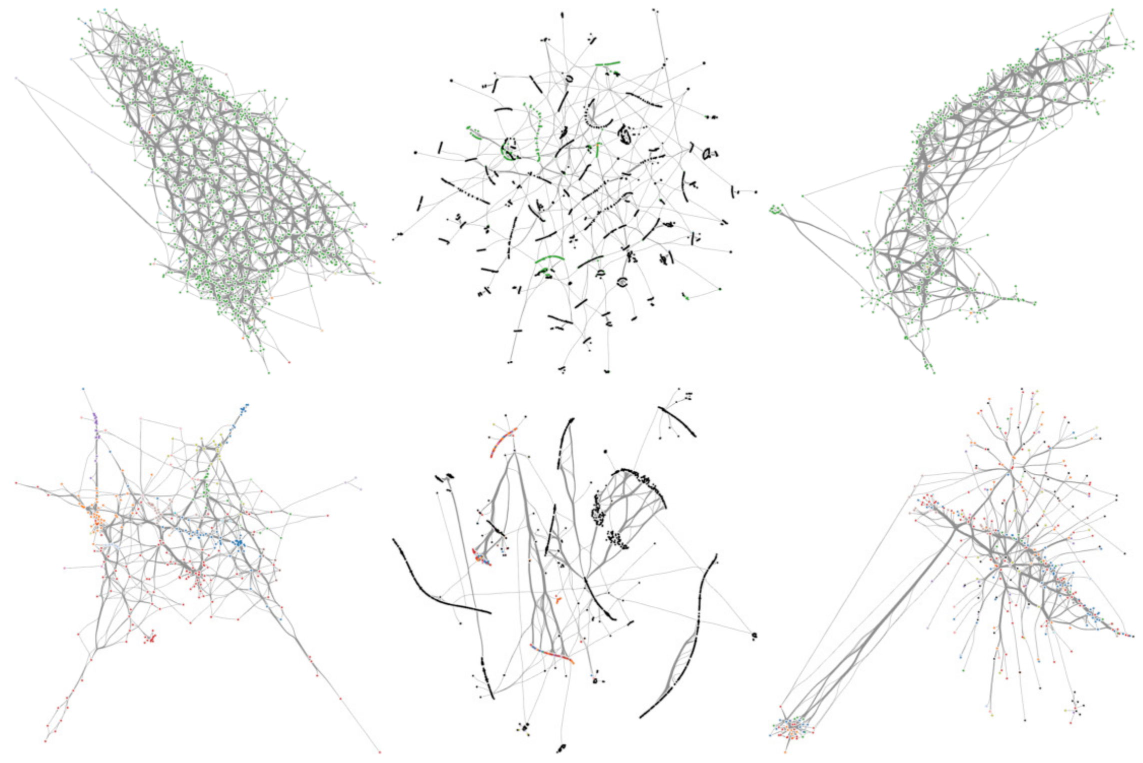

(a) SPS (b) $FF_E$ (c) DSS

Figure 8.8. Graph drawings of the different sampling methods for the ND3k (top row) and Facebook (bottom row) data set using the t-SNE layout. All drawings use the third level of sparsification.

by $FF_E$ mixes the clusters together, resulting in a confusing structure. The layout produced by SPS shows a much smoother and nicer cluster separation and a more revealing overall structure. When it comes to the IL2010 data set, $FF_E$ results in a tree-like graph, while SPS shows a more dispersed structure that looks similar to a flipped version of the underlying geographical map of the state of Illinois. Capturing and representing features like geographical and geometric structures underlines the advantages of SPS over random sampling methods.

Figure 8.10 shows the USPS10NN and Mesh data sets using the EIGEN (top row) and t-SNE (bottom row) layouts. USPS10NN uses the finest level of sparsification while Mesh

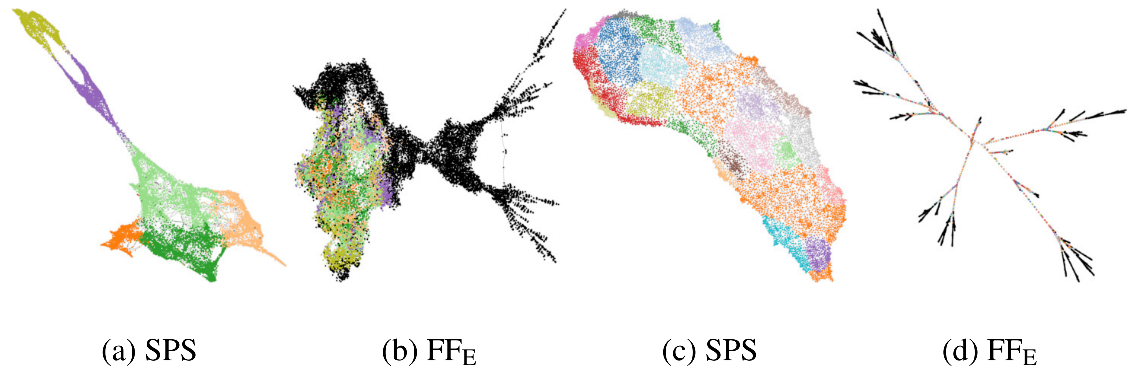

(a) SPS (b) $\text{FF}_\text{E}$ (c) SPS (d) $\text{FF}_\text{E}$

Figure 8.9. Graph drawings of the different sampling methods using the $\text{FM}^3$ layout. (a) and (b) are for the Protein_DB data set at the finest level of sparsification. (c) and (d) are for the IL2010 data set at the coarsest level of sparsification.

uses the coarsest level. For the USPS10NN data set, $\text{FF}_\text{E}$ finds one cluster instead of multiple ones like the SPS method. Thus the resulting drawing for the $\text{FF}_\text{E}$ sample is very dense and cluttered into one corner (EIGEN) or a hairball (t-SNE) instead of more evenly distributed like the drawing of SPS. For the Mesh data set, the drawing of the $\text{FF}_\text{E}$ sample again shows tree-like and hairball-like structures for the EIGEN and t-SNE layouts, respectively. The drawing of the SPS sample, on the other hand, highlights the grid-like structure of the underlying mesh in either layout. This shows that based on spectral analysis, SPS can reveal the underlying structures well at both the finest and coarsest levels.

**Comparison of MNR and METIS.** In Figure 8.11, we show a comparison of the MNR and METIS methods. For both drawings, we use the $\text{FM}^3$ layout and keep the cluster labels from SPS for easier comparison. Besides the different cluster ordering, there is no significant visual difference. Nevertheless, we point out that unlike METIS, MNR preserves the spectrum of the graph and does not require layout recomputation as we move from the coarsest level to the finest level. This can be seen in Figure 8.6, where we show the t-SNE layout for the five sparsification levels for MNR and METIS along with edge bundling disabled and enabled. We can see that the graph structure in the drawing is

