Distribution-based Exploration and Visualization of Large-scale Vector and Multivariate Fields

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Kewei Lu, B.S., M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2017

Dissertation Committee:

Han-Wei Shen, Advisor

Yusu Wang

Ponnuswamy Sadayappan

Abstract

Due to the ever increasing of computing power in the last few decades, the size of scientific data produced by various scientific simulations has been growing rapidly. As a result, effective techniques to visualize and explore those large-scale scientific data are becoming more and more important in understanding the data. However, for data at such a large scale, effective analysis and visualization is a non-trivial task due to several reasons. First, it is often time consuming and memory intensive to perform visualization and analysis directly on the original data. Second, as the data become large and complex, visualization usually suffers from visual cluttering and occlusion, which makes it difficult for users to understand the data.

In order to address the aforementioned challenges, in this dissertation, a distribution-based query-driven framework to visualize and analyze large-scale scientific data is proposed. We propose to use statistical distributions to summarize large-scale data sets. The summarized data is then used to substitute the original data to support efficient and interactive query-driven visualization which is often free of occlusion. In this dissertation, the proposed framework is applied to flow fields and multivariate scalar fields.

We first demonstrate the application of the proposed framework to flow fields. For a flow field, the statistical data summarization is computed from geometries such as streamlines and stream surfaces computed from the flow field. Stream surfaces and streamlines are two popular methods for visualizing flow fields. When the data size is large, distributed

memory parallelism usually is needed. While several parallel streamline computation algorithms exist, relatively little research has been done to parallelize stream surface generation. This is because load-balanced parallel stream surface computation is non-trivial, due to the strong dependency in computing the positions of the particles forming the stream surface front. In this dissertation, a new scalable algorithm is proposed to compute stream surfaces from large-scale flow fields efficiently on distributed memory machines. After we obtain a large number of computed streamlines or stream surfaces, a direct visualization of all the densely computed geometries is seldom useful due to visual cluttering and occlusion. To solve the visual cluttering problem, a distribution-based query-driven framework to explore those densely computed streamlines is presented. The proposed framework is based on the observation that statistical distributions of measurements along the trajectory of a streamline can be used as a robust and effective descriptor to measure the similarity between streamlines.

Then, the proposed framework is applied to multivariate scalar fields. When dealing with multivariate data sets, in order to understand the data, it is often useful to show the regions of interest based on user specified criteria. In the presence of large-scale multivariate data, efficient techniques to summarize the data and answer users' queries are needed. In this dissertation, we first propose to use multivariate histograms to summarize the data and demonstrate how effective query-driven visualization can be achieved based on those multivariate histograms. However, storing multivariate histograms in the form of multi-dimensional arrays is very expensive as the size of the histogram grows exponentially with the number of variables. To enable efficient visualization and exploration of multivariate data sets, we present a compact structure to store multivariate histograms to reduce their huge space cost while supporting different kinds of histogram query operations efficiently.

We also present an interactive system to assist users to effectively design multivariate transfer functions. Multiple regions of interest could be highlighted through multivariate volume rendering based on the user specified multivariate transfer function.

To the memory of my father

Acknowledgments

During my Ph.D. studies, there are so many people I would like to thank for their support and help. The doctoral dissertation would not have been possible to complete without their support and help. To only some of them, it is possible to give particular mention here.

First of all, I would like to express my sincere gratitude to my advisor Dr. Han-Wei Shen for his encouragement, patience, and professional guidance during my Ph.D. studies at The Ohio State University. I am grateful to join The Graphics and Visualization Study (GRAVITY) research group and begin my Ph.D. under the guidance of Dr. Han-Wei Shen. His encouragement, patience, and professional knowledge have guided me through various difficulties and challenges.

I would also thank Dr. Yusu Wang and Dr. Ponnuswamy Sadayappan for serving on my dissertation committee and providing insightful advice.

My sincere thanks also goes to my internship mentors, Pak Chung Wong from Pacific Northwest National Laboratory, Tom Peterka from Argonne National Laboratory, and Christopher Sewell and Li-Ta Lo from Los Alamos National Laboratory, for their valuable guidance and advice.

I would like to extend my thanks to the past and present members of my lab. When I initially joined Dr. Han-Wei Shen's group, I learned a lot from senior students including Teng-Yok Lee, Abon Chaudhuri and Boonthanome Nouanesengsy. I also would like to

thank to my lab colleagues, including, but not limited to, Chun-Ming Chen, Xin Tong, Xiaotong Liu, Tzu-Hsuan Wei, Ayan Biswas, Soumya Dutta, Wenbin He, Cheng Li, Ko-Chih Wang, Subhashis Hazarika and Junpeng Wang. I enjoy the time working with them.

Finally, I would like to thank my parents and all the rest of my family in China for their unqualified love and support. I pay my deepest respect to my mother for her love, encouragement and support, and to my late father who will always be in my heart. I have no words to express my love and gratitude to my parents. This dissertation would not existed without their love, encouragement and support,

Vita

October 9, 1987	Born - Luoyang, China
2009	B.S. Computer Science and Technology, Wuhan University of Technology, China
2011	M.S. Computer Science and Engineering, The Ohio State University, USA
2010 - 2011	Student Assistant, The Ohio State University
2011 - Present	Graduate Research Associate, The Ohio State University
June - August, 2011	Research Intern, Pacific Northwest National Laboratory
June - August, 2012	Research Intern, Pacific Northwest National Laboratory
May - July, 2013	Research Intern, Argonne National Laboratory
January - April, 2015	Graduate Teach Associate, The Ohio State University
May - July, 2015	Research Intern, Los Alamos National Laboratory

Publications

Research Publications

Kewei Lu and Han-Wei Shen, "A Compact Multivariate Histogram Representation for Query-driven Visualization". In *Proceedings of the 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*, 25-26 Nov. 2015

Kewei Lu, Han-Wei Shen and Tom Peterka, "Scalable Computation of Stream Surfaces on Large Scale Vector Fields". In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*, pp.1008,1019, 16-21 Nov. 2014.

Pak Chung Wong, Han-Wei Shen, Ruby Leung, Samson Hagos, Teng-Yok Lee, Xin Tong and Kewei Lu, "Visual analytics of large-scale climate model data". In *Proceedings of the 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)*, pp.85,92, 9-10 Nov. 2014

Kewei Lu, Abon Chaudhuri, Teng-Yok Lee, Han-Wei Shen and Pak Chung Wong, "Exploring vector fields with distribution-based streamline analysis". *IEEE Pacific Visualization Symposium (PacificVis) 2013*, pp.257,264, Feb. 27 2013-March 1 2013.

Boonthanome Nouanesengsy, Teng-Yok Lee, Kewei Lu, Han-Wei Shen and Tom Peterka, "Parallel particle advection and FTLE computation for time-varying flow fields". In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*, pp.1,11, 10-16 Nov. 2012.

Jie Zhang, Kewei Lu, Yang Xiang, Muhtadi Islam, Shweta Kotian, Zeina Kais, Cindy Lee, Mansi Arora, Hui-wen Liu, Jeffrey D. Parvin and Kun Huang "Weighted Frequent Gene Co-expression Network Mining to Identify Genes Involved in Genome Stability". *PLoS Computational Biology*, 2012, 8(8).

Yang Xiang, Kewei Lu, Stephen L. James, Tara B. Borlawsky, Kun Huang and Philip R.O. Payne, "k-neighborhood Decentralization: A Comprehensive Solution to Index the UMLS for Large Scale Knowledge Discovery". *Journal of Biomedical Informatics*, Vol 45, Issue 2, pp 323-336, April 2012

Fields of Study

Major Field: Computer Science and Engineering

Studies in:

Computer Graphics	Prof. Han-Wei Shen
Artificial Intelligence	Prof. Eric Fosler-Lussier
Statistics	Prof. Shili Lin

Table of Contents

List of Tables

List of Figures

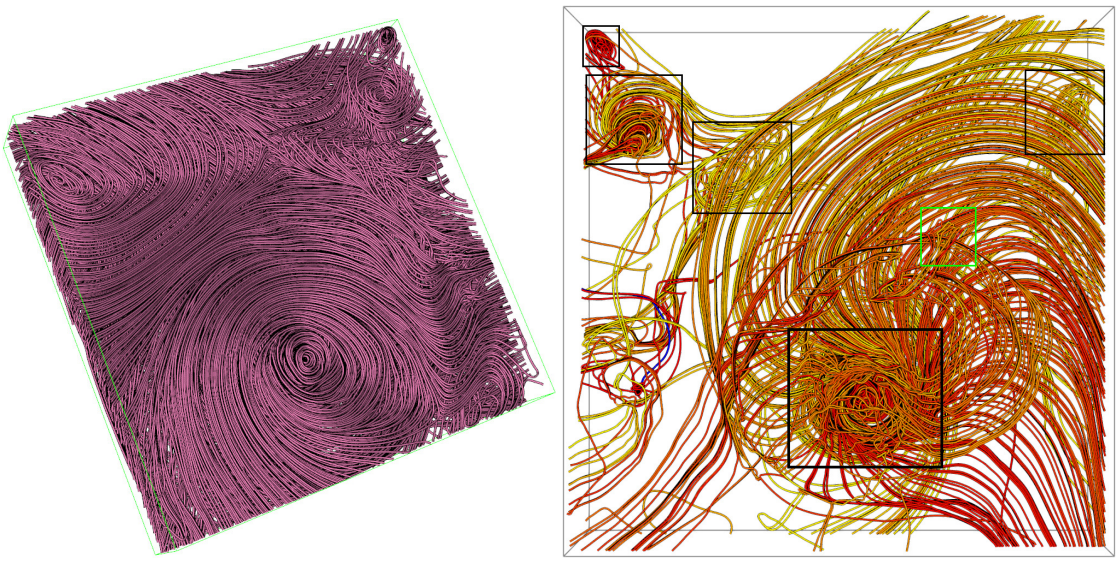
Chapter 1: Introduction

1.1 Background and Motivation

Due to the ever increasing of computing power in the last few decades, petaflop computers enable various scientific and engineering simulations to generate data sets on an unprecedented scale. Around the year 2018, we are expected to enter the age of exaflop computing [1] when we will be able to generate much more data than we have today. As the size of data continues to increase, effective techniques to visualize and explore those large-scale data in many science and engineering disciplines are becoming increasingly more important.

For data at such a scale, traditional approaches to perform visualization and exploration are no longer effective due to several reasons:

- The computation of visualization is slow and expensive. When the data size becomes large, the time taken to compute visualization from the data such as streamlines from vector fields is expensive. Besides time consuming, we have constraint in the memory when the data become too big to be stored in a single machine's memory. One approach to reduce the computation and memory costs is to use distributed-memory parallelism. Thus, the original visualization algorithms need to be revised to provide high parallel efficiency.



(a) (b)

Figure 1.1: (a). The visualization of 2000 streamlines generated from Hurricane Isabel data set. The visualization suffers from visual cluttering and occlusion. It is difficult for users to discover features hidden inside the data set. (b). A query-driven based visualization. Five big swirling areas highlighted by black squares and one small swirling area highlighted by a green square area could be observed.

- As the data become larger and more complex, visualization suffers from visual cluttering and occlusion problems. This problem is illustrated in Figure 1.1. Figure 1.1(a) shows the visualization of all 2000 streamlines generated from the Hurricane Isabel data set. The visualization suffers from visual cluttering and occlusion problems. Figure 1.1(b) shows a query-driven based visualization. Several swirling areas could be easily observed from the visualization.

In order to address the aforementioned problems, we propose a statistical distribution-based query-driven framework to visualize and analyze large-scale vector fields and multivariate scalar fields. Query-driven visualization is an efficient technique to interactively explore and discover interesting features existing in a large-scale scientific data set. It

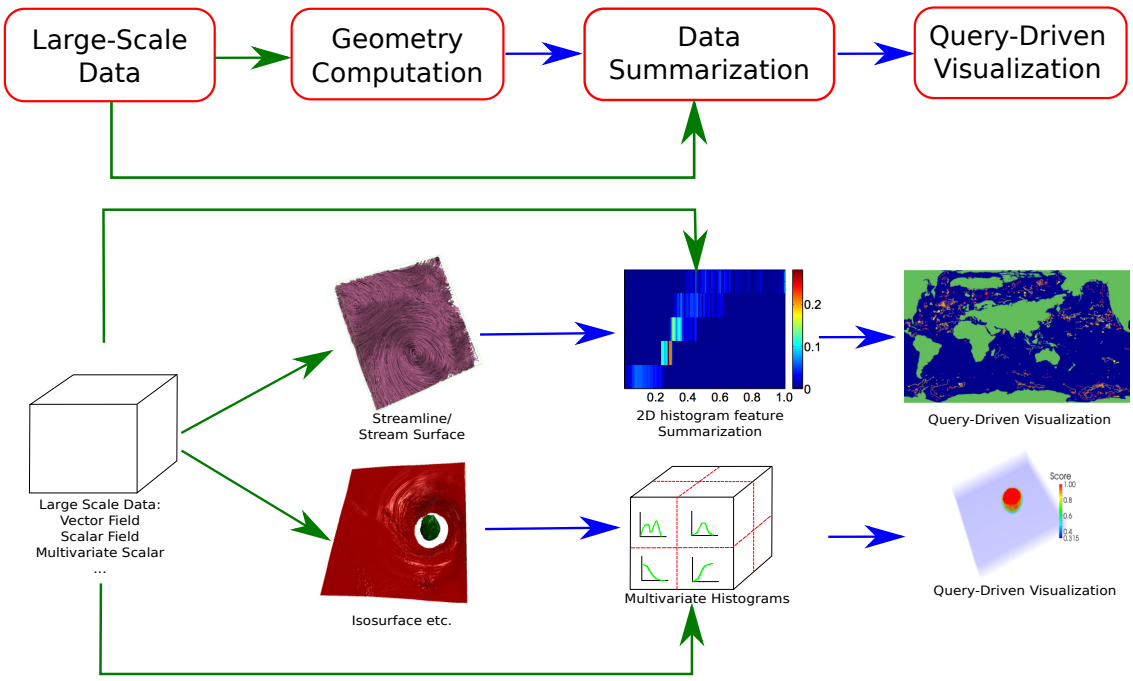


Figure 1.2: The general statistical distribution-based query-driven framework. Given a large-scale data set, the statistical data summarization could be computed from either the original data sets or the geometries (streamlines, stream surfaces, isosurface etc.) computed from the original data sets. The final query-driven visualization is then directly applied to the summarized data instead of the original data.

allows users to analyze and visualize large-scale data by selecting data records that are interesting based on a specific criterion. However, directly applying query-driven approaches to the original large-scale data set is usually inefficient due to its huge size. In order to achieve efficient query-driven visualization for large-scale scientific data, we propose a statistical distribution-based query-driven framework which first uses distributions to summarize the large-scale data set and then applies the query-driven visualization techniques on the summarized data set. A schematic view of our framework is shown in Figure 1.2. In the framework, the statistical data summarization could be computed from either the original data sets or the geometries (streamline, stream surface, isosurface etc.) computed

from the original data sets. The final query-driven visualization is then directly applied to the summarized data instead of the original data. For the steps that access and interact with the original data such as geometry computation and data summarization (green arrows), we face the computation and memory intensive issues as mentioned earlier. Distributed-memory parallelism could be adopted to solve the problem in the proposed framework.

1.2 Distribution-based Query-driven Visualization of Flow Fields

The first part of the dissertation focuses on developing techniques to visualize and analyze large-scale flow data. Flow fields are commonly encountered in many science and engineering disciplines. Streamlines and stream surfaces are two popular methods to visualize flow fields. However, when the size of data is large, computing geometries such as streamlines and stream surfaces from flow fields is a challenging task. Even we are able to compute a large number of geometries from flow fields efficiently with some scalable parallel algorithms, directly visualizing those densely generated geometries will suffer from visual cluttering and occlusion. Thus, an efficient technique to explore those densely generated geometries is needed to perform effective analysis on the original flow field.

1.2.1 Scalable Stream Surfaces Computation

As mentioned earlier, when data are too large to fit into a single machine's memory, distributed-memory parallelization can be used to support efficient computation of streamlines and stream surfaces. A streamline is defined as a curve traced from a seed location inside the flow field and is tangential everywhere to the local flow direction while a stream surface is defined as a surface traced from a seeding curve inside the flow field. They both can be used to reveal flow features such as vortices inside the flow field.

Several parallel algorithms have been proposed for streamlines computation, and those algorithms can be divided into two types: parallelizing-over-seeds and parallelizing-over-data methods. For the first type, seeds are distributed across processes, and then each process computes the streamlines originated from the assigned seeds. For the other type, the entire data is decomposed and then distributed across processes. Each process computes the streamlines in its own blocks. However, relatively little research has been done to parallelize stream surface generation on distributed memory machines. This is because load-balanced parallel stream surface computation is nontrivial, due to the strong dependency in computing the positions of the particles forming the stream surface front.