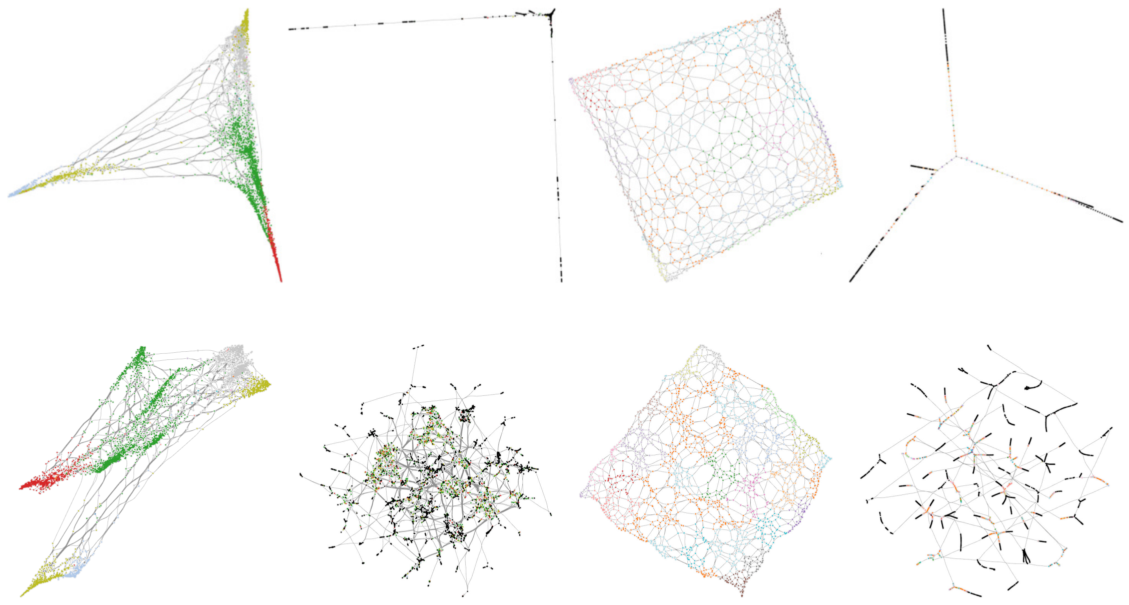


Figure 8.10. Graph drawings of the different sampling methods using the EIGEN (top row) and t-SNE (bottom row) layouts. (a) and (b) are for the USPS10NN data set at the finest level of sparsification. (c) and (d) are for the Mesh data set at the coarsest level of sparsification.

fairly consistent across the five levels with MNR, which is certainly not the case with METIS. Since the t-SNE layout is based on the leading eigenvectors resulting from SPS, we can claim that MNR better preserves the spectrum of the underlying graph. Furthermore, the MNR results show many shorter edges, indicating a well-translated structure from the reduced graph into the drawing.

For edge bundling, clearly, it helps to reduce visual clutter, especially for the five levels with METIS where edges are longer. However, edge bundling introduces ambiguities at the endpoints of thicker bundles.

**Visual quality of spectral graph.** We point out that spectral drawing of a graph may not necessarily lead to good visual quality. The general idea behind spectral graph drawing is to translate the spectral properties of the graph to the visualization. Prior works on graph

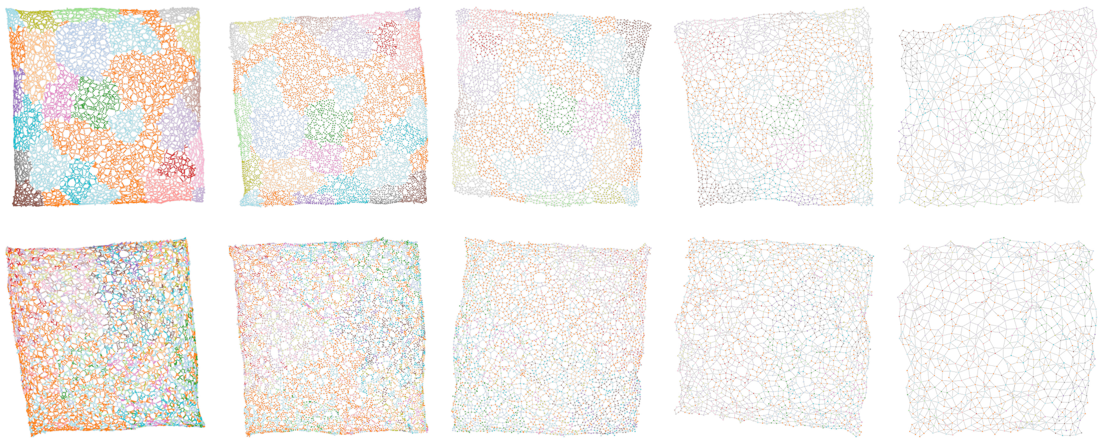


Figure 8.11. Graph drawings of the Mesh data set using the FM$^3$ layout. The drawings from left (finest) to right (coarsest) show the five levels of simplification. Top row: MNR. Bottom row: METIS.

drawing using spectral information [20, 89, 122, 123, 168] do not necessarily generate visually pleasing or aesthetic layouts either. Our observations are that spectral methods are good for drawing grid- or mesh-like graphs, but could be bad for other graphs. In those cases, the nodes in the spectral layout could overlap with each other (due to the great similarity of their spectral properties) or form a linearization pattern.

## 8.4 Quantitative Comparison

### 8.4.1 Quality Metrics

To evaluate the visual quality of graph samples, Nguyen et al. [162] introduced the proxy quality metric, which compares the drawing of a graph sample to the underlying graph in order to express the faithfulness of the drawing. This metric compares the similarity of each node in the drawing to the node in the underlying graph using one-to-one correspondence. The SPS algorithm, however, does not preserve such a correspondence due to the introduction of pseudo-nodes in MNR (refer to Chapter 8.2.2). Therefore, we use the proxy quality metric to compare the samples after SES but before MNR. We employ four other statistical metrics to quantify the sampling quality of MNR.

The proxy quality metric obtains a shape graph from the sampled graph drawing and then compares it to the original graph. Formally

$$Q_{\mu,\phi}(G,S(G)) = \mu(G,\phi(S(G))), \tag{8.14}$$

where $\mu$ is a comparison function that compares the two graphs and returns a real number, $\phi$ is a shape graph function, and $S(G)$ is a sample of the original graph $G$. Examples of shape graphs include the *$\alpha$-shape* [60], *k-nearest neighbor graph* (k-NN graph), *Gabriel graph*, *relative neighborhood graph* (RNG), and *Euclidean minimum spanning tree* (EMST). The similarity between two graphs of the same vertex set can be measured efficiently using the *mean Jaccard similarity* (MJS). In this work, we used the Gabriel graph as the shape graph function $\phi$ and the MJS as the comparison function $\mu$. The MJS between $S(G)$ and $G$ is defined as

$$\mathrm{MJS}(S(G),G) = \frac{1}{|N|}\sum_{v\in N}\frac{|N_{S(G)}(v)\cap N_G(v)|}{|N_{S(G)}(v)\cup N_G(v)|}, \tag{8.15}$$

where $N_{S(G)}(v)$ and $N_G(v)$ are the neighborhoods of node $v$ in $S(G)$ and $G$, respectively. For simplicity, we define $Q$(SPS), $Q$(DSS), and $Q$(FF$_\mathrm{E}$) for the MJS between the original graph and its sample with SPS, DSS, and FF$_\mathrm{E}$, respectively.

To evaluate the quality of the MNR samples, we use four of the five metrics used by Hong et al. [98]:

- *degree correlation assortativity* (DCA) which describes how well similar nodes are connected to each other [160];
- *closeness centrality* (CCe) which sums the lengths of the shortest paths from each node to all other nodes [66];

- *clustering coefficient* (CCo) which measures how well nodes cluster together within the graph [176];
- *average neighborhood degree* (AND) which averages the degrees of neighboring nodes for each node [9].

We do not use the fifth metric, largest connected component (LCC), as SPS and FF always yield a graph with a single connected component. We compare the metric on a given sample and the original graph using the Kolmogorov-Smirnov (KS) test. The KS-test computes the difference between two probability distributions and describes it as a result between 0 (same) and 1 (completely dissimilar).

TABLE 8.2

AVERAGED RESULTS OVER THREE LAYOUT ALGORITHMS OF THE QUANTITATIVE COMPARISON USING MJS

| data set | edges after SES | $Q$(SPS)/$Q$(FF$_E$) | $Q$(SPS)/$Q$(DSS) |
|---|---:|---:|---:|
| Facebook | 7,872 | 1.03 | 2.84 |
| Airfoil | 4,934 | 0.72 | 3.19 |
| ND3k | 16,745 | 1.73 | 50.33 |
| USPS10NN | 12,900 | 0.89 | 2.96 |
| Mycielskian14 | 14,060 | 1.34 | * |
| Appu | 16,366 | 2.34 | 3.65 |

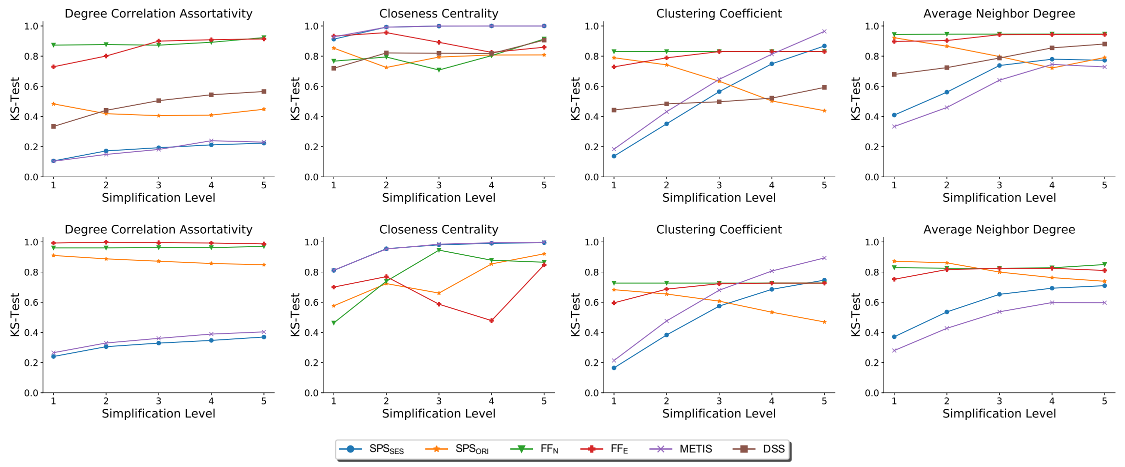
| data set | edges after SES | $Q$(SPS)/$Q$(FF$_E$) |
|---|---:|---:|
| Vsp | 42,752 | 2.30 |
| Protein_DB | 70,491 | 2.49 |
| Mesh | 45,261 | 66.71 |
| CFD | 106,879 | 1.64 |
| DBLP | 358,226 | 6.25 |
| ND | 388,436 | 2.85 |
| IL2010 | 504,465 | 7.60 |

Left table: small graphs. Right table: big graphs. The columns show the number of edges after SES and the quality ratios (higher is better). The * denotes that DSS does not achieve a MJS within the machine precision, i.e., it is very close to zero.

### 8.4.2 Comparison Results

**Proxy quality metric.** In Table 8.2, we report the averaged MJS ratios $Q$(SPS)/$Q$(DSS) and $Q$(SPS)/$Q$($\mathrm{FF_E}$) for the comparison between SPS and DSS, as well as SPS and $\mathrm{FF_E}$ respectively. Note that we only use $\mathrm{FF_E}$ here, as we only compare the results of SES, an edge-based sparsification technique. We use the t-SNE, EIGEN, and $\mathrm{FM}^3$ layouts for this comparison. The ratio values above 1.0 favor SPS over the comparing method. We can see that SPS generally achieves a better quality than DSS and $\mathrm{FF_E}$, with the exception of the Airfoil and USPS10NN data sets when compared to $\mathrm{FF_E}$. This is mainly because for these two data sets, $\mathrm{FF_E}$ sampling vastly outperforms SPS sampling with the t-SNE and $\mathrm{FM}^3$ layouts. Further worth mentioning are the high values of $Q$(SPS)/$Q$(DSS) for ND3k and $Q$(SPS)/$Q$($\mathrm{FF_E}$) for Mesh. These are due to the fact that SPS sampling vastly outperforms the sampling being compared across all three layouts. With these results, we conclude that the SES step of SPS preserves the structure of the original graph better than DSS and $\mathrm{FF_E}$.

**Sampling quality metrics.** Figure 8.12 shows the KS-test results between the original graph and a given sample for the four different metrics (lower KS-test values are better). In the charts, we can see that SPS (either $\mathrm{SPS_{SES}}$ or $\mathrm{SPS_{ORI}}$) generally outperforms the $\mathrm{FF_E}$ sampling methods, but there is no clear winner between SPS and DSS. METIS behaves very similar to $\mathrm{SPS_{SES}}$ in terms of DCA and CCe, while it performs better in terms of CCo and worse in terms of AND. While DCA remains mostly stable among all methods and sample sizes, the other metrics show interesting trends. CCe yields worse results for $\mathrm{SPS_{SES}}$ than $\mathrm{SPS_{ORI}}$. This means that the shortest path lengths after MNR are closer to the ones of the original graph than to those after SES. For the sake of argument, consider the average of the shortest path lengths for each node to all other nodes. If we compare the distribution of those average shortest path lengths (1) between the sampled graph and the original graph and (2) between the sampled graph and the graph after SES, then the difference between the sampled graph and the graph after SES will be smaller. This is because SES takes a graph as input and produces another graph that is similar to a spanning



(a) DCA (b) CCe (c) CCo (d) AND

Figure 8.12. Averaged KS-test values (lower is better) between the original graph and the sample. Top row: small graphs. Bottom row: big graphs. $\text{SPS}_{\text{SES}}$ denotes the comparison between MNR and the graph after SES. $\text{SPS}_{\text{ORI}}$ denotes the comparison between MNR and the original graph.

tree of the original graph. Evaluating the average of the shortest paths (for each node) in a spanning tree will be quite different from using the original graph.

Now if we consider the distribution of the average of shortest paths in a graph after applying the MNR procedure. As shown in Figure 8.3 (a), MNR reduces the graph through node aggregation: a pseudo-node at level $i+1$ represents multiple nodes at level $i$. Any edge between two nodes which are both represented by the same pseudo-node is removed. The pseudo-node becomes incident to any edge that connects two nodes of which only one of them is represented by the pseudo-node. As we aggregate nodes together and take the edges from all their original nodes, the graph at a coarser level is less similar to a spanning tree of the original graph. The impact of this is the opposite of what SES has on a graph. Therefore, comparing closeness centrality after MNR with respect to the original graph shows more similarity than that after SES.

For CCo, we see that typically after the third level of MNR, the $\text{SPS}_{\text{SES}}$ and $\text{SPS}_{\text{ORI}}$

lines cross. The reason for this is analog to what is discussed previously. The difference is that we consider between-cluster and within-cluster edges in the graph. Since MNR is applied after SES, it uses a spanning tree as input. Therefore at the finer levels, it is more like a spanning tree and less like the original graph, while at the coarser level, MNR produces a graph that is less like a spanning tree. The worse score for METIS in terms of CCo is due to the number of edges. Over all data sets and all simplification levels, METIS produces an average of 13% (20-70% for denser graphs, e.g., ND and Vsp, and less than 10% for sparser graphs, e.g., Airfoil, IL2010) more edges compared to MNR. As the input graph for this comparison is the graph after SES, i.e., a very sparse graph, the denser output can translate into a different CCo.

For AND, we can see that $\text{SPS}_{\text{SES}}$ and $\text{SPS}_{\text{ORI}}$ trend toward similar values the more iterations of SPS we run. This is because, with a more reduced graph, there are only the important nodes and their neighborhood relationships left to represent the original (sparsified) graph. Interestingly, METIS has an AND value more similar to the graph after SES than MNR. This shows that our MNR removes edges more aggressively to preserve spectral properties.