Therefore, in this dissertation, we focus on scalable stream surfaces computation. We present a parallel algorithm for scalable stream surfaces computation on distributed memory machines. In this algorithm, the seeding curves are divided into segments and then assigned to different processes. Each process computes a part of the surface from the seeding curve segments assigned. For each seeding curve segment, the seeds at the boundary are duplicated so that each seeding curve segment can be advanced independently by the processes. The entire data is also divided into blocks and then distributed across the processes. During integration, a data block is loaded on demand from other processes if the data block is not in the current process's local memory. Load balancing is achieved by a work stealing mechanism combined with a runtime seeding curve partitioning scheme.

1.2.2 Distribution-based Streamline Analysis

After streamlines or stream surfaces are computed from large-scale flow fields, we need to visualize and explore those computed streamlines or stream surfaces to help users to better understand the data. For example, the flat region will have straight streamlines while the

turbulent region is more likely to have curved streamlines. However, a direct visualization of those densely computed streamlines is seldom useful due to the difficulties caused by visual cluttering and occlusion. Also, users are often interested in specific flow features, for example, the circular flow structure around a vortex. Hence, it is important to be able to cluster streamlines based on their similarity and show only those that are relevant to the user's query.

We propose a distribution-based query-driven framework to interactively visualize and explore flow fields. Our approach is motivated by the observation that statistical distributions of measurements along the trajectory of a streamline can be used as a robust and effective descriptor to measure the similarity between streamlines. In our approach, we use the distribution of feature measures over a streamline to be the descriptor, and then use this descriptor to measure the similarity between streamlines. With this descriptor, we are able to group streamlines with similar shape, so that each time we can display a group of streamlines which shows a particular feature to minimize the occlusion problem.

1.3 Distribution-based Visualization and analysis of Multivariate Data

The second part of the dissertation focuses on analysis of large-scale multivariate data. When dealing with multivariate data, in order to understand the data, it is useful for us to study the relationships among different variables and show the regions of interest based on user specified criteria. In the presence of large-scale multivariate data, efficient techniques to summarize and analyze the data are needed.

1.3.1 A Compact Multivariate Histogram Representation for Query-driven Visualization

Most scientific data sets contain multiple variables. For example, the data set Hurricane Isabel which models a strong hurricane in the west Atlantic region in September 2003 contains thirteen variables, such as pressure, temperature, and QRain. Compared to univariate data, analyzing and visualizing multivariate data sets is much more challenging and hence remains an active topic of research. Several query-driven techniques have been proposed to analyze and visualize multivariate data by selecting data records that are interesting based on a specific criterion. An example of such criterion could be a boolean range query such as $(100 \leq \mathit{pressure} \leq 200)$ AND $(0.0 \leq \mathit{QRain} \leq 0.01)$ [95]. When the size of data is large, it is nontrivial to analyze multivariate data efficiently due to the memory constraint and a large number of data records.

In this dissertation, we present a distribution-based method for the analysis and visualization of large-scale multivariate data. Distributions play an important role in analyzing and visualizing data generated from different scientific and engineering simulations, as they are particularly effective for data aggregation, summarization, and query. We propose a query-driven visualization framework based on block-wise multivariate distributions to analyze multivariate data. Those multivariate distributions are computed at block levels, for example, block of sizes 8^3, 16^3. However, computing and storing multivariate histograms is nontrivial because the memory requirement can grow exponentially with respect to the number of variables if stored as a multi-dimensional array. In order to solve this problem, we present a compact representation of multivariate histogram to reduce the storage cost in Chapter 5. We also describe several histogram query operations. Based on our compact representation and those query operations, several query-driven visualization applications

to explore and analyze multivariate data sets are presented to illustrate the effectiveness of our technique.

1.3.2 Multivariate Volumetric Data Analysis and Visualization through Bottom-up Subspace Exploration

As an effective way to visualize and analyze multivariate data sets, volume rendering has been frequently used, although designing good multivariate transfer functions is still non-trivial. In this dissertation, an intuitive and interactive workflow allowing users to design multivariate transfer functions is presented. To handle large scale data sets, in the preprocessing stage we reduce the number of data points through data binning and aggregation, and then a new set of data points with a much smaller size is generated. The relationship between all pairs of variables is presented in a juxtaposed matrix view, where users can navigate through the different subspaces. Each subspace shows the joint distribution of two variables. An entropy-based method is used to help users to choose which subspace to explore. We propose two weights: scatter weight and size weight that are associated with each projected point in different subspaces. Based on those two weights, data point filter and kernel density estimation operations are employed to assist users to discover interesting features. For each user-selected feature, a Gaussian function is constructed and updated incrementally. Finally, all those selected features are visualized through multivariate volume rendering to reveal the structure of the data. With our system, users can interactively explore different subspaces and specify multivariate transfer functions in an effective way.

1.4 Organization

The rest of the dissertation is organized as follows: Chapter 2 provides the necessary background and related works. Chapter 3 and 4 present the proposed works on large-scale flow fields visualization and analysis. Chapter 3 presents the work on scalable computation of stream surfaces on large scale vector fields and Chapter 4 describes the work on exploring vector fields with distribution-based streamline analysis. The next two chapters, Chapter 5 and 6 discuss the proposed methods for large-scale multivariate data visualization and analysis. Chapter 5 discusses a query-driven visualization method based on a compact multivariate histogram representation and Chapter 6 presents a system for bottom-up multivariate volumetric data exploration and visualization. Finally, Chapter 7 summarizes the dissertation.

Chapter 2: Background and Related Work

This chapter discusses and reviews the related works for parallel streamline and stream surface computation, streamline similarity quantification, multivariate data analysis and distribution-based query-driven visualization. In Section 2.1, different strategies of parallel streamline computation are discussed. Existing algorithms for stream surface generation and the previous works for parallel stream surface computation are also discussed. Section 2.2 reviews the previous works on streamline shape analysis and similarity measurement. Related works on flow field analysis via streamline clustering and query are also discussed. Section 2.3 of this chapter reviews previous research on multivariate data analysis. Finally, Section 2.4 discusses the related works on distribution-based query-driven visualization which includes distribution-based data visualization, efficient distribution computation, query-driven visualization and multivariate histogram representation.

2.1 Parallel Flow Visualization

2.1.1 Parallel Streamline Computation

A streamline is defined as a curve traced from a seed location in the flow field and is tangential everywhere to the local flow direction. It is computed by applying numerical integration techniques such as the Runge-Kutta methods to obtain a sequence of positions starting from a user-specified seed location. To parallelize streamline computation,

two strategies are often used. One is to decompose the entire data set into a number of disjoint blocks and then distribute those blocks to the processes. With the block assignment, each process will compute the streamline segments only if they pass through their own blocks. This parallelizing-over-data strategy was used in an early parallel streamline computation method proposed by Sujudi and Haimes [33]. Based on this idea, Peterka et al. [81] presented a study of parallel particle tracing for both streamline and pathline generation for steady and unsteady flow fields. Kendall et al. [54] proposed a MapReduce [28] like system called *DStep* for parallel streamline computation. Nouanesengsy et al. [75] addressed the load balancing issue in the parallelizing-over-data approach and proposed a workload-aware partitioning algorithm based on a graph representation of the original flow field. Since parallel stream surface generation is fundamentally different from parallel streamline generation in many ways, it is difficult to directly apply this workload-aware partitioning algorithm to balance the parallel stream surface generation. Instead, a dynamic load balancing approach is used in our method. The second strategy of parallel streamline computation is parallelizing-over-seeds, where each process computes streamlines from the seeds that are assigned to it and loads the data blocks required to complete the integration of the seeds on demand. Camp et al. [15] investigated the benefit of using an extended memory hierarchy for the parallelizing-over-seeds strategy. Muller [73] et al. studied the use of work requesting on the integral curve computation based on the parallelizing-over-seeds strategy. Pugmire et al. [85] reviewed both strategies and presented a new hybrid approach for streamline computation.

2.1.2 Parallel Stream Surface Computation

A stream surface is defined as a surface traced from a seeding curve inside the flow field. An ideal stream surface can be constructed by densely sampling an infinite number of seeds on the seeding curve and connecting the resulting streamlines together. The streamlines originated from the seeding curve can be parametrized by their arc length $t \in [0,N]$. For a constant t, the positions of all streamlines form a front, sometimes referred to as a time line, and can be parameterized by another parameter $s \in [0,1]$, which indicates the originating position of the streamline on the seeding curve. These two sets of curves, streamlines and the fronts, define the stream surface parametrized over s and t. Any point on a stream surface can be represented as:

$$\Upsilon(s,t) = X_{C(s)}(t) \tag{2.1}$$

where C is the seeding curve parametrized by s, $X_{C(s)}(t)$ defines a curve traced from a seed $C(s)$ on C and parametrized by t; and Υ defines the stream surface constructed by advancing C and is parametrized by s and t. Figure 2.1 illustrates the parameterization of a stream surface.

In practice, tracing an infinite number of streamlines to construct the stream surface is not feasible. A front-advancing algorithm for generating a stream surface was first presented by Hultquist [45]. In the algorithm, the seeding curve is discretized by a number of seed points, which are then advanced by integrating these seed points with a numerical method. Adjacent pairs of streamlines form stream ribbons which are triangulated by a greedy method. Adaptive refinement such as insertion and deletion of streamline seeds is employed in Hultquist's algorithm to control the resolution of the advancing front. Splitting of the surface is handled by comparing the advancing direction of adjacent seeds in

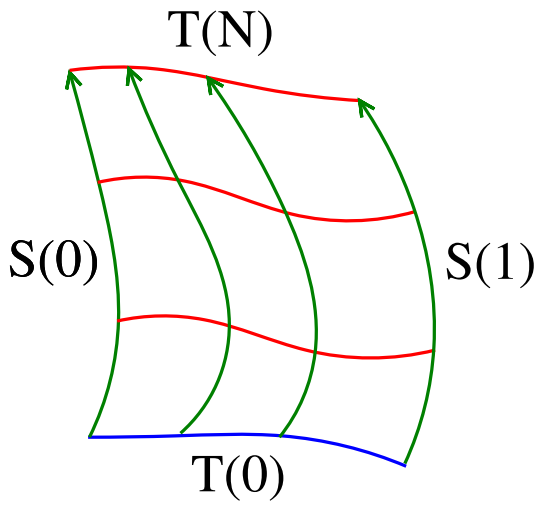


Figure 2.1: A stream surface parametrized by s and t. The blue curve is the initial seeding curve. The red curves, also called time lines, can be seen as the front of the seeding curve moved by the flow direction.

Hultquist's algorithm. While Hultguist's algorithm is simple to implement, it may not perform well when the flow in local regions has large variations in direction and magnitude. Later, an improved algorithm was proposed by Garth et al. [36]. In their algorithm, the authors employ an arc length based streamline integration scheme. Also, additional front refinement criteria such as surface curvatures were introduced.

In addition to Hultquist's and Garth's algorithms that construct stream surfaces explicitly, there exist methods that construct stream surfaces implicitly [96, 101]. In the method by van Wijk [101], continuous scalar values are specified for the grid points on the boundaries of the flow field, and then the scalar values for the interior grid points are computed by backward tracing of streamlines. With the scalar field, stream surfaces can be constructed by extracting isosurfaces. Stöter et al. [96] extended van Wijk's approach to stream, path, streak, and time surfaces for 3D time-varying flow fields and solve some limitations of

van Wijk's approach such as limited domain coverage and limited control of the seeding curve. Stream surface algorithms have also been proposed for curvilinear grids [100] and tetrahedral grids [89].

Besides the computation of stream surfaces, rendering of surfaces has also been studied [10, 46]. Hummel et al. [46] studied how to use transparency and texturing mapping techniques to visualize and convey more information of integral surfaces. Born et al. [10] utilized contour lines and halftoning to enhance the surface shape representation. Illustrative surface streamlines are used to depict the flow direction on the stream surface. Several interaction features are also introduced.

Compared with streamlines, parallelization of stream surfaces is more challenging because of the dependency in computing the streamline points that form the stream surface front. This dependency can be seen from the sequential stream surface algorithms [36, 45, 70], which construct stream surface by advancing the stream surface front step by step across the seeding curve and applying adaptive refinement at every step to ensure the accuracy of the stream surface. This dependency makes both the parallelizing-over-data and parallelizing-over-seed strategies inefficient because excessive communication between processes may occur. Previously, Camp et al. presented a parallel stream surface algorithm [13] that does not utilize the front-advancing method. Instead, in their method the stream surface is approximated by tracing streamlines from seed points on the seeding curve and triangulation is performed afterwards. Surface refinement is achieved by inserting new seeds between two adjacent seed points if the distance between the streamlines originating from these two seeds is larger than a predefined threshold. This refinement process stops when no adjacent pair of streamlines exceeds the threshold. Since it is likely that the discontinuities in the flow field may prevent the refinement of a stream surface from

stopping, the algorithm does not insert new seeds if the distance between the adjacent seed points is less than a threshold.

In Chapter 3, we present a parallel algorithm utilizing the front-advancing scheme for explicit stream surface computation. We present a hybrid parallelizing-over-data and parallelizing-over-seed algorithm to keep the I/O cost low and to ensure a balanced workload distribution. With the front-advancing scheme, our algorithm can address some of the limitations of Camp's algorithm such as no seed deletion is done when flow begins to converge, resulting in too many unnecessary points being generated on the stream surface. Also, in Camp's algorithm, seed insertion only takes place on the original seeding curve, and hence no adaptive refinement is performed across the surface. Below we describe our algorithm in detail.

2.2 Streamline Similarity Quantification

Streamlines have been studied as a major technique for flow visualization over the past decades. Quantifying the importance of a streamline in terms of representing flow features and measuring the similarity between streamlines in terms of shape and location are identified as two key problems. Moberts et al. [72] evaluated the role of four different similarity metrics, namely mean of closest points, closest points, hausdorff [25] and end points distance [11], in clustering DTI fibers, which are geometrically identical to streamlines. Pointwise Euclidean distance based similarity metrics [67] are widely used for clustering. Shi et al. [92] utilized the variation of different geometric properties of pathlines as a mean to classify them. Schlemmer et al. [90] presented an approach to extract and visualize flow patterns for 2D flow fields based on moment invariants. A more recent work [86] proposed

Hausdorff distance as a similarity measure for streamlines and achieved meaningful clustering based on that. However, none of the above methods compute and utilize distributions from streamlines. Our proposed technique in Chapter 4 shows that distributions computed from streamlines can be effective as a distance measure, which, as opposed to some existing techniques, is less sensitive to location and orientation and faster to compute.

We are able to achieve streamline clustering and querying based on our proposed metric. Clustering of streamlines [86, 111] and equivalently, clustering of the underlying field [43, 97] can be achieved in different ways. Querying and selection of streamlines is not trivial mainly because there is no standard way to formulate such queries. A query-by-example scheme for streamlines is proposed by Wei et al. [104] which accepts shape templates from the user. Any form of query that can be converted to a distribution should be applicable to our proposed method.

2.3 Multivariate Data Analysis

2.3.1 Multivariate Attributed Space Exploration

Multivariate data usually results in a high dimensional attribute space. In order to visualize this high dimensional attribute space and discover interesting features, several approaches have been proposed. One way to explore and visualize the high dimensional attribute space is to use various dimension reduction techniques. The goal of those dimension reduction techniques is to reduce the number of dimension of the original data while preserving the original high dimensional data characteristics. Two common dimension reduction techniques are Principle Component Analysis (PCA) and Multidimensional Scaling(MDS). PCA transforms the original high dimensional data into an orthogonal coordinate system while maximizing variance along the axes. MDS maps the original high

dimensional data into a low dimensional space while maintaining the dissimilarities between data points. PCA [77] and MDS [40] have been utilized in various techniques to visualize and analyze the multivariate data. Parallel coordinate plots(PCP) [47, 48] is another way to visualize the attribute space and help users to analyze and understand the multivariate data. In a parallel coordinate system, attributes/variables are defined as parallel vertical lines. A data point corresponds to a polyline constructed by connecting vertices on the parallel vertical axes. The position of the i_{th} vertex on the i_{th} axis is defined by the value of the i_{th} attribute. However, parallel coordinate has a major limitation which is when there is a larger number of data points, the visualization becomes clutter and it is ineffective to show the internal structure of the data. A number of techniques have been proposed to deal with this problem [8, 44, 51, 76]. In [51], the authors solved the visual clutter problem by first clustering the polylines into clusters and then using high-precision textures to represent those clusters to make both the visualization and rendering effective. In order to highlight different structural information, the authors also proposed to apply transfer functions on the high-precision textures. Novotny et al. [76] presented a focus+context parallel coordinate visualization based on binned data representation. Besides visualizing the high dimensional attribute space, there are techniques focusing on identifying interesting features presented multivariate data. In [32], the authors presented a framework to support interactive specification and identification of features in multivariate data. In [49], the authors introduce local statistical complexity which can be used to detect important features in a multivariate dataset.