## 8.5 Conclusions

We have presented SPS, an effective solution for spectrum-preserving sparsification of big graphs. The innovation of SPS is that for the first time, it combines spectral graph sparsification to achieve scalable visualization of large graphs while allowing for spectral clustering analysis at the same time. Our SPS algorithm includes two steps: spectral edge sparsification (SES) followed by multilevel node reduction (MNR). The SES algorithm is three to four orders of magnitude faster than the state-of-the-art DSS algorithm. The dramatic gain in speed performance enables us to handle edge sparsification and subsequent node reduction on big graphs with hundreds of thousands of nodes and millions of edges, which was previously impossible. Furthermore, using different graph drawing layouts (EIGEN/t-SNE, FM$^3$), we find that in general, SPS outperforms DSS and FF under a proxy quality metric (for the SES step) and other statistical properties of the graphs (for the MNR step). We demonstrate the effectiveness of our approach using results gathered from a number of graph data sets of varying sizes and characteristics.

# CHAPTER 9

# CONCLUSIONS

In this dissertation, I explored several GPU-accelerated techniques to summarize and reconstruct big data in the area of scientific and information visualization. Besides simplifying data to a more manageable amount and summarizing them for visual representation, I also showed possible ways to identify and reconstruct missing information. Throughout this endeavor, I have used several techniques ranging from GPU-based parallelization and approximation, over iterative optimization, to deep learning techniques. Doing so, I approached different types of data to show how the used methods can help to generate insights into them without overwhelming a potential viewer.

For time-varying multivariate data, I presented an approach on how to approximate and simplify the isosurface selection. The presented algorithm avoids fully extracting isosurfaces. It thus omits computing unnecessary information during the evaluation of isosurfaces similarity to allow for a faster importance ranking. Based on said ranking, it is then possible to only fully extract the representative isosurfaces for a given volume. This procedure resulted in a speedup of up to $20\times$ without significant loss in quality. Further, it allowed subsequent works doing a more in-depth analysis of time-varying multivariate data. The first of these works aimed to overcome oversampling of unevenly spread value ranges in scientific volumetric data. I presented a two-stage iterative optimization algorithm to find nearly equally spaced isosurfaces for a given volume. The algorithm starts from an evenly distributed sample and first quickly converges to a rough solution. This solution avoids the problem that a part of the value range is oversampled while other, shorter ranges are undersampled. After that, a refinement procedure continues to optimize the solution until nearly equally spaced isosurfaces are found. The second of these works allowed us to create a complete overview of a time-varying multivariate data set. To do so, we need to analyze hundreds of time steps and tens of variables quickly. By using the acceleration technique, it was possible to extract representative surfaces across the whole time and variable domain and compare these among each other. Based on this analysis, an interface can be used to spot interesting developments to gain an overview and then further explore the data more thoroughly.

For vector field data, I presented a study that examines the strengths and weaknesses of three deep-learning methods for reconstructing missing time steps in unsteady flow data. Flow simulation data often can only be stored in coarse granularity, and thus reconstruction would be beneficial for post hoc analysis for scientists. I showed that, when splitting the three channels of VFD, it is possible to synthesize a set of vector fields between two others with high quality and accuracy for tracing streamlines and pathlines.

For graph data, I presented a framework that allows us to quickly reduce a big graph to a small, yet visually accurate representation while creating multiple levels of detail. The presented framework, SPS, first recursively down-samples a big graph to a manageable size before computing a layout. The computed layout is then mapped back from the smallest level in the same recursive manner. During the whole procedure, the spectrum of the graph is preserved with high accuracy, making it possible to have a drawing in every level representing the graph with high visual quality.

# BIBLIOGRAPHY

1. The 2015 Gordon Research Conference on Visualization in Science and Education. `https://www.grc.org/visualization-in-science-and-education-conference/2015/`. Accessed: 2019-10-16.

2. I. Abraham and O. Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of ACM Symposium on Theory of Computing*, pages 395–406, 2012.

3. H. Akiba and K.-L. Ma. A tri-space visualization interface for analyzing time-varying multivariate volume data. In *Proceedings of Eurographics - IEEE VGTC Symposium on Visualization*, pages 115–122, 2007.

4. H. Akiba, C. Wang, and K.-L. Ma. AniViz: A template-based animation tool for volume visualization. *IEEE Computer Graphics and Applications*, 30(5):61–71, 2010.

5. A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, F.-F. Li, and S. Savarese. Social LSTM: Human trajectory prediction in crowded spaces. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971, 2016.

6. D. Angus, A. Smith, and J. Wiles. Conceptual recurrence plots: Revealing patterns in human discourse. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):988–997, 2012.

7. C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *Proceedings of IEEE Visualization Conference*, pages 167–173, 1997.

8. J. Barnes and P. Hut. A hierarchical $O(n \log n)$ force-calculation algorithm. *Nature*, 324(6096):446, 1986.

9. A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences*, 101 (11):3747–3752, 2004.

10. M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *Proceedings of International AAAI Conference on Weblogs and Social Media*, 2009.

11. F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum*, 36(1):133–159, 2017.

12. M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 25(4):1636–1650, 2019.

13. J. Berry. Improving discrete mathematics and algorithms curricula with LINK. *ACM SIGCSE Bulletin*, 29(3):14–20, 1997.

14. A. Biswas, S. Dutta, H.-W. Shen, and J. Woodring. An information-aware framework for exploring multivariate data sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2683–2692, 2013.

15. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

16. J. M. Blondin and A. Mezzacappa. Pulsar spins from an instability in the accretion shock of supernovae. *Nature*, 445(7123):58–60, 2007.

17. M. Bostock, V. Ogievetsky, and J. Heer. D$^3$ data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

18. P. Bourke. Polygonising a scalar field. `http://paulbourke.net/geometry/polygonise/`, 1994. [Online; accessed 1-June-2017].

19. A. Brambilla, R. Carnecky, R. Peikert, I. Viola, and H. Hauser. Illustrative flow visualization: State of the art, trends and challenges. In *Eurographics State-of-the-Art Reports*, pages 75–94, 2012.

20. U. Brandes and T. Willhalm. Visualizing bibliographic networks with a reshaped landscape metaphor. In *Proceedings of Eurographics - IEEE TCVG Symposium on Visualization*, pages 159–164, 2002.

21. P.-T. Bremer, V. Pascucci, and B. Hamann. Maximizing adaptivity in hierarchical topological models. In *Proceedings of International Conference on Shape Modeling and Applications*, pages 298–307, 2005.

22. P.-T. 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.

23. W. L. Briggs, H. Van Emden, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, second edition, 2000.

24. S. Bruckner and T. Möller. Isosurface similarity maps. *Computer Graphics Forum*, 29(3):773–782, 2010.

25. G. H. Bryan and J. M. Fritsch. A benchmark simulation for moist nonhydrostatic numerical models. *Monthly Weather Review*, 130(12):2917–2928, 2002.

26. Y. Carbonneaux, J.-M. Laborde, and R. M. Madani. CABRI-Graph: A tool for research and teaching in graph theory. In *Proceedings of International Symposium on Graph Drawing*, pages 123–126, 1995.

27. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proceedings of ACM Symposium on Discrete Algorithms*, pages 918–926, 2000. ISBN 0-89871-453-2.

28. H. Carr, B. Duffy, and B. Denby. On histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1259–1266, 2006.

29. 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.

30. C.-S. Chang and S. Ku. Spontaneous rotation sources in a quiescent tokamak edge plasma. *Physics of Plasmas*, 15(6):062510, 2008.

31. C.-S. Chang, S. Ku, P. Diamond, Z. Lin, S. Parker, T. Hahm, and N. Samatova. Compressed ion temperature gradient turbulence in diverted tokamak edge. *Physics of Plasmas*, 16(5):056108, 2009.

32. C.-S. Chang, S. Ku, G. R. Tynan, R. Hager, R. M. Churchill, I. Cziegler, M. Greenwald, A. E. Hubbard, and J. W. Hughes. Fast low-to-high confinement mode bifurcation dynamics in a tokamak edge plasma gyrokinetic simulation. *Physical Review Letters*, 118(17):1–6, 2017.

33. C. Chen. Top 10 unsolved information visualization problems. *IEEE Computer Graphics and Applications*, 25(4):12–16, 2005.

34. C.-K. Chen, C. Wang, K.-L. Ma, and A. T. Wittenberg. Static correlation visualization for large time-varying volume data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 27–34, 2011.

35. J. Chen and I. Safro. Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, 33(6):3468–3490, 2011.

36. W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, 2011.

37. X. Chen and D. Cai. Large scale spectral clustering with landmark-based representation. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 313–318, 2011.

38. Y. Chen, J. Cohen, and J. Krolik. Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6): 1448–1455, 2007.

39. H.-C. Cheng, A. Cardone, S. Jain, E. Krokos, K. Narayan, S. Subramaniam, and A. Varshney. Deep-learning-assisted volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(2):1378–1391, 2019.

40. M. Chimani, K. Klein, S. Beyer, M. Gronemann, I. Hedtke, D. Kurz, S. Mahmalat, J. Schierbaum, J. M. Schmidt, S. Semper, M. H. Wagner, T. Wiedera, and B. Zey. Open Graph Drawing Framework (OGDF). `https://ogdf.uos.de/`, 2019.

41. P. Christiano, J. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of ACM Symposium on Theory of Computing*, pages 273–282, 2011.

42. R. M. Churchill, C. S. Chang, S. Ku, and J. Dominski. Pedestal and edge electrostatic turbulence characteristics from an XGC1 gyrokinetic simulation. *Plasma Physics and Controlled Fusion*, 59(10):105014, 2017.

43. Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3D U-Net: Learning dense volumetric segmentation from sparse annotation. In *Proceedings of International Conference on Medical Image Computing and Computer Assisted Interventions*, pages 424–432, 2016.

44. R. F. Cohen, A. Meacham, and J. Skaff. Teaching graphs to visually impaired students using an active auditory interface. *ACM SIGCSE Bulletin*, 38(1):279–282, 2006.

45. C. D. Correa, P. Lindström, and P.-T. Bremer. Topological spines: A structure-preserving visual representation of scalar fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1842–1851, 2011.

46. R. A. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proceedings of IEEE Visualization Conference*, pages 261–267, 1993.

47. W. M. Davis, M. K. Ko, R. J. Maqueda, A. L. Roquemore, F. Scotti, and S. J. Zweben. Fast 2-D camera control, data acquisition, and database techniques for edge studies on NSTX. *Fusion Engineering and Design*, 89(5):717–720, 2014.

48. M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

49. I. Demir, J. Kehrer, and R. Westermann. Screen-space silhouettes for visualizing ensembles of 3D isosurfaces. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 204–208, 2016.

50. J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

51. L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

52. D. D'Ippolito, J. Myra, and S. Zweben. Convective transport by intermittent blob-filaments: Comparison of theory and experiment. *Physics of Plasmas*, 18(6):060501, 2011.

53. U. Doğrusöz, B. Madden, and P. Madden. Circular layout in the graph layout toolkit. In *Proceedings of International Symposium on Graph Drawing*, pages 92–100, 1996.

54. C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

55. B. Duffy, H. Carr, and T. Möller. Integrating isosurface statistics and histograms. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):263–277, 2013.

56. T. Dwyer. Scalable, versatile and simple constrained graph layout. *Computer Graphics Forum*, 28(3):991–998, 2009.

57. P. Eades, S.-H. Hong, A. Nguyen, and K. Klein. Shape-based quality metrics for large graph visualization. *Journal of Graph Algorithms and Applications*, 21(1):29–53, 2017.

58. P. Eades, Q. H. Nguyen, and S.-H. Hong. Drawing big graphs using spectral sparsification. In *Proceedings of International Symposium on Graph Drawing*, pages 272–286, 2017.

59. M.-L. Eckert, K. Um, and N. Thuerey. ScalarFlow: A large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics*, 38(6):239:1–239:16, 2019.

60. H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–558, 1983.

61. M. Edmunds, R. S. Laramee, G. Chen, N. Max, E. Zhang, and C. Ware. Surface-based flow visualization. *Computers & Graphics*, 36(8):974–990, 2012.

62. M. Elkin, Y. Emek, D. A. Spielman, and N. Srivastava. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

63. J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. GraphViz - open source graph drawing tools. In *Proceedings of International Symposium on Graph Drawing*, pages 483–484, 2001.

64. Z. Feng. Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis. In *Proceedings of ACM/EDAC/IEEE Conference on Design Automation*, pages 1–6, 2016.

65. Z. Feng. Similarity-aware spectral sparsification by edge filtering. In *Proceedings of ACM/EDAC/IEEE Conference on Design Automation*, pages 152:1–152:6, 2018.

66. L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.

67. B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

68. S. Frey, F. Sadlo, and T. Ertl. Visualization of temporal similarity in field data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2023–2032, 2012.

69. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

70. W. Fung, R. Hariharan, N. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *Proceedings of ACM Symposium on Theory of Computing*, pages 71–80, 2011.

71. M. Glatter, J. Huang, S. Ahern, J. Daniel, and A. Lu. Visualizing temporal patterns in large multivariate data using textual pattern matching. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1467–1474, 2008.

72. M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, 2011.

73. G. Gouesbet and C. Letellier. Global vector-field reconstruction by using a multivariate polynomial L2 approximation on nets. *Physical Review E*, 49(6):4955–4972, 1994.

74. A. Graves, A.-R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

75. Y. Gu and C. Wang. A study of hierarchical correlation clustering for scientific volume data. In *Proceedings of International Symposium on Visual Computing*, pages 437–446, 2010.

76. Y. Gu and C. Wang. iTree: Exploring time-varying data using indexable tree. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 137–144, 2013.

77. Y. Gu and C. Wang. TransGraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2015–2024, 2011.

78. Y. Gu, C. Wang, T. Peterka, R. Jacob, and S. H. Kim. Mining graphs for understanding time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):965–974, 2016.

79. H. Guo, H. Xiao, and X. Yuan. Scalable multivariate volume visualization and analysis based on dimension projection and parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1397–1410, 2012.

80. L. Guo, S. Ye, J. Han, H. Zheng, H. Gao, D. Z. Chen, J.-X. Wang, and C. Wang. SSR-VFD: Spatial super-resolution for vector field data analysis and visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, 2020. Accepted.

81. S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proceedings of International Symposium on Graph Drawing*, pages 285–295, 2004.

82. S. Hadlak, H. Schumann, and H.-J. Schulz. A survey of multi-faceted graph visualization. In *Proceedings of Eurographics Conference on Visualization - State of The Art Reports*, pages 1–20, 2015.

83. M. Haidacher, S. Bruckner, and E. Gröller. Volume analysis using multimodal surface similarity. *IEEE Transactions on Visualization and Computer Graphics*, 17(12): 1969–1978, 2011.

84. K. M. Hall. An *r*-dimensional quadratic placement algorithm. *Management Science*, 17(3):219–229, 1970.

85. J. Han and C. Wang. TSR-TVD: Temporal super-resolution for time-varying data analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):205–215, 2020.