2.3.2 Transfer Function Design

Volume rendering is a popular method to visualize volumetric data. It maps data values to optical properties such as color and opacity to visually reveal the structure of the data. The mapping from data values to optical properties is achieved through transfer functions. A good transfer function can reveal the underlying features in the data in a more effective way. A lot of research effort have been devoted to the design of transfer function in the past. Arens et. al. [4] presented a survey on different types of transfer function used for volume rendering. The transfer function is categorized into six types: 1D data-based, gradient based, curvature based, size based, texture based and distance based. In order to better separate materials and boundaries, many techniques consider gradient magnitude while designing transfer function [56, 58, 59, 66, 103]. The gradient magnitude quantifies how fast the value changes. By incorporating gradient magnitude in the design of transfer function, different materials can be better characterized. Kindlmann et. al. [55] presented a transfer function design approach by utilizing curvatures to enhance the effectiveness of volume rendering. The size of features has also been considered when designing transfer functions. [26]. Correa et. al. [27] observed that the structure of features in volume data could be classified based on the occlusion pattern. Based on this observation, the authors proposed a new transfer function to classify volume data by utilizing the ambient occlusion information of voxels. Several other properties, such as texture [12] and statistical information [41, 79] have also been used to assist the design of transfer function. Besides those different types of transfer function used for volume rendering, techniques that focus on semi-automatic transfer function design have also been proposed [99, 114].

When we are dealing with multivariate data, multidimensional volume rendering can be used to visualize and analyze the data. However, designing an effective multidimensional

transfer function for multidimensional volume rendering is even harder and requires lots of researches. Kniss et. al. [57] discussed the issues of using separate transfer functions for each variable and multidimensional transfer function lookup table. Then, the authors proposed to use Gaussian transfer function which overcomes the issues mentioned above. Liu et. al. [64] analyzed high dimensional data through identifying a set of low-dimensional linear subspaces. Then, the authors presented an approach to assist multivariate transfer function design by animating transitions between different views. Guo et. al. [40] presented an effective and scalable system that combines parallel coordinates plots (PCP) and multidimensional scaling (MDS) projection to assist novel multidimensional transfer function design.

2.4 Distribution-based Query-driven Visualization

2.4.1 Distribution-based data visualization

As an effective way to summarize large scale scientific data, distributions have been used in various data analysis and visualization applications. HIXEL, introduced by Thompson et al. [98], stores one histogram per grid point or a data block. Based on this representation, the authors proposed an algorithm for fuzzy isosurface computation. Gu et al. [39] compute block-level histograms to track features presented in time-varying datasets. Lundstrom et al. [65] studied the design of transfer functions in direct volume rendering based on local histograms and demonstrated its effectiveness in a clinical evaluation. Given the distributions computed from data, many statistical metrics can be derived. Xu et al. [110] used Shannon's entropy computed from point-wise distributions to analyze the complexity of local data. Martin and Shen [68] introduced histogram spectra which are computed

in each subvolume over a range of sampling frequency. The histogram spectra is used to achieve interactive level of detail selection for large scale time-varying multivariate data.

2.4.2 Distribution Computation

Due to the important role that distributions play in data analysis and visualization, various techniques have been proposed for efficient computation of distributions. Integral histogram was proposed by Porikli et al. [83] to compute the histogram of arbitrary region in constant time. However, because a histogram contains multiple bins, storing one integral histogram per grid point is very expensive and hence better solutions are needed for three-dimensional data. Martin and Shen [69] developed span distributions to reduce the I/O and memory costs for range queries. Two novel transformations, a decomposition and a similarity-driven indexing, of the integral histogram was proposed by Chaudhuri [20] to reduce the storage cost associated with integral histograms. Lee et al. [63] tackled the storage problem of integral histograms by using the discrete wavelet transform. When the data size is large, distributions could be computed on distributed machines in parallel [19]. Rubel et al. [87] used FastBit to achieve efficient parallel computation of 2D histograms and conditional histograms. In [94], the authors presented several efficient algorithms for computing 2D conditional histograms using bitmap indexing in parallel. All these methods focus on efficient computation of univariate or 2D histograms, while we focus on the computation of multivariate histograms.

2.4.3 Query-driven Visualization

Query-driven visualization(QDV) allows users to analyze and visualize large scale data by selecting data records that are defined to be interesting based on a specific criterion. Those specific criteria could be specified as Boolean range queries such as $(100 \leq pressure \leq$

$200)$ AND $(0.0 \leq QRain \leq 0.01)$ [95]. Effective QDV techniques rely on fast retrieval of those data records. Different indexing techniques such as FastBit [95, 108, 109] have been used to effectively identify those interesting data records. Distribution-based query-driven techniques have been proposed in [52, 98].

2.4.4 Multivariate Histogram Storage

Unlike univariate histogram, the computation of multivariate histogram is nontrivial. One challenge related to multivariate histograms is the curse of dimensionality which states that as the number of variables increases, the size of the multivariate histogram increases exponentially. To solve this problem, a compact representation was proposed by Chanussot [18]. In Chanussot's method, the data space is indexed by a space filling curve so that the N dimensional vector could be represented as a single scalar value which is its curvilinear abscissa along the space filling curve. Then, instead of storing all the entries of the multivariate histogram, they only store the non-empty entries and represent the multivariate histogram as a table with two columns: the first column contains the index of the non-empty entries. The second column stores the corresponding frequencies. A potential problem is that given a large number of variables and bins, the curvilinear abscissa could be an extremely large value that requires a large number of bits to represent. The authors also present another modified representation in which the first column records the number of empty entries between the current non-empty entry with the previous non-empty one. By using this modified representation, it will give relative smaller values in the first column. However, a problem related to this representation is that it does not explicitly contain the bin index for each variable which is required by several query operations. Summarization and factorization is needed to get the bin index of each variable. Goil et al. [37] presented

a bit-encoded sparse structure(BESS) to store multi-dimensional sparse data. The multi-dimensional data is divided into chunks first, and then each chunk is indexed separately with bit strings. Since the size of a chunk is smaller than the original data, they require less bits to encode. The chunk offset for each chunk also needs to be stored in order to dereference to get the original bin indices. In Chapter 5, we propose a new representation to store block-wise multivariate histograms with less storage. Instead of dividing the multi-dimensional histogram into chunks, a data space transformation is applied first to reduce the size of the original multivariate histogram and then the smaller multivariate histogram is indexed. Compared with chunking, this representation gives us an one to one mapping from the encoded bin index to the original bin index which allows us to perform certain query operations such as histogram marginalization without decoding.

Chapter 3: Scalable Computation of Stream Surfaces on Large Scale Vector Fields

Effective visualization of flow fields plays an important role in analyzing data generated from scientific simulations, for which displaying streamlines and stream surfaces are two popular methods. A streamline is the trajectory of a mass-less particle traced from a seed point in the field, and a stream surface is a surface traced from seeds originated from a curve. A stream surface can be seen as the union of an infinite number of streamlines, and is typically approximated by a polygonal mesh that connects streamlines seeded from selected positions on a seeding curve. As the size of data continues to grow, efficient computation of streamlines and stream surfaces becomes increasingly more difficult. To address the challenge, researchers have proposed various parallel computation algorithms, most of which are for streamline computation [14, 16, 23, 75, 81, 85] with a few for stream surfaces [13]. Generally speaking, stream surface computation is more complicated than computing streamlines since stream surface integration requires seeds to be inserted or deleted dynamically across the stream surface front when the flow diverges or converges. This dynamic insertion and deletion of seeds requires synchronization among the computation of individual streamlines, and thus makes parallel computation of stream surfaces much more challenging.

Algorithms for parallel streamline computation can be divided into two types: *parallelizing-over-seeds* and *parallelizing-over-data* methods. For the parallelizing-over-seeds methods, seeds are distributed across all processes, and then each process computes the streamlines originated from the assigned seeds and loads the required data when necessary. For the parallelizing-over-data methods, data are first decomposed into blocks, and these data blocks are distributed to the processes. Each process computes the streamlines in its own blocks and sends the streamlines to the other processes once the streamlines hit the block boundaries. Applying these two strategies directly to parallel stream surface computation is nontrivial because of the dependency in the adjacent streamlines that form a stream surface. When using the parallelizing-over-seeds strategy, for example, if two adjacent seeds are distributed to different processes, a considerable amount of communication is needed to connect the streamlines to a surface. On the other hand, if parallelizing-over-data is used, challenges arise when the same seeding curve passes multiple data blocks that belong to different processes.

In this chapter, we present a scalable parallel stream surface computation algorithm based on a front-advancing approach that is also used by several sequential stream surface algorithms [36, 45, 70]. The main contribution is an efficient algorithm to compute stream surfaces for large scale vector fields that can handle flow convergence, divergence and split. We show that our algorithm is highly scalable when running on large supercomputers. We use a hybrid parallelizing-over-data and parallelizing-over-seed strategy to ensure a balanced workload distribution for machines with large processor counts. In our algorithm, we divide the seeding curve into segments and then assign these segments to different processes so that each process independently runs Garth's front-advancing algorithm [36] to compute a part of the surface. Flow convergence, divergence, and split are handled by seeds

deletion, insertion and surface ripping, which is similar to the sequential front-advancing algorithm. For each seeding curve segment, the seeds at the boundary are duplicated so that each seeding segment can be advanced independently by the processes. We divide the entire data set into blocks which are distributed across the processes. During integration, when a process needs a data block not in its local memory, the process will get the block from the other process over the network on demand. Load balancing is ensured by a work stealing mechanism combined with a runtime seeding curve partitioning scheme.

The rest of the chapter is organized as follows. In Section 3.1, we describes our algorithm in detail, including preprocessing, parallel computation, and load balancing. Section 3.2 presents several experimental results to study the scalability of our method. Conclusion and future work are presented in Section 3.3.

3.1 Method

In this section, we present our parallel algorithm for stream surface computation based on a front-advancing method. Our algorithm is closely related to Garth's algorithm [36], but other front-advancing methods [45, 70] can also benefit from our parallelization strategy. In our implementation, we are using OSUFlow, a parallel particle advection library [81] developed by The Ohio State University and Argonne National Laboratory. In our implementation, the Runge-Kutta-Crash-Karp($RK45$) numerical integration scheme is used. DIY [80] is used to decompose the domain, assign data blocks, and perform inter-process communication. The Block I/O layer (BIL) library is used to achieve high I/O efficiency when loading disjoint data blocks across the processes [53]. By using BIL, each process posts requests for the required data blocks, and then a single collective I/O is issued to read the data in parallel. MPI-I/O is used to write the final results to disk in parallel.

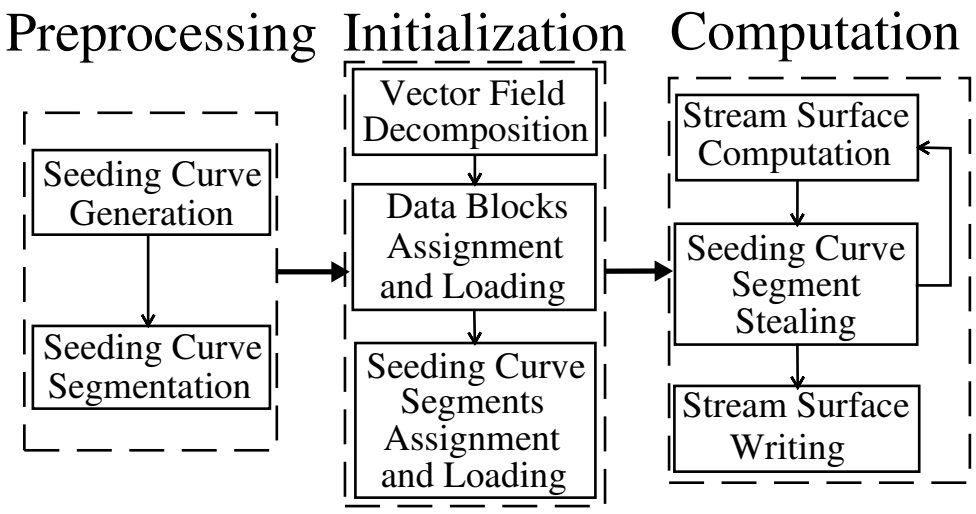


Figure 3.1: An overview of the pipeline used in our algorithm. The preprocessing stage runs in serial, and the initialization and computation stages run in parallel.

An overview of the pipeline in our algorithm is shown in Figure 3.1, which can be divided into three main stages: preprocessing, initialization and stream surface computation. The preprocessing stage does not need run in parallel since the computation cost of this stage is negligible compared with the initialization and computation stages. In our implementation, the preprocessing stage runs in serial, and the initialization and computation stages run in parallel.

3.1.1 Preprocessing

The preprocessing stage includes seeding curve generation and seeding curve segmentation. Seeding curves can be either provided by the user or randomly generated. In our implementation, we randomly generate the seeding curves, where the user specifies how many seeding curves to generate, the maximum number of seeds on each seeding curve,

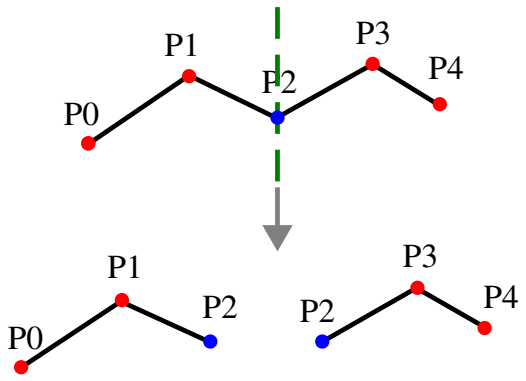


Figure 3.2: An example of our seeding curve segmentation. Given a seeding curve with five particles P_0 to P_4 shown at the top, we uniformly divide it into two pieces at particle P_2. The particle P_2 is the boundary particle after the cutting, and is duplicated for both of the two seeding curve segments.

and the distance between adjacent seeds on the seeding curve. Generating seeding curves is shown in Algorithm 1.

Given a set of seeding curves, we partition them into segments that have an equal number of seeds to balance the computational workload. The seeds at the boundaries of the seeding curve segments, referred to as the boundary seeds, are duplicated so that they are in both the left and the right segments as shown in Figure 3.2. The duplication of the boundary seeds eliminates the need to communicate when advancing the surface fronts between different processes.

3.1.2 Initialization

The purpose of initialization in our algorithm is to distribute the data and the seeding curve segments to the processes. In this stage, first the vector field is decomposed into axis-aligned 3D blocks, where the number of blocks should be greater or equal to the

Algorithm 1 Seeding Curve Generation

Let n be the number of seeding curves, m be the maximum number of seeds per seeding curve, and s be the distance between adjacent seeds on the seeding curves
for $i = 1$ to n **do**
    Randomly generate two seeds p_0 and p_1
    $dir \leftarrow$ normalized vector points from p_0 to p_1
    $d \leftarrow$ distance between p_0 and p_1
    Initialize l to be an empty list
    Initialize $curDistance \leftarrow 0$
    for $j = 0$ to m **do**
        $p \leftarrow p_0 + j \times s \times dir$
        $curDistance \leftarrow curDistance + s$
        Push p to l
        if $curDistance > d$ **then**
            Break
        end if
    end for
    Add l to the seeding curve pool
end for

number of processes. To allow a continuous interpolation of data in each block independently, one layer of ghost cell is added to the boundary of each dimension. After data decomposition, different block assignment strategies can be used such as the round-robin and the processes-order continuous schemes [81]. The round-robin assignment scheme tends to perform better than the processes-order continuous assignment scheme for parallel streamline computation as illustrated by Peterka in [81] because a better computation load balancing can be achieved. However, in our parallel stream surface algorithm since we use runtime seeding curve segment partitioning and work stealing to solve the load balancing problem, the choice of data assignment scheme is not as important. In our implementation, we use a simple round-robin data assignment scheme, in which data blocks are assigned to processes by using a block-cyclic distribution.

Besides the data blocks, seeding curve segments also need to be distributed to the processes. Similarly, we can employ various strategies to assign the seeding curve segments. In our algorithm, we assign seeding curve segments to processes in round-robin order.

3.1.3 Parallel Stream Surface Computation

The next stage in our algorithm is to compute the stream surfaces by parallelizing the computation of stream surface patches originated from the seeding curve segments. Then, the individual stream surface patches computed by different processes are combined together to form a complete stream surface. Our parallel algorithm employs Garth's front-advancing algorithm to complete each stream surface patch, with several strategies to ensure a scalable parallel performance. We discuss these strategies below.

Data Loading On Demand

Our algorithm is parallelized over seeding curve segments. Seeding curve segments are first distributed across the processes evenly, and then the processes integrate the assigned stream surface patches in parallel until the maximum number of steps for each patch is reached or the patch goes out of bounds. During integration, data blocks are loaded into memory when necessary. Unlike the load on demand implementation presented in [85] where data blocks are loaded from disk and I/O can become a bottleneck, we load all the data blocks from disk in the initialization stage as described in section 3.1.2, and then during integration, if the required block is not available locally it will be requested from the process who owns it via communication. In our algorithm, a new data block is needed when the particle (the current position of a streamline) on the stream surface front requires it for the integration. Because all particles on the stream surface front need to be integrated one step together to test the flow divergence or convergence condition, when a particle needs

a new block, we need to immediately obtain it to continue the surface patch computation. When no more local memory is available to accommodate the new block, one of the in-core blocks needs to be evicted to make room for the block.

In our algorithm, each process maintains a list of the assigned seeding curve segments $\{c_0, c_1, ..., c_n\}$ and computes the stream surface patches one at a time. During the computation, particles on the seeding curve segment are integrated one step further to advance the stream surface front. When process P_i integrates a particle, it first checks which data block this particle needs, and then checks whether this data block is already in memory. The block is available in the local memory either because this block was loaded in the initialization stage, or it has already been requested from another process. If the block is not available locally, first P_i checks which process has the data block and then sends a message to that process say P_j asking for the data block. When process P_j receives the request, it sends the data block to P_i, and in the mean time still keeps the original copy of the data block.

Cache

To minimize the number of times that a process has to request data blocks from the other processes, each process maintains a small cache to store the data blocks after they are received. The Least Recently Used (LRU) replacement policy is employed. As will be shown in section 3.2, the cache helps reduce the number of data requests a process issues to the other processes; this is because our method does not access data blocks randomly but has a specific data access pattern which is favored by the cache. Given a seeding curve segment, because of the front-advancing algorithm we adopt, in every iteration, our algorithm advances the front of the stream surface by integrating all the particles on the current front from one side to the other side. Because the distance between adjacent particles on the

Algorithm 2 Main Structure

```
1: Partition domain and tasks
2: for all data blocks assigned to my process do
3:     read the data blocks
4: end for
5: for all tasks assigned to my process do
6:     read the tasks to my task pool T_p
7: end for
8: while My task pool T_p is not empty do
9:     Get a task from the head of T_p and execute
10: end while
11: Steal tasks from other processes until no task can be stolen
```

current surface front is usually smaller than the block size, the adjacent particles are most likely to stay in the same block. Also, since the step size is usually much smaller than the block size, the particles on the current surface front are most likely to be in the same block as they were in in last iteration. Therefore, in the next iteration, the same blocks are likely to be accessed again. In general, the data blocks are repeatedly accessed in a short amount of time. A 2D example is shown in Figure 3.3. The 2D vector field data is decomposed to 4 blocks with ID from 0 to 3. The blue curve is the initial seeding curve segment with five particles on it, and it is advanced for five steps. Suppose we integrate from left to right, blocks $\{0,0,0,1,1\}$ are accessed in the first iteration. In the next iteration, we still access blocks $\{0,0,0,1,1\}$, the same blocks as before. Considering all these five steps, the order of data blocks to be accessed is: {0,0,0,1,1,0,0,0,1,1,0,0,1,1,1,2,0,1,1,1,2,2,3,3,1,2,2,3,3,3}. Notice that data blocks are repeatedly accessed frequently which makes a cache beneficial.

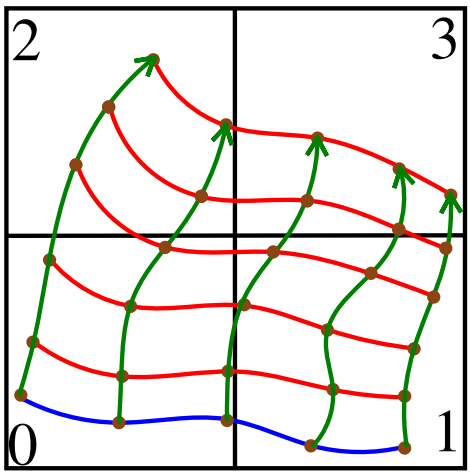


Figure 3.3: An example that shows the data access pattern of stream surface integration. The blue curve is the seeding curve segment with five particles. Since the front of the seeding curve segment is advanced one step at a time by integrating the particles. Data blocks are repeatedly accessed.

Dynamic Load Balancing

In this section, we discuss our dynamic load balancing strategies, namely work stealing and runtime seeding curve segment subdivision. A high level description of our algorithm is listed in Algorithm 2.

As mentioned in section 3.1.3, each process maintains a list, referred to as the task pool T_p, to store the assigned seeding curve segments $\{c_0, c_1, ..., c_n\}$, each of which in the pool is a task. To improve load balancing of our algorithm, a dynamic load balancing scheme based on work stealing [30] is employed. We call the process who steals tasks a *thief* and the process whose tasks are stolen a *victim*. When a process finishes all its tasks, it steals tasks from the other processes. To do this, the thief must first select a victim that has extra

unfinished tasks. This selection of victim is done based on a random selection method that has been proven optimal [9]. Once a victim is selected, an MPI message is sent to the victim to ask for tasks. When the victim receives the message, it checks its task queue to see whether there are extra tasks. If there are available tasks, half of the tasks starting from the end of the victim's task queue are sent to the thief. Previously researchers have shown that stealing half of the available tasks each time can generate satisfactory results. Since tasks are distributed to more processes, it is easier for the idle processes to find tasks to steal [30]. On the other hand, If the victim has no tasks available, it sends a message back to inform the thief. The thief will then select a new victim randomly and repeats the process until either it finds available tasks or a global termination of the program is detected. Work stealing is shown in Algorithm 3.

Runtime seeding curve segment subdivision means that during integration, if a seeding curve segment grows too wide as the result of flow expansion, the segment is split and half of the segment is pushed to the end of the task pool T_p. The purpose of seeding curve segment subdivision is to split the seeding curve segments that have high workload into smaller pieces, which allows better workload distribution when combined with our work stealing scheme. In our experiments, we found that the number of particles on a seeding curve segment provides a good approximation of the workload. To incorporate this idea, in our implementation we check the number of particles on the seeding curve segments after every integration step. If it is greater than a user defined *cut threshold* Th_{cut}, the middle particle on the segment is selected as the partitioning particle. As in the segmentation of the seeding curves in the preprocessing stage, the partitioning particle is duplicated to the two resulting seeding curve segments. We continue advancing one of the two new seeding

Algorithm 3 Work Stealing

```
1: while my task pool is empty and no global termination do
2:     Randomly select a victim p_v
3:     Send a message to p_v to ask for tasks and wait for a reply
4:     if p_v has tasks available then
5:         Get half of tasks t from the tail of p_v's task pool
6:         Put t to my task pool
7:     end if
8: end while
```

curve segments and put the other one to the end of the task pool. Runtime subdivision is shown in Algorithm 4.

Global Termination Detection

To issue a global termination of the parallel program, we have to detect when all the processes are idle and have no more work to do. In our algorithm, we adopted Francez's algorithm [35] for this purpose. We organize the processes into a binary tree, and the termination process involves sending messages up and down the tree in multiple rounds. A round consists of a bottom-up and a top-down propagation. In each round, messages are propagated from bottom up first. When the 'root' receives messages from its children, it determines whether the program should be terminated or not and propagates the decision top to down. This multi-round communication continues until all processes become thieves and no process is a victim since the last round.

Data Structure

Since the original seeding curves are partitioned into multiple segments and these segments may further be subdivided later, when a seeding curve segment is stolen, additional information such as the original seeding curve it belongs to needs to be sent to the thief with

Figure 3.4: An example that shows the structure of a seeding curve segment. The yellow header includes three floating point numbers representing the number of steps this seeding curve segment has advanced, the ID of the original seeding curve it comes from, and the number of particles on the seeding curve segment. The red part is the particles. The green part is the current step size for each particle and the blue part is the number of steps that have been integrated for each particle.

the seeding curve segment. In our implementation, the data stored in the data structure of a seeding curve segment include four parts as shown in Figure 3.4. The yellow header in Figure 3.4 contains three floating point numbers that record the number of steps this seeding curve segment has advanced, the ID of the original seeding curve it comes from, and the number of particles on the seeding curve segment. For n particles on the seeding curve, the next $n \times 3$ floating point numbers shown in red color record the 3D positions of the particles on the seeding curve segment. The green part contains n floating point numbers recording the current step size for each particle on the seeding curve segment. Because we use the RK45 integration scheme with adaptive step size, the step size of each particle is dynamically changed based on the flow complexity. If a seeding curve is stolen by another process, this information is needed for the thief to continue to integrate the particles with the right step size so that the boundary of each patch matches as shown in Figure 3.5. The last n floating point numbers store the number of steps that each particle has integrated. In either Hultquist's or Garth's algorithm, as the stream surface front moves forward, whether a particle should advance or not for this iteration depends on whether the flow is divergent,

Algorithm 4 Runtime Subdivision

1: Given a seeding curve segment in the task pool T_p
2: **while** The maximum number of steps is not reached and the seeding curve segment has not hit the global boundary **do**
3: Advance one step further
4: **if** The number of particles on the current seeding curve segment $> Th_{cut}$ **then**
5: Divide the seeding curve segment at the middle seed resulting in two new seeding curve segments C_1 and C_2
6: Set the current seeding curve segment to C_1
7: Put C_2 to the tail of the task pool
8: **end if**
9: **end while**

convergent or neither. This makes it necessary to record the number of steps that each particle has integrated since different particles may have traveled a different distance. Given this seeding curve segment data structure, each seeding curve segment is a $5 \times n + 3$ floating point array. The reason of using floating point numbers for everything even for integer properties such as count is that we can pack everything into a single array and then send to the other processes. This choice is mainly driven by our implementation and we can adopt other data types if necessary.

As explained above, the work stealing victim sends half of its seeding curve segments from its task queue to the thief. These seeding curve segments are organized as a single message. The structure of this message is shown in Figure 3.6. Suppose the message contains N seeding curve segments, the header shown in yellow color contains $N+1$ floating point numbers. The first number records how many seeding curve segments are contained in this message. The next N numbers record the length of each segment. The rest are the seeding curve segments. So each stealing operation results in a length of $N+1+\sum_{i=0}^{N}(5 \times N_i + 3)$ floating point numbers to be communicated between processes, where N_i is the number of seeds on the i^{th} seeding curve segment.

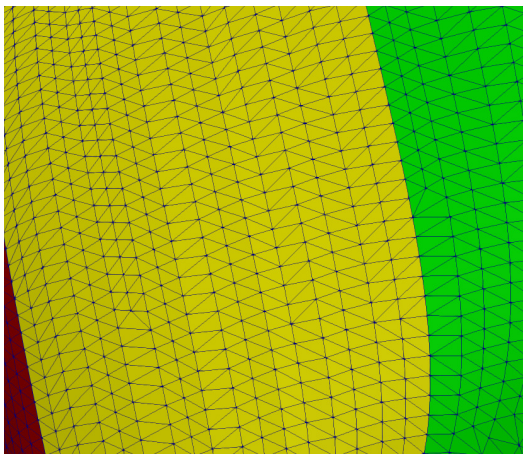


Figure 3.5: By sending the current step size of each particle along with the seeding curve segment to the thief, the boundaries of different patches match with each other.

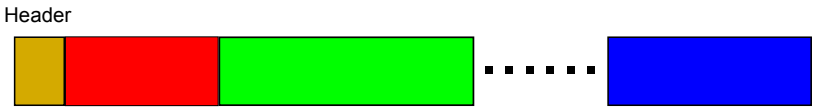
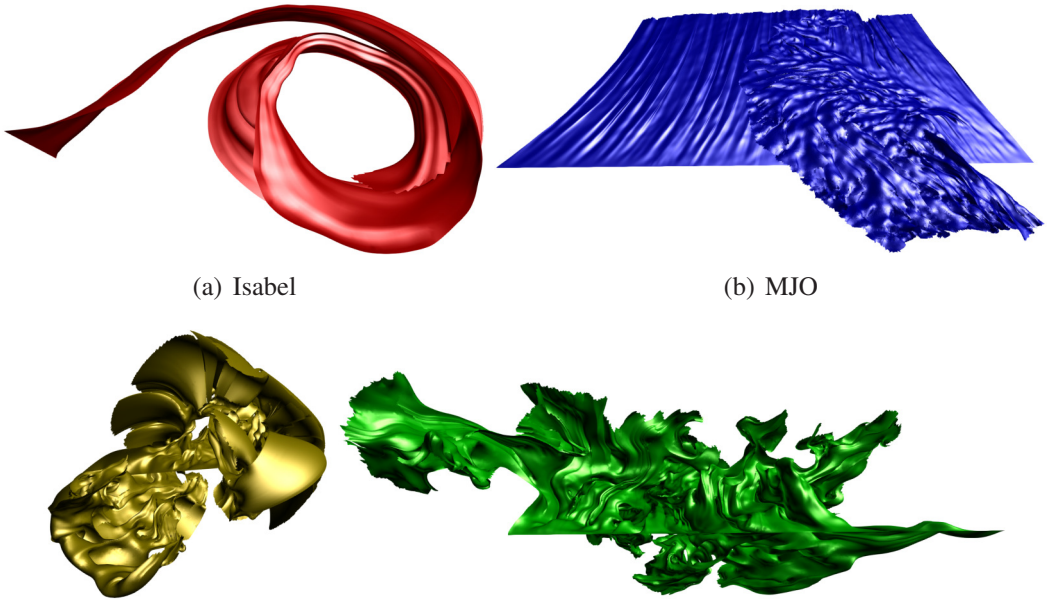


Figure 3.6: An example that shows the structure of the message containing the tasks stolen by the thief. Suppose the thief stole N seeding curve segments. The yellow part is the header that includes $N+1$ floating point numbers. The first number represents the number of the seeding curve segments. The following N numbers are the length of each seeding curve segment. The remainder is the N seeding curve segments.

3.2 Results

In this section, we present the performance of our parallel stream surface algorithm. We conducted several experiments using four steady flow field datasets of different resolutions. The dataset *Isabel* models a strong hurricane in the West Atlantic region in September 2003. Its size is 287MB with a resolution of $500\times500\times100$. We randomly generated 256 seeding curves, each with a maximum 128 seeds and the distance between the seeds



(a) Isabel (b) MJO

(c) Plume (d) Nek

Figure 3.7: Images of a single stream surface computed from the datasets Isabel, MJO, Plume, and Nek respectively.

was 0.5 cell. *Madden-Julian Oscillation(MJO)* is a dataset simulating the Madden-Julian oscillation effect over the Indian and Pacific Oceans. It is 926MB with a resolution of $2699\times599\times50$. 256 seeding curves were randomly generated, each with a maximum of 128 seeds and 0.5 cell between the seeds. The *Plume* data set is generated from a simulation of solar plume on the surface of the sun with a resolution of $504\times504\times2048$. The data size is 5.9GB. 64 seeding curves were generated, each with a maximum of 512 seeds and 0.5 cell sample distance between the seeds. The last test dataset, referred to as *Nek*, is a simulation of thermal hydraulics generated by the Nek5000 solver. It was created from a large-eddy simulation of the Navier-Stokes equation for the MAX experiment [71]. It has a resolution of $2048\times2048\times2048$ for a total size of 96GB. 64 seeding curves were generated, each with

a maximum of 2048 seeds and 0.5 cell between the seeds. Figure 3.7 shows images of the stream surfaces computed from these four datasets.

For all the experiments, we measured the time spent on stream surface computation and communication for each process. The computation time, referred to as Comp, is the total amount of time spent on advancing seeding curve segments using the Runge-Kutta-Crash-Karp integration scheme. The reported computation times are the average computation time per process. The communication time, referred to as Comm, measures the time for sending and receiving messages to get data blocks from the other processes, work stealing, global termination detection, and the time to manage communication. The average communication time per process is reported.

3.2.1 Parameter Setting

There are several parameters which influence the performance of our parallel stream surface computation algorithm: the block size, the cache size, the threshold used to divide the seeding curves in the preprocessing stage (referred to as the preprocessing threshold), and the threshold used to divide the seeding curves at run time (referred to as the runtime threshold). We conducted experiments to study the influence of these parameters to the performance of our algorithm. All tests were conducted on the Blue Gene/Q *Vesta* system at the Argonne Leadership Computing Facility. *Vesta* has 2,048 nodes, each holding 16 PowerPc A2 1600MHz cores sharing 16 GB of RAM and utilizes the General Parallel File System. The total memory is 32 TB.