86. J. Han, J. Tao, H. Zheng, H. Guo, D. Z. Chen, and C. Wang. Flow field reduction via reconstructing vector data from 3D streamlines using deep learning. *IEEE Computer Graphics and Applications*, 39(4):54–67, 2019.

87. J. Han, J. Tao, and C. Wang. FlowNet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 26(4):1732–1744, 2020.

88. J. Hare, S. Samangooei, and D. Dupplaw. Open Intelligent Multimedia Analysis for Java (OpenIMAJ). `http://openimaj.org/`, 2019.

89. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *Proceedings of International Symposium on Graph Drawing*, pages 207–219, 2002.

90. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *Journal of Graph Algorithms and Applications*, 8(2):195–214, 2004.

91. K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

92. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

93. W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, M. Raj, Y. S. G. Nashed, and T. Peterka. InSituNet: Deep image synthesis for parameter space exploration of ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):23–33, 2020.

94. I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

95. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997.

96. D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5): 741–748, 2006.

97. F. Hong, J. Zhang, and X. Yuan. Access pattern learning with long short-term memory for parallel particle tracing. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 76–85, 2018.

98. S.-H. Hong, Q. Nguyen, A. Meidiana, J. Li, and P. Eades. BC tree-based proxy graphs for visualization of big graphs. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 11–20, 2018.

99. H. Hoppe. Progressive meshes. In *Proceedings of ACM SIGGRAPH Conference*, pages 99–108, 1996.

100. P. Hu and W. C. Lau. A survey and taxonomy of graph sampling. arXiv:1308.5865, 2013.

101. X. Hu, A. Lu, and X. Wu. Spectrum-based network visualization for topology analysis. *IEEE Computer Graphics and Applications*, 33(1):58–68, 2013.

102. Y. Hu. Efficient, high-quality force-directed graph drawing. *The Mathematica Journal*, 10(1):37–71, 2005.

103. M. Imre, J. Tao, and C. Wang. Efficient GPU-accelerated computation of isosurface similarity maps. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 180–184, 2017.

104. M. Imre, J. Tao, and C. Wang. Identifying nearly equally spaced isosurfaces for volumetric data sets. *Computers & Graphics*, 72:82–97, 2018.

105. M. Imre, J. Han, J. Dominski, M. Churchill, R. Kube, C.-S. Chang, T. Peterka, H. Guo, and C. Wang. ContourNet: Salient local contour identification for blob detection in plasma fusion simulation data. In *Proceedings of International Symposium on Visual Computing*, pages 289–301, 2019.

106. M. Imre, W. Chang, S. Wang, C. Trinter, and C. Wang. GraphVisual: Design and evaluation of a web-based visualization tool for teaching and learning graph visualization. In *Proceedings of American Society for Engineering Education Annual Conference*, 2020. Accepted.

107. M. Imre, J. Tao, Y. Wang, Z. Zhao, Z. Feng, and C. Wang. Spectrum-preserving sparsification for visualization of big graphs. *Computers & Graphics*, 87:89–102, 2020.

108. M. Jacomy, T. Venturini, S. Heymann, and M. Bastian. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PloS One*, 9(6):e98679, 2014.

109. H. Jänicke, M. Böttinger, and G. Scheuermann. Brushing of attribute clouds for the visualization of multivariate data. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1459–1466, 2008.

110. T. J. Jankun-Kelly and K.-L. Ma. A study of transfer function generation for time-varying volume data. In *Proceedings of Eurographics - IEEE TCVG Workshop on Volume Graphics*, pages 51–66, 2001.

111. H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz. Super SloMo: High quality estimation of multiple intermediate frames for video interpolation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 9000–9008, 2018.

112. A. Joshi and P. Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *Proceedings of IEEE Visualization Conference*, pages 679–686, 2005.

113. T. Karras. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proceedings of ACM SIGGRAPH/Eurographics Conference on High-Performance Graphics*, pages 33–37, 2012.

114. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

115. J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(3): 495–513, 2013.

116. S. Khuri and K. Holzapfel. EVEGA: An educational visualization environment for graph algorithms. *ACM SIGCSE Bulletin*, 33(3):101–104, 2001.

117. B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. Deep Fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70, 2019.

118. D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference for Learning Representations*, 2015.

119. P. Kolev and K. Mehlhorn. A note on spectral clustering. arXiv:1509.09188, 2015.

120. A. Kolla, Y. Makarychev, A. Saberi, and S.-H. Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. In *Proceedings of ACM Symposium on Theory of Computing*, pages 57–66, 2010.

121. Y. Koren. Drawing graphs by eigenvectors: Theory and practice. *Computers & Mathematics with Applications*, 49(11-12):1867–1888, 2005.

122. Y. Koren. On spectral graph drawing. In *Proceedings of International Conference on Computing and Combinatorics*, pages 496–508, 2003.

123. Y. Koren, L. Carmel, and D. Harel. ACE: A fast multiscale eigenvectors computation for drawing huge graphs. In *Proceedings of IEEE Symposium on Information Visualization*, pages 137–144, 2002.

124. I. Koutis, G. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 235–244, 2010.

125. I. Koutis, G. Miller, and D. Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011.

126. A. Krishnamurthy, S. McMains, and K. Haller. GPU-accelerated minimum distance and clearance queries. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):729–742, 2011.

127. R. Kube, O. E. Garcia, B. LaBombard, J. Terry, and S. Zweben. Blob sizes and velocities in the alcator c-mod scrape-off layer. *Journal of Nuclear Materials*, 438: S505–S508, 2013.

128. M. Lage, F. Petronetto, A. Paiva, H. Lopes, T. Lewiner, and G. Tavares. Vector field reconstruction from sparse samples with applications. In *Proceedings of Brazilian Symposium on Computer Graphics and Image Processing*, pages 297–306, 2006.

129. O. D. Lampe, C. Correa, K.-L. Ma, and H. Hauser. Curve-centric volume reformation for comparative visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1235–1242, 2009.

130. R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–222, 2004.

131. R. S. Laramee, H. Hauser, L. Zhao, and F. H. Post. Topology-based flow visualization, the state of the art. In H. Hauser, H. Hagen, and H. Theisel, editors, *Topology-Based Methods in Visualization*, chapter 1, pages 1–19. Springer Berlin Heidelberg, 2007.

132. E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1–8, 2000.

133. J. R. Lee, S. O. Gharan, and L. Trevisan. Multiway spectral partitioning and higher-order Cheeger inequalities. *Journal of the ACM*, 61(6):37:1–37:30, 2014.

134. R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics, 1998.

135. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diamaters and possible explanations. In *Proceedings of ACM SIGKDD Conference*, pages 177–187, 2005.

136. C. Letellier, L. Le Sceller, P. Dutertre, G. Gouesbet, Z. Fei, and J. Hudson. Topological characterization and global vector field reconstruction of an experimental electrochemical system. *The Journal of Physical Chemistry*, 99(18):7016–7027, 1995.

137. A. Lhuillier, C. Hurter, and A. Telea. FFTEB: Edge bundling of huge graphs by the fast Fourier transform. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 190–199, 2017.

138. F. Liu and Y. J. Kim. Exact and adaptive signed distance fields computation for rigid and deformable models on GPUs. *IEEE Transactions on Visualization and Computer Graphics*, 20(5):714–725, 2014.

139. J. Liu, C. Wang, M. Danilevsky, and J. Han. Large-scale spectral clustering on graphs. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 1486–1492, 2013.