Block size Versus Cache Size

Among the four parameters mentioned above, the block size and the cache size influence the time to load data blocks from the other processes the most. To study the impact of

these two parameters, we performed a sequence of experiments using all four datasets. For *Isabel* and *MJO*, the block sizes tested were 5^3, 10^3, 25^3, and 50^3. Blocks of 8^3, 16^3, 32^3, and 64^3 were tested for *Plume*, and blocks of 32^3, 64^3, 128^3 were tested for *Nek*. These sizes do not include the ghost layer, although the actual data block loaded has one layer of ghost cells in each dimension. In our tests, 128, 512, 1024, and 1024 processes were used for *Isabel*, *MJO*, *Plume*, and *Nek* respectively. The size of the cache is controlled by the number of blocks N that it can hold. For the *Isabel* data set, for example, a cache that can hold N blocks means the amount of memory allocated to the cache is $N \times 50^3$ vectors. Therefore, with the same amount of memory, more than N blocks that have a smaller size such as 5^3,10^3 and 25^3 can fit to the cache. In our experiments, the cache size was varied with values of $1, 2, 3, 4$, and infinity for *Isabel* and *MJO*, where infinity means the amount of memory allocated to the cache is able to hold the entire dataset. For *Plume* and *Nek*, the cache size was varied with values of $2, 4, 8$, and 16. As for the stream surfaces generated in our test, the maximum number of integration steps was set to $400, 200, 600$, and 500 for *Isabel*, *MJO*, *Plume*, and *Nek*, respectively. The seeding curve partitioning thresholds, i.e., the width of the seeding curve segment in the preprocessing stage, and the threshold for the run time seeding curve partitioning were fixed across all experiments. The influences of these two parameters on our algorithm will be studied in section 3.2.1.

Figure 3.8 shows the results of our tests. Besides the communication time, we also measured and reported the average amount of data transferred by each process, and the average number of data blocks loaded by each process. Previously Peterka et al. [81] reported that for parallel streamline computation, using small block size and round-robin block assignment will distribute workload more evenly, but the downside is that it incurs more communication. In our tests, we found that smaller block sizes did not improve load

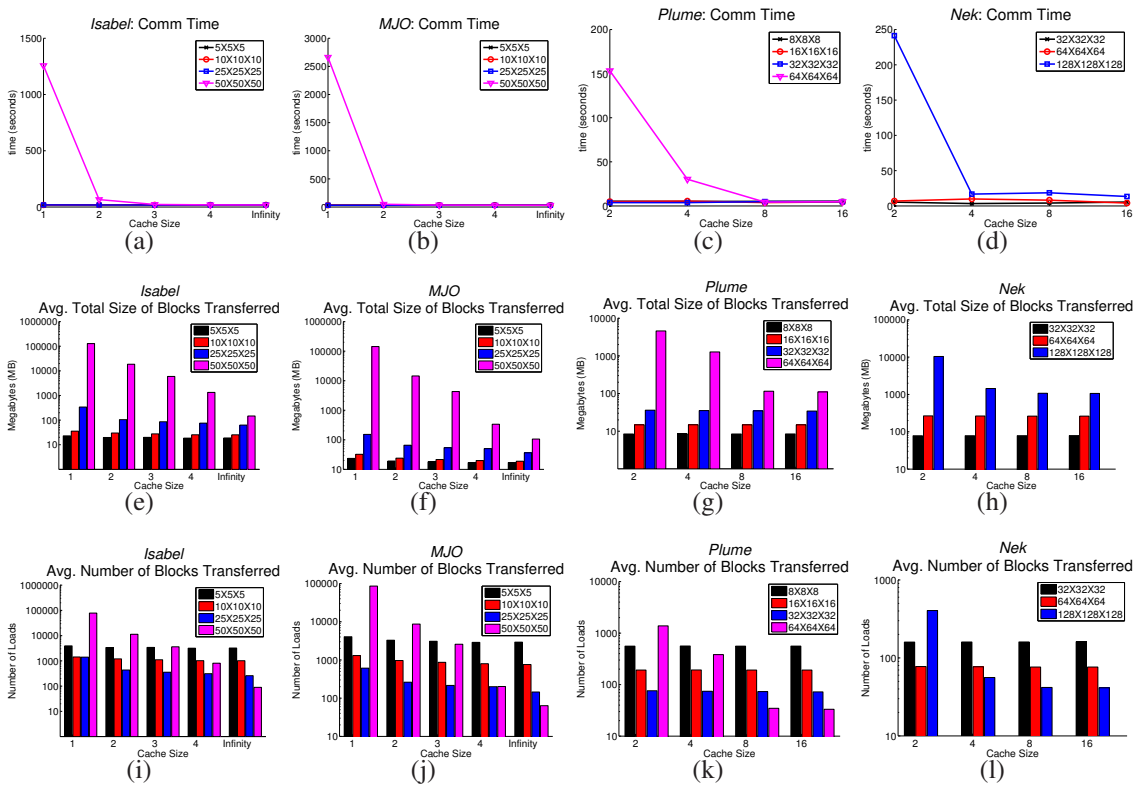


Figure 3.8: Top row: The cache size versus communication time under different block sizes. Middle row: The cache size versus the average total size of data blocks transferred per process under different block sizes. Bottom row: The cache size versus the average number of block loads per process under different block sizes.

balancing since the workload of a process mainly depends on the seeding curve segments instead of the block size. In general, a smaller block size results in less data to be transferred among processes because smaller blocks pack the stream surface front more tightly and hence less unnecessary data are transferred. This is shown in Figure 3.8(e)-3.8(h).

As the cache size increases, the number of blocks that were loaded decreases. When the cache size is large enough or goes to infinity, using a larger block size requires fewer loads compared with a smaller block size, because a smaller block size implies the stream surface front will pass through more blocks. Figure 3.8(i)-3.8(l) show this trend. The

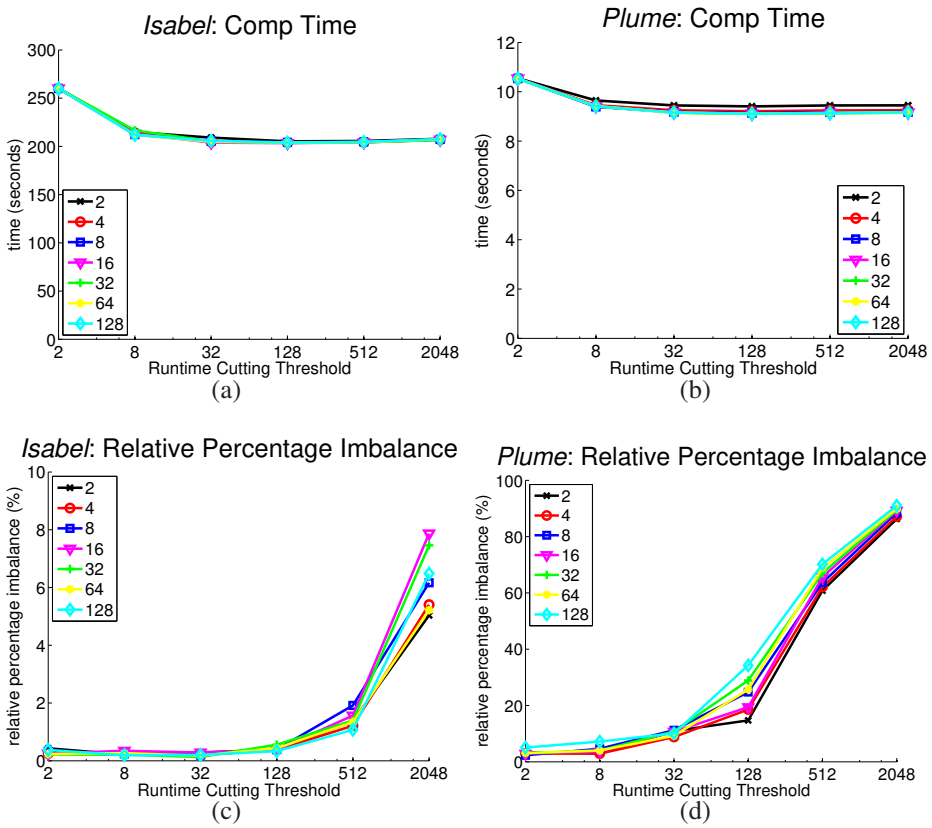


Figure 3.9: Top row: Runtime cutting threshold versus computation time under different static cutting thresholds. Bottom row: Runtime cutting threshold versus relative percentage imbalance under different static cutting thresholds.

influence of block size for the communication cost is shown in Figure 3.8(a)-3.8(d). For a fixed cache size, using a large block size will increase the data loading time, if the cache is not large enough to load the required blocks to cover the stream surface front. As cache size increases, the difference between different block sizes vanishes.

Seeding Curve Cutting Thresholds

The seeding curve cutting thresholds in the preprocessing stage and at run time also can influence performance and load balance. Hereafter we refer to the threshold used in

the preprocessing as *static cutting threshold*, and the threshold used at run time as *runtime cutting threshold*. We conducted tests using *Isabel* and *Plume* with 128 and 1024 processes to study the impact of these two thresholds. The block size was set at 25^3 for *Isabel* and 32^3 for *Plume*. For both tests, we used a large cache size to minimize the number of data loads so that we can focus on the computation time and workload balance. Cache size for *Isabel* was 1600 and for *Plume* was 200. Other parameters used are identical to those in section 3.2.1.

The results of our test are shown in Figure 3.9, where curves of different colors represent the computation times using different static cutting thresholds, and the x axis represents the runtime cutting threshold. As can be seen in Figure 3.9(a) and 3.9(b), as we increase the value of the runtime cutting threshold, i.e., making the seeding curve segments wider, the decreases in the computation time are noticeable at the beginning but then stop later. This is because when the runtime cutting threshold is too small, such as 2, too many seeds are duplicated, hence increasing the computation time. On the other hand, when the runtime cutting threshold increases, the number of duplicated seeds decreases, decreasing the computation time. When the runtime cutting threshold becomes even larger, such as 32, the duplicated seeds are only a small fraction of the total number of seeds, so the decrease in computation time becomes negligible.

We use the formula $(\frac{t_{max} - t_{avg}}{t_{max}}) \times 100\%$, referred to as the relative percentage imbalance, to measure the load balancing of our algorithm, where t_{max} is the maximum computation time taken by a process and t_{avg} is the average computation time over all processes. The relative percentage imbalance are shown in Figure 3.9(c) and 3.9(d). The increase of relative percentage imbalance when the runtime cutting threshold becomes larger indicates that workloads among the processes become imbalanced. The reason for this is that when

using a larger threshold, fewer seeding curve segments are produced so that it is harder for a process to find work to steal. Also, larger thresholds produce longer seeding curve segments with larger differences between the workload of each. From the figures, we can also see that the static cutting threshold does not have a significant impact on the performance of our algorithm compared with the runtime cutting threshold. However, we note that the static cutting threshold should not be too small, such as 2. This is because if the flow converges, different seeding curve segments can not be merged at runtime. Generally speaking, extremely small or large cutting thresholds degrade the performance. Considering the tradeoff between having more work and being less load balanced, in our experiments, a value between 32 and 128 was best.

Cache Size Versus Runtime Cutting Threshold

In this section we investigate the relationship between the runtime cutting threshold and the cache size. We conducted tests using *MJO* and *Nek* with 512 and 1024 processes, respectively. The block size was 25^3 for *MJO* and 32^3 for *Nek*. Different runtime cutting thresholds were tested for each dataset. The cache size was represented by the maximum number of blocks that can be stored and varied for each runtime cutting threshold. Other parameters are identical to those in section 3.2.1.

The results in Figure 3.10(a) and 3.10(b) show that as the cache size increases, the communication time decreases at the beginning and then becomes stable. When a smaller cache size is used, fewer number of blocks could be loaded which are not enough to cover the entire stream surface front, resulting in too many data blocks to be loaded on the fly. Increasing the cache size allows more data blocks to be loaded, resulting a decreased number of data block loads. As we continue to increase the cache size, at some point the performance is not improved. This is because the current cache size is large enough to

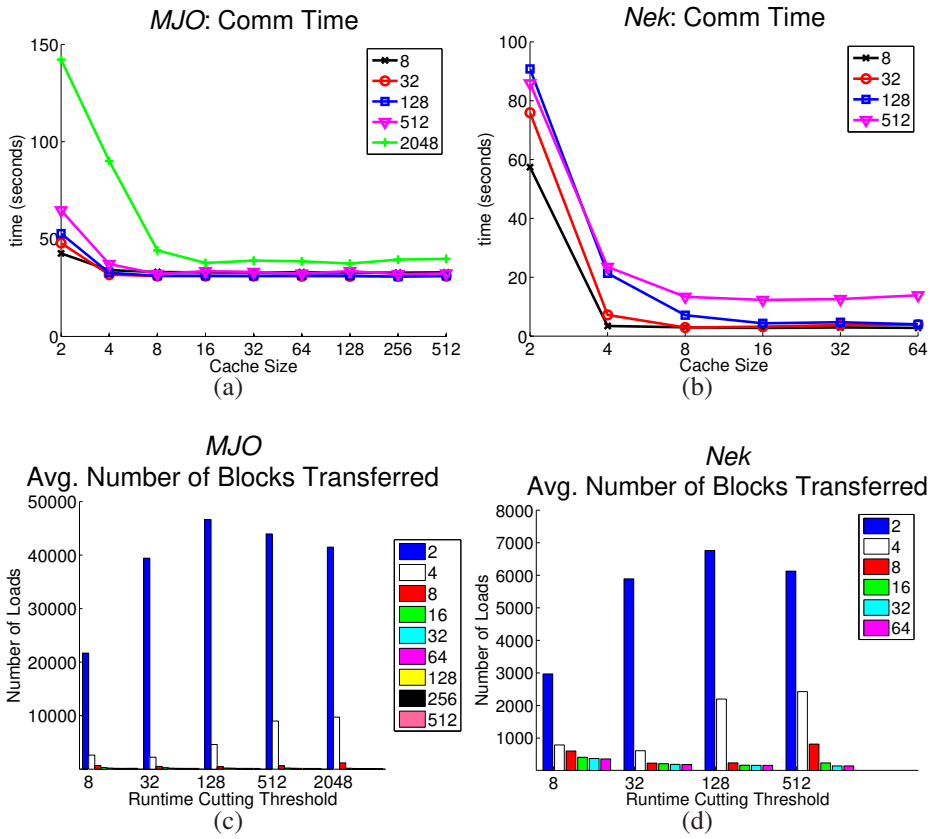


Figure 3.10: Top row: Cache size versus communication time under different runtime cutting thresholds. Bottom row: Runtime cutting threshold versus average number of block loads under different cache sizes.

load sufficient blocks to cover the surface front. This point depends on the runtime cutting threshold because a larger runtime cutting threshold produces longer seeding curve segments which generally need more data blocks.

3.2.2 Scalability

Strong scaling tests were conducted to show the scalability of our algorithm. We measured the total time for running our algorithm, including time for stream surface integration,

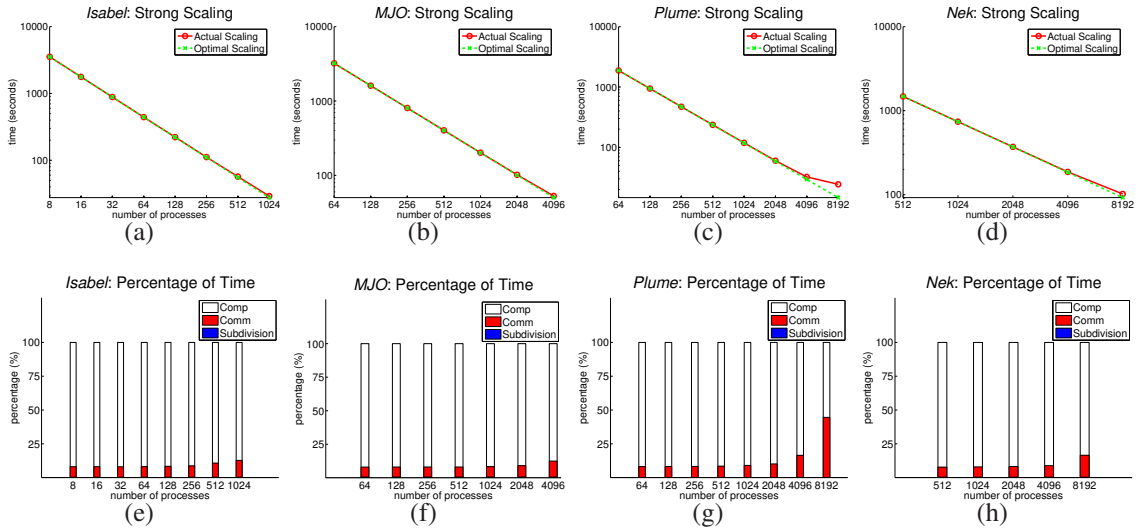


Figure 3.11: Strong scaling results. The top row graphs the time for running our algorithm, including time for stream surface computation, communication and runtime seeding curve segment subdivision for different numbers of processes. The bottom row contains the percentage of time spent on each component.

communication and runtime seeding curve segment subdivision. Disk I/O time was not included, because it was done in the initialization stage through the library BIL [53]. For all four of our datasets, the cache size was 16. The static cutting threshold and the runtime cutting threshold were 16 and 128 for *Isabel*, *MJO*, and *Nek* respectively. For *Plume*, they were set at 16 and 32. For *Isabel*, up to 1K processes were used. The maximum number of integration steps was 400, and 25^3 block size was used. For *MJO*, up to 4K processes were used. The maximum number of integration steps was 200 and the block size was 25^3. For *Plume*, up to 8K processes were used. 900 was the maximum number of integration steps and the block size was 32^3. For *Nek*, up to 8K processes were used. The maximum number of integration steps was 650, and the block size was 32^3.

The results of strong scaling tests are shown in Figure 3.11. The top row shows the scalability of our algorithm. The results demonstrate a good scalability of our algorithm up to a large process count. The bottom row shows the percentage of time spent on different components: computation, communication and runtime seeding curve segment subdivision. Overall, a majority of the time was spent on computation. For all of the four datasets, the time for seeding curve segment subdivision is negligible. Also, as the process count increases, the percentage of communication increases. This is because with a large number of processes, the time spent on work stealing increases.

3.2.3 Load Balancing

As described in Section 3.1.3, work stealing and runtime seeding curve subdivision are employed in our algorithm to balance the computational workload of different processes. To demonstrate the effectiveness of our load balancing algorithm, we computed the relative percentage imbalance under different process count and the results are shown in the top row of Figure 3.12. Perfect computational load balancing was observed for *Isabel*, *MJO* and *Nek*. For *Isabel* and *MJO*, the relative percentage imbalance was below 5% for up to 1K and 4K processes. For *Nek*, the imbalance was below 10% for up to 8K processes. *Plume* has a relative percentage imbalance value below 10% for up to 4K process. When $8K$ processes were used for *Plume*, the relative percentage imbalance increased to around 40% and this was because there were not enough tasks for this large number of processes. We also measured the computation time for each process when 1K processes were used for *Isabel* and 4K processes for *MJO*, *Plume*, and *Nek*. The results are shown in the bottom row of Figure 3.12. The small difference in the computation time among the different processes indicates the effectiveness of our load balancing algorithm.

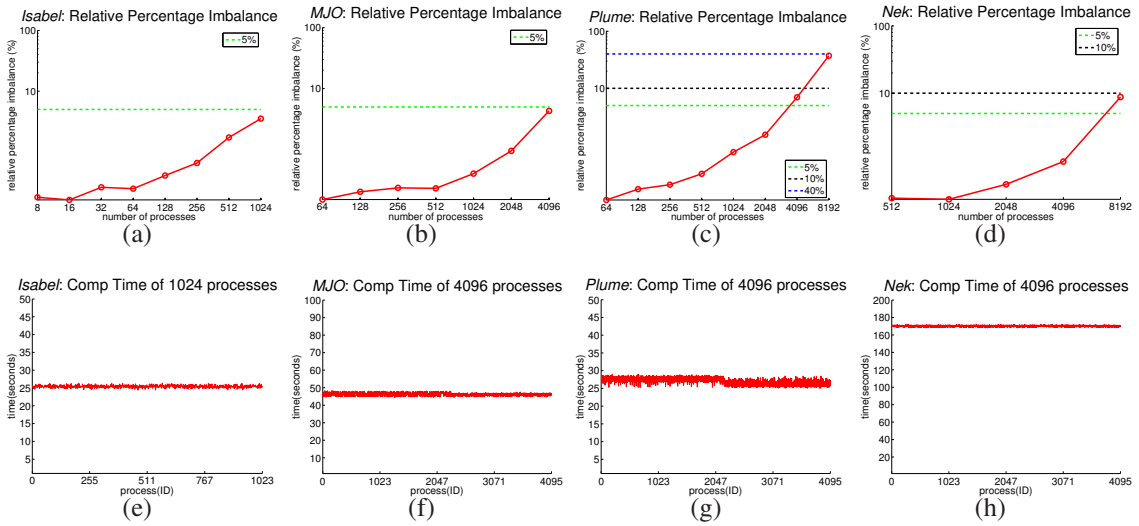


Figure 3.12: The top row shows the relative percentage imbalance under different process count. The bottom row contains the computation time for each process when 1K processes were used for *Isabel* and 4K processes were used for *MJO*, *Plume*, *Nek*.

3.2.4 Discussion

Among these four parameters that are studied in section 3.2.1, the runtime cutting threshold determines the total amount of particle tracing to be done and the load balancing of our algorithm. Our experiments suggest that the runtime cutting threshold should not be too small; otherwise, it will increase the total amount of work due to seed duplication. On the other hand, if the threshold is too large, load imbalance occurs. In our experiments, a value between 32 and 128 was best. The static cutting threshold should be smaller than the runtime cutting threshold to avoid immediate partitioning at runtime but not extremely small such as 2. The cache improves the performance of our algorithm by holding the recently used data blocks and reducing the number of data block loads. The cache size is related to the block size. Given a fixed amount of memory allocated to the cache, more blocks of smaller size could be loaded compared with blocks of larger size. Given a fixed

block size, the ideal cache size is to be able to load the blocks that cover the stream surface front, which is related to the runtime cutting threshold. In general, a smaller block size is preferred since it incurs less data transfer among the processes and also they can cover the stream surface front more tightly. However, a side effect of using a smaller block size is that it will cause more data loading operations. When an extremely small block size is used, a larger number of data loading will take place, and hence the performance will be impacted negatively because of the overhead of each data loading.

Comparing the performance with Camp's algorithm where I/O and busy wait time dominate the total time as mentioned by Camp in [13], in our method the majority of time is spend on computation. Also, they mentioned that their method can suffer from load imbalance, while because of the dynamic load balancing strategy we used, our method does not have load imbalance problem as shown in Section 3.2.3.

3.3 Conclusion and Future work

In this chapter, we present a new algorithm for parallel stream surface computation. By partitioning seeding curves into segments, stream surfaces are split into small stream surface patches, and processes integrate different stream surface patches in parallel. Several strategies are applied to improve the overall performance of our algorithm. Loading data from other processes instead of from disk reduces the I/O cost. Cache is used to reduce the number of data loads. Runtime seeding curve segment subdivision and work stealing is used to improve the load balancing of our algorithm dynamically. Several experiments were conducted to study different parameters' influence on the performance of our algorithm and their relationships. Based on these experiments, we provided some guidance for parameter

setting. We also demonstrated the scalability of our algorithm up to a large number of processes.

In the future, we plan to extend our algorithm to unsteady flow fields. For time-varying flows, the memory requirement would increase because multiple time steps are involved. The constraint in available memory size will prevent us from loading all the data blocks to memory in the initialization stage, and hence a more complicated I/O strategy such as using the flow graph to predict the global flow behavior [21,22] and improve the I/O performance will be required.

Chapter 4: Explore Vector Fields with Distribution-based Streamline Analysis

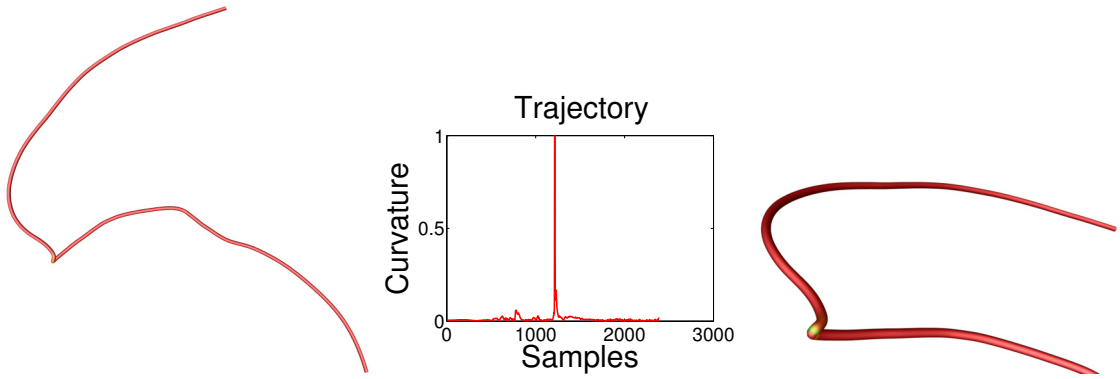
Visualization and exploration of vector fields plays an important role in understanding the data generated from simulations in many science and engineering disciplines. Among the various methods that are currently in use, streamline based techniques continue to play a central role in vector field exploration. When dealing with very large scale vector fields, computing a large number of streamlines is often necessary, where different shapes and orientations of the streamlines represent different flow features in the underlying field. A direct visualization of all the densely computed streamlines, however, is seldom useful due to the difficulties caused by visual cluttering and occlusion. In addition, the user is often more interested in specific flow features, for example, the occurrences of a specific type of vortex at a certain scale. Hence, it is important to be able to classify the streamlines based on their similarity and display only those that are relevant to the user's objective.

Streamlines can be classified based on the similarity in their shapes. Previously, several streamline similarity measures have been developed. A majority of these methods treat streamlines as a points trace and the similarity between streamlines is computed by the distance between the points, for example, the *mean of closest point distance* [25] and the *hausdorff distance* [86]. Although these methods are convenient and easy to understand, they are sensitive to translation and rotation since their distance measures are based on the

points' Euclidean distances. Given two streamlines, if they are translated or rotated individually, their distance will change. A different method to measure streamline distances is to calculate some geometric measures such as curvature or torsion along the streamline, and then combine these measures into a feature vector to describe the shape of the streamline. The similarity between streamlines is then measured through the distance between the feature vectors, for example, the edit distance [102, 104]. However, directly apply these similarity metrics usually bias to length. Streamlines that have a larger difference in length tends to have larger distance even though they may have similar shape. For example, two vortices with different sizes.

In this chapter, we propose a novel idea that uses the distribution of feature measures over a streamline to be the descriptor, and use this descriptor to measure the similarity between streamlines. Our research is motivated by the following observation: The features of a streamline may vary widely along the curve, leading to a wide variation in the measured values (Figure 4.1(a)). Also due to noises in the data and sharp turns or twists at certain locations, sudden fluctuations in the values of the measure may occur on the streamline (Figure 4.1(b)). However, individual low or high values do not necessarily represent a feature (Figure 4.1(c)). Hence, dealing with the measured values at each point, which is the finest level of granularity, often is not robust enough. Besides, point-wise distance calculations between two streamlines can be expensive since a streamline can easily contain hundreds of sample points, and position-based distance metrics are not invariant to translation and rotation.

In our framework, a streamline is divided into segments based on the feature's distribution along it. For each segment, a 1D histogram is constructed to represent its feature distribution and these 1D histograms are concatenated to produce a 2D histogram to



(a) (b) (c)

Figure 4.1: Limitation of point-based metric computation. (a) A streamline (color coded by curvature) with high range of curvature along the length. (b) The curvature values ordered along the length. (c) The zoomed in view of the point with highest curvature.

represent the entire streamline. Distances between the streamlines can be calculated by computing the distance between the 2D histograms. To match the segments between two streamlines that have different lengths or different numbers of segments, Dynamic Time Warping(DTW) is used. Using the distribution-based streamline distance metric has three major properties:

- Unlike point location-based distance metrics such as the *hausdorff distance* and the *mean of closest point distance*, the distributions of certain geometric properties on a streamline such as curvature and torsion are invariant to translation and rotation.

- Since our method is based on distributions, it does not need similar streamlines to have similar length. As long as they have similar distribution they will have small distance.

- Our method is much faster than other methods such as those that use the *hausdorff distance* and the *edit distance*. The timing performance of our method is summarized in section 4.3.

We demonstrate the utility of our approach with several examples of 3D flow field exploration. At run time, when the user finds a streamline with interesting feature, he can select this streamline and query for streamlines that have similar features. The similar streamlines will be extracted based on the distribution-based distance metric and rendered on the display window to highlight the feature of interest over the entire domain. Besides the interactive user-centric query, our system can create hierarchical streamline clusters for the user to explore particular flow features at different levels of detail. In all these applications, we show that the computation of streamline distance can be done very efficiently.

4.1 Method

The main idea behind the proposed method is to represent each streamline as one or more distributions of geometric measures. These distributions are used as the feature signatures to efficiently compute the similarity between any pair of streamlines. Given a large number of streamlines, the distributions can be used to compute clusters in an unsupervised manner. We illustrate the main steps of our framework in Figure 4.2.

4.1.1 Construction of Distribution From Streamlines

Given a user-specified measure, statistical distributions can be used to describe its values and spread along the trajectory of a streamline. In this chapter we demonstrate that several geometric measures such as curvature and torsion can be used to describe the feature of

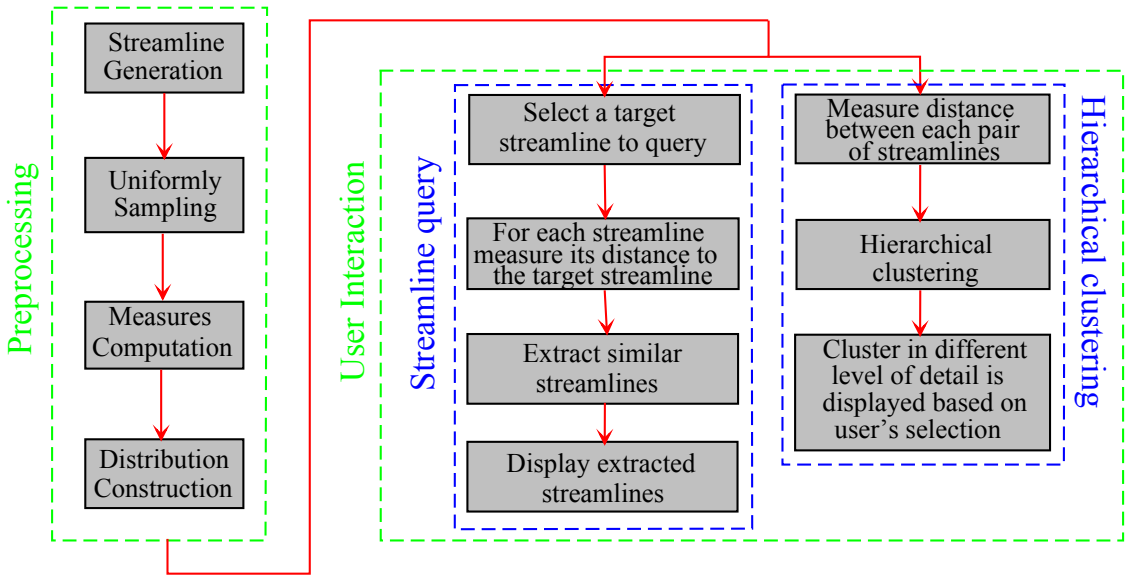


Figure 4.2: The major steps for our distribution-based flow analysis framework

streamline. When the streamline is sampled discretely, different distributions will be produced with different degrees of sample density on the streamline. In our implementation, we do evenly-spaced sampling and the distribution is constructed from the measurements collected along the streamline using the same non-adaptive step size as the Runge-Kutta 4th order numerical integrator.

Assuming k measures are selected for analysis, we compute each of them at every sample point s_p along a streamline, resulting in a k-tuple measure $\{m_1^p, m_2^p, \ldots, m_k^p\}$ for each point. For each measure, we normalize the value to 0 and 1 based on the range of the sample values on all streamlines. We compute a distribution separately from each measure, each of which either can be represented *analytically* by a set of statistical properties such as measures of central tendency, moments and central moments that characterize the distribution in a compact way, or can be expressed *empirically* by a histogram, which is a discrete

approximation of the underlying distribution. Analytical representations work well when the nature of the distribution is known. For example, a normal distribution can be represented by only its mean and standard deviation. However, the exact type of distribution for the geometric measures on a streamline is often unknown. In such cases, histogram based representation is more suitable, and hence is adopted in our method. Remember that we need to normalize the measurements to the range between 0 and 1. There is an issue when a few sample points having extreme values compared with other sample points, because the normalization will destroy the histogram. In order to address this issue, we clamp the top 5 percentage of measurements to 1 and then do normalization on the other 95 percentage of measurements in our implementation.