140. X. Liu and H.-W. Shen. Association analysis for visual exploration of multivariate scientific data sets. *IEEE Transactions on Visualization and Computer Graphics*, 22 (1):955–964, 2016.

141. O. Livne and A. Brandt. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

142. S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

143. J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

144. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH Conference*, pages 163–169, 1987.

145. A. Lu and H.-W. Shen. Interactive storyboard for overall time-varying data visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 143–150, 2008.

146. B. Ma, S. K. Suter, and A. Entezari. Quality assessment of volume compression approaches using isovalue clustering. *Computers & Graphics*, 63:18–27, 2017.

147. K.-L. Ma. Visualizing time-varying volume data. *IEEE Computing in Science and Engineering*, 5(2):34–42, 2003.

148. P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 89–94, 2015.

149. M. M. Malik, C. Heinzl, and M. E. Gröller. Comparative visualization for parameter studies of dataset series. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):829–840, 2010.

150. S. Martin, W. M. Brown, R. Klavans, and K. W. Boyack. OpenOrd: An open-source toolbox for large graph layout. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*, page 786806, 2011.

151. N. Marwan, J. Kurths, and P. Saparin. Generalised recurrence plot analysis for spatial data. *Physics Letters A*, 360(4-5):545–551, 2007.

152. R. McKinley, R. Wepfer, T. Gundersen, F. Wagner, A. Chan, R. Wiest, and M. Reyes. Nabla-net: A deep dag-like convolutional architecture for biomedical image segmentation. In *Proceedings of International Workshop on Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 119–128, 2016.

153. T. Mcloughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based geometric flow visualization. *Computer Graphics Forum*, 29(6): 1807–1829, 2010.

154. M. Meyer, C. E. Scheidegger, J. M. Schreiner, B. Duffy, H. Carr, and C. T. Silva. Revisiting histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1659–1666, 2008.

155. F. Milletari, N. Navab, and S.-A. Ahmadi. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. In *Proceedings of International Conference on 3D Vision*, pages 565–571, 2016.

156. B. Moberts, A. Vilanova, and J. J. van Wijk. Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proceedings of IEEE Visualization Conference*, pages 65–72, 2005.

157. M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

158. F. A. Mussa-Ivaldi. From basis functions to basis fields: Vector field approximation from sparse data. *Biological Cybernetics*, 67(6):479–489, 1992.

159. V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of International Conference on Machine Learning*, pages 807–814, 2010.

160. M. E. Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

161. A.-D. Nguyen, W. Kim, J. Kim, and S. Lee. Video frame interpolation by plug-and-play deep locally linear embedding. *arXiv preprint arXiv:1807.01462*, 2018.

162. Q. H. Nguyen, S.-H. Hong, P. Eades, and A. Meidiana. Proxy graph: Visual quality metrics of big graph sampling. *IEEE Transactions on Visualization and Computer Graphics*, 23(6):1600–1611, 2017.

163. S. Niklaus, L. Mai, and F. Liu. Video frame interpolation via adaptive convolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 670–679, 2017.

164. R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of IEEE International Conference on Machine Learning*, pages 1310–1318, 2013.

165. V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli. Multi-resolution computation and presentation of contour trees. In *Proceedings of IASTED Conference on Visualization, Imaging, and Image Processing*, pages 452–290, 2004.

166. V. Pekar, R. Wiemker, and D. Hempel. Fast detection of meaningful isosurfaces for volume data visualization. In *Proceedings of IEEE Visualization Conference*, pages 223–230, 2001.

167. R. Peng, H. Sun, and L. Zanetti. Partitioning well-clustered graphs: Spectral clustering works! *Proceedings of Machine Learning Research*, 40:1423–1455, 2015.

168. T. Pisanski and J. Shawe-Taylor. Characterizing graph drawing with eigenvectors. *Journal of Chemical Information and Computer Sciences*, 40(3):567–571, 2000.

169. W. P. Porter, Y. Xing, B. R. von Ohlen, J. Han, and C. Wang. A deep learning approach to selecting representative time steps for time-varying multivariate data. In *Proceedings of IEEE Conference on Visualization (Short Papers)*, pages 131–135, 2019.

170. L. Prantl, B. Bonev, and N. Thuerey. Generating liquid simulations with deformation-aware neural networks. In *Proceedings of International Conference for Learning Representations*, 2019.

171. M. P. Rast. The scales of granulation, mesogranulation, and supergranulation. *The Astrophysical Journal*, 597(2):1200–1210, 2003.

172. K. Rohe, S. Chatterjee, and B. Yu. Spectral clustering and the high-dimensional stochastic blockmodel. *The Annals of Statistics*, 39(4):1878–1915, 2011.

173. O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of International Conference on Medical Image Computing and Computer Assisted Interventions*, pages 234–241, 2015.

174. O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

175. T. Salzbrunn, H. Jänicke, T. Wischgoll, and G. Scheuermann. The state of the art in flow visualization: Partition-based techniques. In *Proceedings of Simulation and Visualization Conference*, pages 75–92, 2008.

176. J. Saramäki, M. Kivelä, J.-P. Onnela, K. Kaski, and J. Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2): 027105, 2007.

177. V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *Proceedings of ACM SIGMOD Conference*, pages 721–732, 2011.

178. N. Sauber, H. Theisel, and H.-P. Seidel. Multifield-Graphs: An approach to visualizing correlations in multifield scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):917–924, 2006.

179. C. E. Scheidegger, J. M. Schreiner, B. Duffy, H. Carr, and C. T. Silva. Revisiting histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1659–1666, 2008.

180. J. Schmidt, C. Heinzl, and S. Bruckner. VAICo: Visual analysis for image comparison. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2090–2099, 2013.

181. D. Schneider, A. Wiebel, H. Carr, M. Hlawitschka, and G. Scheuermann. Interactive comparison of scalar fields based on largest contours with applications to flow

visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6): 1475–1482, 2008.

182. J. Seitzer. Parallel computation of the Euclidean distance transform on a three-dimensional image array. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):203–212, 2003.

183. M. Seyedhosseini, M. Sajjadi, and T. Tasdizen. Image segmentation with cascaded hierarchical models and logistic disjunctive normal networks. In *Proceedings of IEEE International Conference on Computer Vision*, pages 2168–2175, 2013.

184. H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *Proceedings of IEEE Visualization Conference*, pages 371–377, 1999.

185. N. Shi and Y. Tao. CNNs based viewpoint estimation for volume visualization. *ACM Transactions on Intelligent Systems and Technology*, 10(3):27:1–27:22, 2019.

186. X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Proceedings of Advances in Neural Information Processing Systems*, pages 802–810, 2015.

187. D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing*, 30(3):83–98, 2013.

188. D. A. Spielman. Algorithms, graph theory, and linear qquations in Laplacian matrices. In *Proceedings of International Congress of Mathematicians*, volume 4, pages 2698–2722, 2010.

189. D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

190. D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of ACM Symposium on Theory of Computing*, pages 81–90, 2004.

191. D. A. Spielman and S.-H. Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

192. D. A. Spielman and S.-H. Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Application*, 35(3):835–885, 2014.

193. D. A. Spielman and J. Woo. A note on preconditioning by low-stretch spanning trees. arXiv:0903.2816, 2009.

194. Spielman, D. A. Laplacians.jl: Algorithms inspired by graph Laplacians: linear equation solvers, sparsification, clustering, optimization, etc. https://github.com/danspielman/Laplacians.jl, 2019.

195. G. W. Stewart. A Krylov-Schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23(3):601–614, 2002.

196. K. Stockinger, J. Shalf, K. Wu, and E. W. Bethel. Query-driven visualization of large data sets. In *Proceedings of IEEE Visualization Conference*, pages 167–174, 2005.