Histogram Construction

A histogram is a discrete representation of a probability distribution. Since the property of a streamline can vary from point to point, simply keeping a single 1D histogram may not be sufficient because the order of the measured values along the streamline can be essential for describing its feature and for comparing with other streamlines. For example, Figure 4.3(a) and 4.3(d) show two different streamlines color coded by curvature, one is more turbulent at the end and the other is turbulent at the beginning and the end. However, these two streamlines are undistinguishable by their 1D histograms since they are very similar, as shown in Figure 4.3(c) and 4.3(f).

To resolve the ambiguity, we use 2D histograms instead of the simple 1D histograms to represent a streamline. Given a streamline, it is divided into segments. The goal of dividing a streamline into multiple segments is to preserve the order of feature measured along a streamline, albeit loosely, while still being able to use histograms to describe a streamline. With the segmentation, now the value stored at each sample point s_p, is no longer a scalar

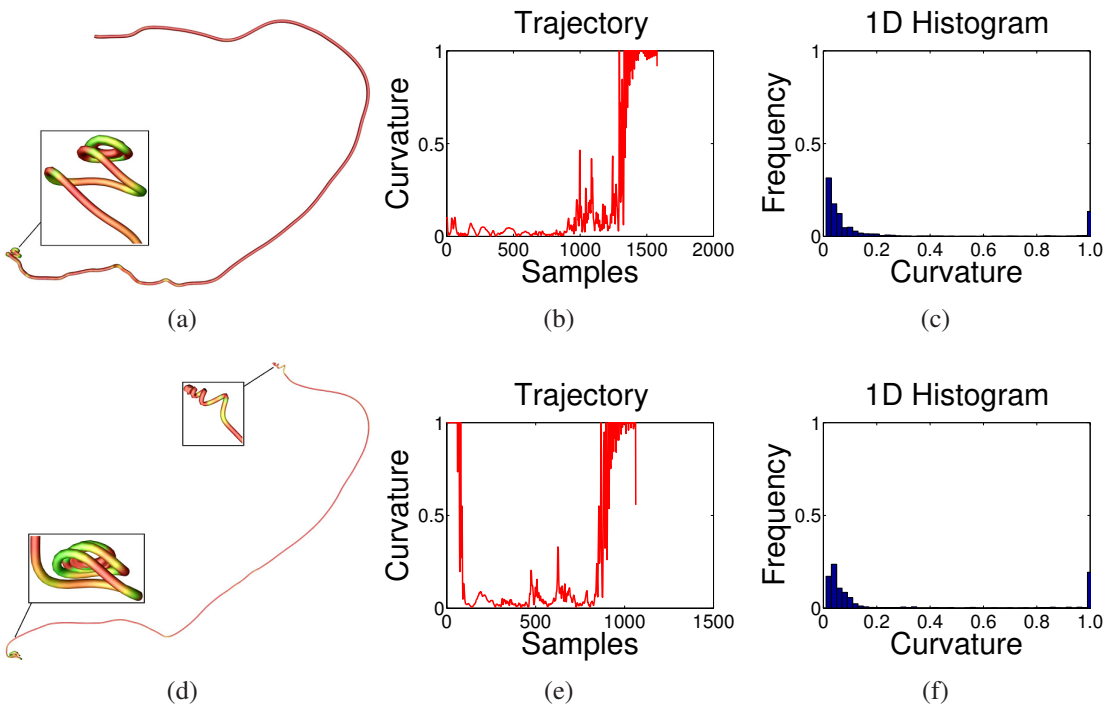


Figure 4.3: Limitation of the 1D histogram representation.

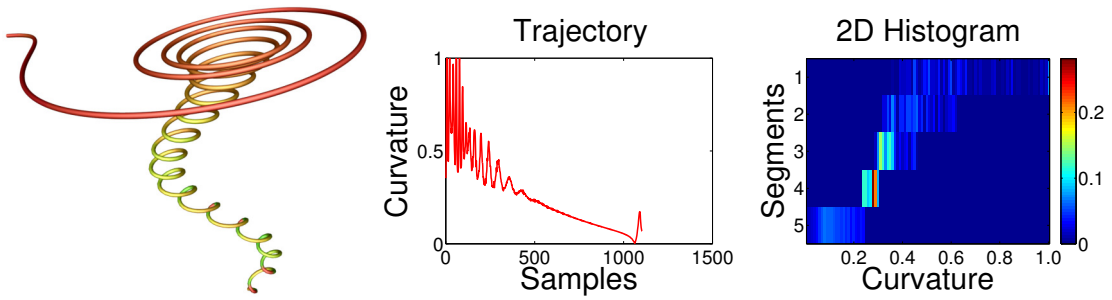
value, but a 2-tuple (x,k), where x denotes which segment this sample point is in and k represents the measurement. Then we can concatenate the 1D histograms of different segments in order, one in each row, and produce a 2D histogram representation for the entire streamline. The use of 2D histograms is a compromise between the use of point-based metrics and 1D histograms. At one extreme when every point represents a segment, the 2D histogram becomes a point-based metric. On the other extreme, if the entire streamline is treated as a segment, the 2D histogram becomes a 1D histogram. However, comparing with the point-based metric, computing the similarity based on 2D histograms is much faster and less sensitive to length, while comparing with the 1D histograms, 2D histograms preserve the order information.

An issue related to histogram construction is to decide the optimal number of bins (or the bin width) of each 1D histogram. In general, determining the optimal number of bins is non-trivial, and can be computationally expensive. However, there exist several empirical rules for this purpose. In our framework, we use Scott's choice [91] to decide the bin width for the 1D histogram within each segment of a 2D histogram. Scott's choice, decides the bin width based on the samples' standard deviation and the number of samples: $K = \frac{3.5\sigma}{N^{1/3}}$. where K is bin width, σ is the sample standard deviation, and N is the number of samples. Furthermore, we use the same number of bins for every segment, the reasons being:

- It speeds up the computation of distance between a pair of histograms, leading to interactive query processing and fast hierarchical clustering.
- It allows the user to compare the variances of features between segments by looking at the heatmap visualization of the 2D histogram.

In our implementation, the standard deviation σ is the average standard deviation of feature measures of all the segments and the sample size is the average sample points per segment: $K = \frac{3.5\sigma}{(\frac{N}{M})^{1/3}}$, where N is the total number of sample points, M is the total number of streamline segments.

An example of a 2D histogram constructed in this way is shown in Figure 4.4. Figure 4.4(c) shows the 2D histogram of curvature for the streamline shown in Figure 4.4(a). Based on the 2D histogram representation of the streamline, we can understand some underlying features about the streamline. For example, from Figure 4.4(c) the represented streamline has high curvature with high variation at the beginning, but it gradually decreases to low curvature. This can be observed from the curvature values ordered along the streamline, as shown in Figure 4.4(b). Since the 2D histogram incorporates the order



(a) (b) (c)

Figure 4.4: (a): The selected streamline color coded by curvature. (b): The curvature values ordered along the streamline. (c): The 2D histogram representation for this streamline, color represents frequency.

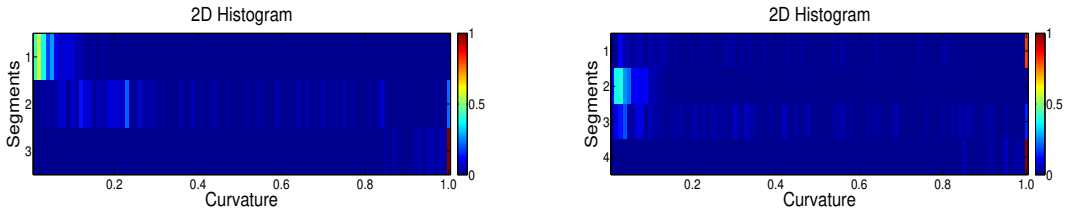


Figure 4.5: From left to right, the 2D histogram for the streamline in Figure 4.3(a) and (b)

information to some extent, it can better differentiate streamlines than simply using 1D histograms. The 2D histograms for the streamlines shown in Figure 4.3(a) and 4.3(d) are shown in Figure 4.5. Based on the two 2D histogram, it is easy to distinguish these two streamlines. The streamline shown in Figure 4.3(a) has high curvatures only at its end, while the one shown in Figure 4.3(d) has high curvatures at both the beginning and the end.

4.1.2 Streamline Segmentation

As discussed in the previous section, a streamline needs to be divided into segments to preserve the order of features measured along the flow direction. In order to preserve

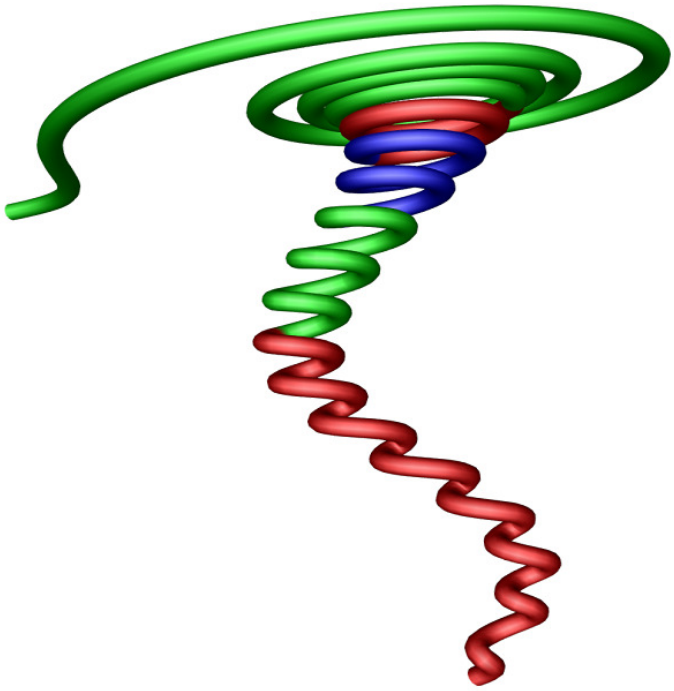


Figure 4.6: A segmentation result where segments are shown by colors.

the order information as much as possible, we do segmentation based on the difference between distributions of features in each possible segments, so that the features within each segment will be similar. To do this, given a streamline and a initial threshold ε, we choose a candidate split point where the difference in the distributions of the two streamline segments is the maximum. If the two distributions are bigger than the threshold ε, we split this streamline, increase the threshold by the initial threshold ε and continue to do so recursively on the resulting two halves. We terminate the segmentation process until the difference between two halves is smaller than ε or the streamline is too short to be segmented. The results of segmentation for two streamlines based on curvature are shown in Figures 4.6 and 4.7(a).

Our streamline segmentation process requires two parameters: one is the minimum length of streamline segments and the other parameter is a threshold ε used to decide whether or not we should split a streamline segment. The first parameter makes sure that for each segment we have enough sample points to construct a meaningful distribution. In our implementation, we empirically set it to 100. The second parameter decides how many segments we have for a streamline. The smaller this parameter is, the more segments we have for a streamline and the more features order information is preserved. The larger this parameter is, the fewer segments we have and then less features order information is preserved. In Figure 4.7 (a)-(c) we show the segmentation results based on three different thresholds, and the corresponding 2D histograms are shown in Figure 4.7 (d)-(f). Our similarity metric is not very sensitive to the number of segments. First, slightly change the threshold will not change the segmentation results too much; second, once the threshold is changed, every streamline will be affected and new segments are produced, hence, comparisons between any two of them will still remain valid; third, the streamline might be over-segmented due to small threshold as shown in Figure 4.7(a). However, when measuring the similarity, as described in Section 4.1.3, we use Dynamic Time Warping [6] to map the segments in two streamlines, which can still map the redundant segments and is thus less sensitive to over-segmentation.

4.1.3 Histogram Similarity Measure

With the 2D histogram computed from each streamline, we can compute the streamline similarity based on the distance between the histograms. We consider two streamlines to be similar to each other in term of the user specified feature if they have similar feature distributions. Recall that each 2D histogram from a streamline consists of a sequence of

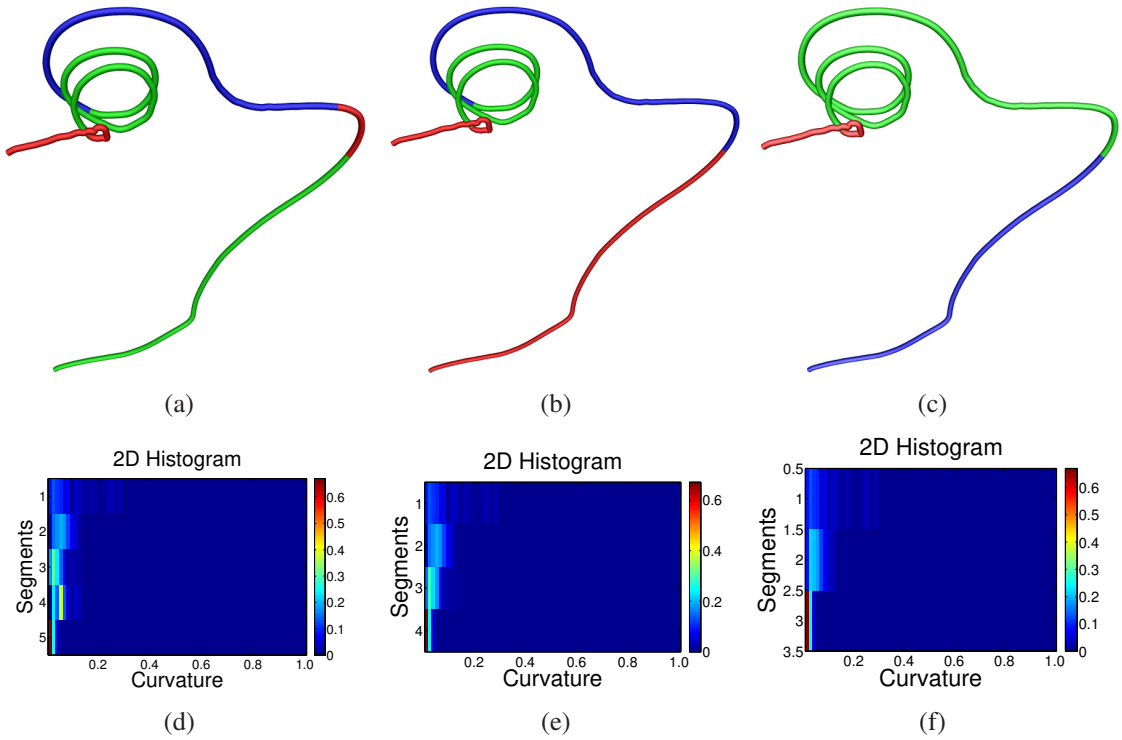


Figure 4.7: Segmentation results based on different threshold. From (a) to (c), the threshold is 0.1, 0.12 and 0.15. (d) to (f) show the corresponding 2D histograms.

1D histograms, one from each streamline segment. Because different streamlines may have different numbers of segments, i.e., different rows in the 2D histograms, a mapping between the segments is needed before calculating the streamline similarity. In our framework, we use Dynamic Time Warping(DTW) [6] to find the mapping between the segments in two streamlines.

DTW is an algorithm based on dynamic programming that finds an optimal mapping between two sequences. In our framework, we treat each streamline as a sequence of streamline segments. For each pair of segments, one from each streamline, the similarity between them is calculated as the distance between the two distributions (described below). From the distance between any pair of segments in the two streamlines, DTW computes

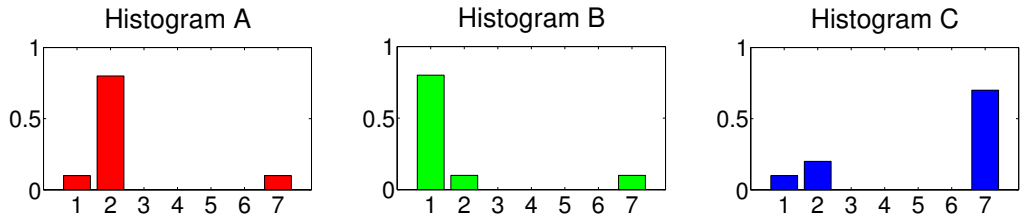


Figure 4.8: Three 1D histograms A, B and C for three different streamlines. H(A)=(0.1 0.8 0 0 0 0 0.1); H(B)=(0.8 0.1 0 0 0 0 0 0.1); H(C)=(0.1 0.2 0 0 0 0 0 0.7).

an optimal mapping between the two streamline sequences, where the cost is used as the distance between the streamlines.