197. J. Sukharev, C. Wang, K.-L. Ma, and A. T. Wittenberg. Correlation study of time-varying multivariate climate data sets. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 161–168, 2009.

198. Y. Tanahashi, N. Leaf, and K.-L. Ma. A study on designing effective introductory materials for information visualization. *Computer Graphics Forum*, 35(7):117–126, 2016.

199. J. Tao, J. Ma, C. Wang, and C.-K. Shene. A unified approach to streamline selection and viewpoint selection for 3D flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):393–406, 2013.

200. J. Tao, M. Imre, C. Wang, N. V. Chawla, H. Guo, G. Sever, and S. H. Kim. Exploring time-varying multivariate volume data using matrix of isosurface similarity maps. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1236–1245, 2019.

201. S.-H. Teng. Scalable algorithms for data and network analysis. *Foundations and Trends in Theoretical Computer Science*, 12(1-2):1–274, 2016.

202. S. Tenginakai, J. Lee, and R. Machiraju. Salient iso-surface detection with model-independent statistical signatures. In *Proceedings of IEEE Visualization Conference*, pages 231–238, 2001.

203. S. K. Thomson. *Sampling*. Wiley-Interscience, second edition, 2002.

204. L. J. P. van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15:1–21, 2014.

205. L. J. P. van der Maaten and G. E. Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

206. S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequence-video to text. In *Proceedings of IEEE International Conference on Computer Vision*, pages 4534–4542, 2015.

207. V. Verma and A. T. Pang. Comparative flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):609–624, 2004.

208. 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 and Computer Graphics*, 14(6):1396–1403, 2008.

209. C. Wang. Graph-based techniques for visual analytics of scientific data sets. *IEEE Computing in Science & Engineering*, 20(1):93–103, 2018.

210. C. Wang and H.-W. Shen. A framework for rendering large time-varying data using wavelet-based time-space partitioning (WTSP) tree. Technical Report OSU-CISRC-1/04-TR05, Department of Computer and Information Science, The Ohio State University, 2004.

211. C. Wang and J. Tao. Graphs in scientific visualization: A survey. *Computer Graphics Forum*, 36(1):263–287, 2017.

212. C. Wang, J. Gao, L. Li, and H.-W. Shen. A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Proceedings of Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 11–19, 2005.

213. C. Wang, H. Yu, and K.-L. Ma. Importance-driven time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1547–1554, 2008.

214. C. Wang, H. Yu, and K.-L. Ma. Application-driven compression for visualizing large-scale time-varying data. *IEEE Computer Graphics and Applications*, 30(1):59–69, 2010.

215. C. Wang, H. Yu, R. W. Grout, K.-L. Ma, and J. H. Chen. Analyzing information transfer in time-varying multivariate data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 99–106, 2011.

216. G. Wang, M. Zuluaga, W. Li, R. Pratt, P. Patel, M. Aertsen, T. Doel, A. David, J. Deprest, S. Ourselin, et al. DeepIGeoS: A deep interactive geodesic framework for medical image segmentation. *arXiv preprint arXiv:1707.00652*, 2017.

217. G. Wang, W. Li, M. A. Zuluaga, R. Pratt, P. A. Patel, M. Aertsen, T. Doel, A. L. David, J. Deprest, S. Ourselin, et al. Interactive medical image segmentation using deep learning with image-specific fine tuning. *IEEE Transactions on Medical Imaging*, 37(7):1562–1573, 2018.

218. M. Wang, J. Tao, C. Wang, C.-K. Shene, and S. H. Kim. FlowVisual: Design and evaluation of a visualization tool for teaching 2D flow field concepts. In *Proceedings of American Society for Engineering Education Annual Conference*, pages 23.609.1–23.609.20, 2013.

219. M. Wang, J. Tao, J. Ma, Y. Shen, and C. Wang. FlowVisual: A visualization app for teaching and understanding 3D flow field concepts. In *Proceedings of IS&T Conference on Visualization and Data Analysis*, pages 476–1–476–10, 2016.

220. T.-H. Wei, T.-Y. Lee, and H.-W. Shen. Evaluating isosurfaces with level-set-based information maps. *Computer Graphics Forum*, 32(3):1–10, 2013.

221. S. Weiss, M. Chu, N. Thuerey, and R. Westermann. Volumetric isosurface rendering with deep learning-based super-resolution. *IEEE Transactions on Visualization and Computer Graphics*, 2019. Accepted.

222. C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Proceedings of Advances in Neural Information Processing Systems*, pages 682–688, 2001.

223. J. Woodring and H.-W. Shen. Multi-variate, time-varying, and comparative visualization with contextual cues. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):909–916, 2006.

224. J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumetric data. In *Proceedings of IEEE Visualization Conference*, pages 417–424, 2003.

225. L. Wu, K. J. Wu, A. Sim, M. Churchill, J. Y. Choi, A. Stathopoulos, C.-S. Chang, and S. Klasky. Towards real-time detection and tracking of spatio-temporal features : Blob-filaments in fusion plasma. *IEEE Transactions on Big Data*, 2(3):262–275, 2016.

226. Y. Wu, N. Cao, D. Archambault, Q. Shen, H. Qu, and W. Cui. Evaluation of graph sampling: A visualization perspective. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):401–410, 2017.

227. C. Xu and J. L. Prince. Gradient vector flow: A new external force for snakes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 66–71, 1997.

228. L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6): 1216–1224, 2010.

229. D. Yan, L. Huang, and M. I. Jordan. Fast approximate spectral clustering. In *Proceedings of ACM SIGKDD Conference*, pages 907–916, 2009.

230. J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz. Real-time concurrent linked list construction on the GPU. *Computer Graphics Forum*, 29(4):1297–1304, 2010.

231. A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le. QANet: Combining local convolution with global self-attention for reading comprehension. In *Proceedings of International Conference for Learning Representations*, 2018.

232. H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications*, 30 (3):45–57, 2010.

233. H. Yu, J. Xie, K.-L. Ma, H. Kolla, and J. H. Chen. Scalable parallel distance field construction for large-scale applications. *IEEE Transactions on Visualization and Computer Graphics*, 21(10):1187–1200, 2015.

234. F. Zhang, S. Zhang, P. C. Wong, J. E. Swan, and T. Jankun-Kelly. A visual and statistical benchmark for graph sampling methods. In *Proceedings of IEEE VIS Workshop on Exploring Graphs at Scale*, 2015.

235. K. Zhang, I. W. Tsang, and J. T. Kwok. Improved Nyström low-rank approximation and error analysis. In *Proceedings of International Conference on Machine Learning*, pages 1232–1239, 2008.

236. Z. Zhao, Y. Wang, and Z. Feng. SAMG: Sparsified graph theoretic algebraic multigrid for solving large symmetric diagonally dominant (SDD) matrices. In *Proceedings of ACM/IEEE International Conference on Computer-Aided Design*, pages 601–606, 2017.

237. Z. Zhao, Y. Wang, and Z. Feng. Nearly-linear time spectral graph reduction for scalable graph partitioning and data visualization. *arXiv preprint arXiv:1812.08942*, 2018.

238. Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. Volume upscaling with convolutional neural networks. In *Proceedings of Computer Graphics International*, pages 38:1–38:6, 2017.

239. S. Zweben, W. Davis, S. Kaye, J. Myra, R. Bell, B. LeBlanc, R. Maqueda, T. Munsat, S. Sabbagh, Y. Sechrest, D. Stotler, and the NSTX Team. Edge and SOL turbulence and blob variations over a large database in NSTX. *Nuclear Fusion*, 55(9):093035, 2015.

*This document was prepared & typeset with* pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2017.2[2017/05/09])*