There exist several standard methods to measure the distance between 1D histograms. Examples are L1 distance, Euclidean distance (L2), Kullback-Leibler distance(KL) [62], Bhattacharyya distance [7] and earth mover's distance(EMD) [88]. These distance measures can be classified into two categories: *bin-to-bin distance functions* and *cross-bin distance functions*. The bin-to-bin distance functions such as L1 distance, Euclidean distance, Kullback-Leibler distance and Bhattacharyya distance do not consider the correlation between non-corresponding bins when computing the distance between histograms. On the other hand, the cross-bin distance function such as the earth mover's distance considers the cross-bin relationship, albeit at higher computation cost.

In the context of our problem, considering cross-bin relationship is important. To show an example, assume that we have three 1D histograms for three streamlines as shown in Figure 4.8. Let A be the target histogram, the distance from B and C to A based on different distance functions is summarized in Table 4.1. It can be noted that all the bin-to-bin distance functions choose C to be more similar to A than B. Only EMD declares B to be more similar to A. Now, if these three histograms were representing curvature distribution

	L1	Euclidean	KL	Bhattacharyya	EMD
D(A,B)	1.4	0.99	1.46	0.41	0.7
D(A,C)	1.2	0.85	0.91	0.27	3.0
Similar one	C	C	C	C	B

Table 4.1: Comparison of different distance measurement functions

of three streamlines, both A and B would have segments dominated by low curvature and C would have segments dominated by higher curvature for a majority of points. This is why we consider Earth mover's distance to be more suitable for our framework and use it in our implementation.

Computation of EMD can be modeled as a classic optimization problem, known as supply-demand transportation problem, which tries to minimize the cost of converting one histogram into another by transporting units of mass from one to the other [88]. The transportation cost between two histograms P and Q can be expressed as:

$$EMD(P,Q) = \min_{\{F=f_{ij}\}} \frac{\sum_{i,j} f_{ij} d_{ij}}{\sum_{i,j} f_{ij}} \tag{4.1}$$

where d_{ij} is a pre-defined ground distance between supplier i and consumer j, and $F = \{f_{ij}\}$ is a set of flows which defines the amount of mass transferred from supplier i to consumer j.

Furthermore, since all the 1D histograms have cumulative histograms whose total areas will be the same, we can estimate EMD with much lower computation cost by the L1-distance between two 1D cumulative histograms [24]:

$$d(P,Q) = \sum_{i=0}^{n} |P_{cdf}(i) - Q_{cdf}(i)| \tag{4.2}$$

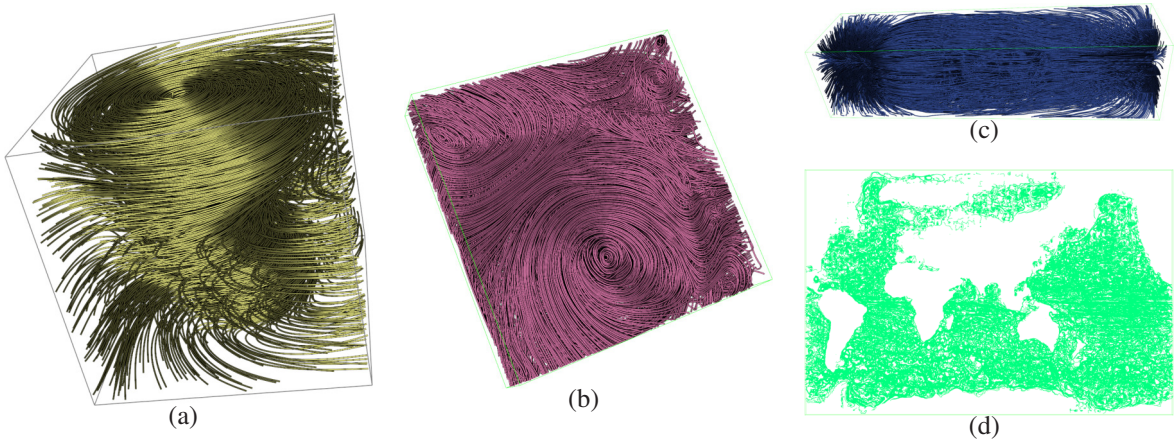


Figure 4.9: Visualization of the data sets Tornado, Hurricane Isabel, Solar Plume and Ocean. The number of testing streamlines are 1000, 2000, 2000, and 4800, respectively.

Given two streamlines $X = (x_1, x_2, ..., x_N)$ and $Y = (y_1, y_2, ..., y_M)$ where $x_1, ...x_N$ and $y_1, ..., y_M$ are the streamline segments. Suppose $V = (v_1, v_2, ..., v_L)$ where $v_l = (n_l, m_l) \in [1:N] \times [1:M]$ provides a mapping from X to Y. The distance between two 2D histogram H_1 and H_2 is:

$$Dist(H_1, H_2) = \min_{v}(\sum_{l=0}^{L} d(x_{n_l}, y_{m_l})) \tag{4.3}$$

which is computed using the DTW algorithm.

4.2 Interactive Visualization Framework Design

In this section, we describe the use of our interactive visualization and analysis framework based on the distribution-based approach for two different visualization applications: similar streamline query and hierarchical clustering. Several 3D flow field data sets are used, including Tornado, Hurricane Isabel, Solar Plume, and Ocean. Tornado is a $48 \times 48 \times 48$ synthetic data set. Hurricane Isabel is a data set with a resolution of $500 \times 500 \times 100$ that models a strong hurricane in the west Atlantic region in September

2003. Solar Plume was generated from a simulation of the solar plume on the surface of the sun with a resolution of $126 \times 126 \times 512$. The Ocean data set with a resolution of $3600 \times 2400 \times 40$ was produced by a high resolution eddy resolving simulation. We place a large number of seeds in the domain to generate streamlines, and collect distributions from the streamlines to analyze the flow features. Visualizations generated from these data sets are shown in Figure 4.9.

4.2.1 Feature Selection

The distribution-based approach presented in this chapter can work with any suitable feature measure or a combination of measures based on the underlying application. For example, when the user is looking for some specific type of feature such as vortices, feature-specific metrics such as winding number [84], helicity [29] and $Lambda_2$ [50] can be used. When the user wants to identify streamlines with similar shapes, curvature and torsion can be used since according to the fundamental theorem of space curves, two space curves, streamlines in our case, defined on the same interval with the same torsion and non-zero curvatures can be translated into one another by application of an Euclidean transformation [61]. In this section, we use curvature, torsion, and curl as the measures to demonstrate our distribution-based approach.

Curvature

The curvature describes the rate of change in the tangent vector over a space curve with respect to the length of arc. Let T denote the unit tangent vector at a point on a streamline and S denotes the length of arc, the curvature κ is the magnitude of the rate of change of T:

$$\kappa = \|\frac{dT}{dS}\| \tag{4.4}$$

Curl

Given a vector field V, curl is denoted as $\nabla \times V$. Curl is a vector measure of the rotation at a point P in the vector field. The vector itself represents the axis of the rotation, and the magnitude of the vector denotes how fast it rotates. Only the magnitude of curl is used in our case since we are more interested in the tendency of circulation in a flow field. The curl of a point in vector field V can be defined as:

$$\nabla \times V = (\frac{\partial V_z}{\partial y} - \frac{\partial V_y}{\partial z})\mathbf{i} + (\frac{\partial V_x}{\partial z} - \frac{\partial V_z}{\partial x})\mathbf{j} + (\frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y})\mathbf{k} \tag{4.5}$$

Torsion

The torsion of a streamline measures the degree of twisting around its binormal vector. It also describes the rate of change of the streamline's osculating plane. If the torsion is zero, it means there is no twisting around the binormal vector, and the streamline completely lies on the osculating plane defined by the tangent and normal vectors. The torsion τ of a streamline is given by: $\tau = -N \cdot B'$; $B = T \times N$, where N, T and B are the normal, tangent and binormal vectors respectively and B' is the derivative of the unit binormal vector.

4.2.2 Similar Streamline Query

Visualizing three-dimensional streamlines can be difficult due to occlusion and visual cluttering. To explore a large number of pre-computed streamlines, users can request to display only streamlines that have a similar shape to that of the target streamline, for example, a streamline with a high degree of swirl that the user spots during exploration. To support this, we can perform a similarity query where all streamlines are compared with the target streamline. Since the user can frequently change the target streamline to look

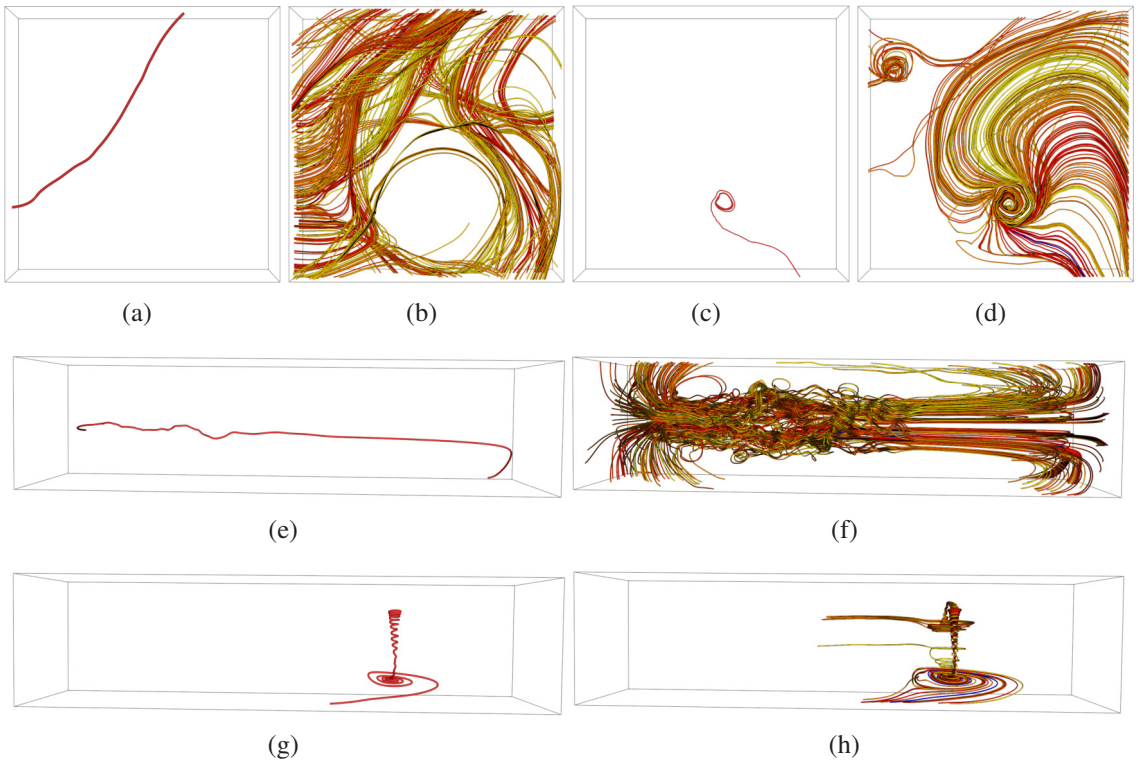


Figure 4.10: Four query results for Isabel data and Plume data. (a),(c),(e) and (g) show the four target streamlines. (b),(d),(f) and (h) show the query results for (a),(c),(e) and (g) respectively. In the results, streamlines are colored from red to yellow where red means more similar and the target streamline is in blue.

for different features, such distance comparisons must be done fast. In this section, we study the effect of using our distribution-based distance metric described in section 4.1.3 to perform streamline query. Performance results are provided in section 4.3.

As mentioned earlier, curvature and torsion can be used to characterize space curves [61]. In our studies, we use a combination of curvature and torsion as the feature to query streamlines. To compute the distance between two given streamlines, we first compute the distance between their curvature histograms and the distance between their torsion histograms.

Then we sum these two quantities to get the final distance. Figure 4.10 shows four different query results. In our first example, we use a streamline with low curvature and low torsion in the Isabel data set as the target streamline, shown in Figure 4.10(a). During the exploration, user can use a slide bar to show top N similar streamlines. In Figure 4.10(b), the top 400 most similar streamlines are shown, where red means more similar and yellow means less similar. For the second query, we use a streamline around the hurricane eye in Isabel as the target. The top 200 streamlines are selected and these streamlines are shown in Figure 4.10(d). The most similar streamlines are all around the hurricane eye and another swirl area in top left is detected. In the third query, the target is a straight streamline passing through the center of the Plume data as shown in figure 4.10(e). The top 200 similar streamlines are shown in Figure 4.10(f). A highly swirling streamline in Plume as shown in Figure 4.10(g) is selected as the target in the last query. The top 20 similar streamlines are selected and shown in Figure 4.10(h).

These four examples show the effectiveness of our query on streamlines with different natures such as straight, medium swirl and highly swirl. By using the combination of curvature and torsion, we are able to find streamlines with similar patterns as the target. In our system, the user can select any interesting streamline he observes to query during exploration, and the distances between the target and all other streamlines will be calculated. The user can use a slider bar to show the top N similar streamlines to explore the flow field. Our query can also be used to extract features such as swirl and vortex in a flow field. Figure 4.11 shows an example of using our query method to extract swirling flow by using curvature and torsion. The target streamline is a streamline with a highly swirl part as shown in Figure 4.11(a). The top 500 similar streamlines are displayed in Figure 4.11(b). The blue streamline is the target and another five big and one small swirling areas in the

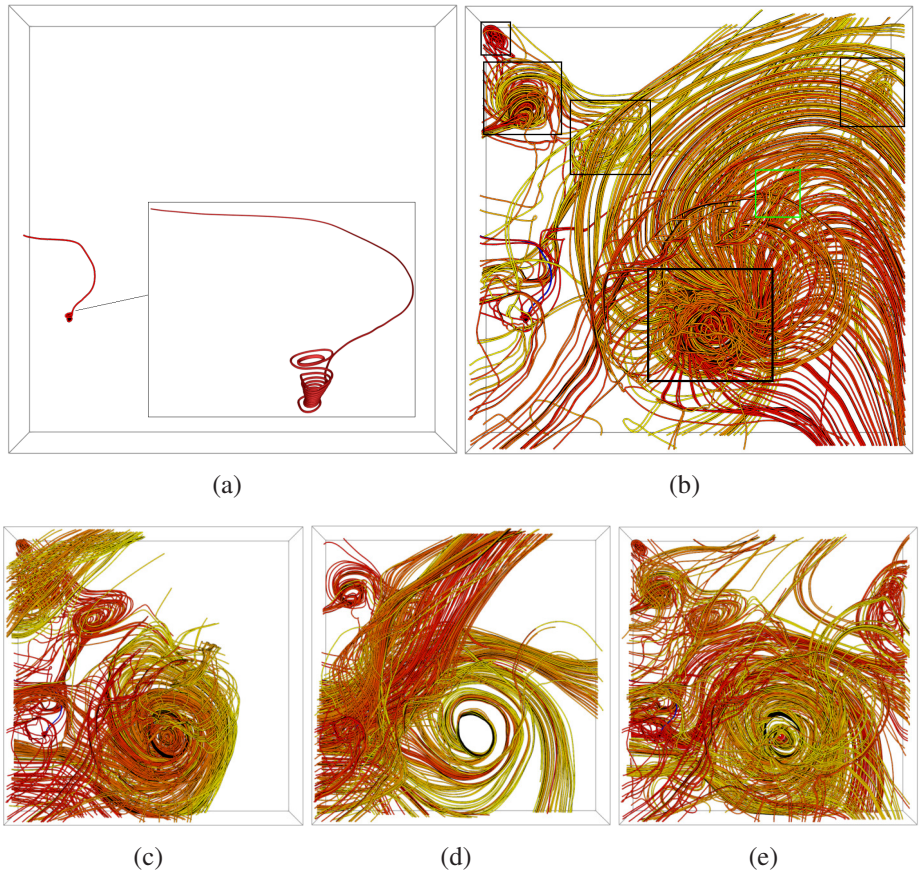


Figure 4.11: (a): The target streamline with a highly swirling part in the end. (b): The top 500 similar streamlines based on our distribution method. Five big swirling areas highlighted by black squares and one small swirling area highlighted by a green square are extracted. (c),(d) and (e) show the top 500 similar streamlines based on the *hausdorff*, *mean of closest point* and *edit distance*.

flow field are extracted. In Figure 4.11(c), 4.11(d) and 4.11(e), we show query results based on the *hausdorff*, *mean of closest point* and *edit distance* [104]. The point location based metrics such as the *hausdorff* and the *mean of closest point distance* tends to extract streamlines which are closed to the target streamline in space but not necessarily focus on the similarity between streamlines' features. The *edit distance* founds the similar features

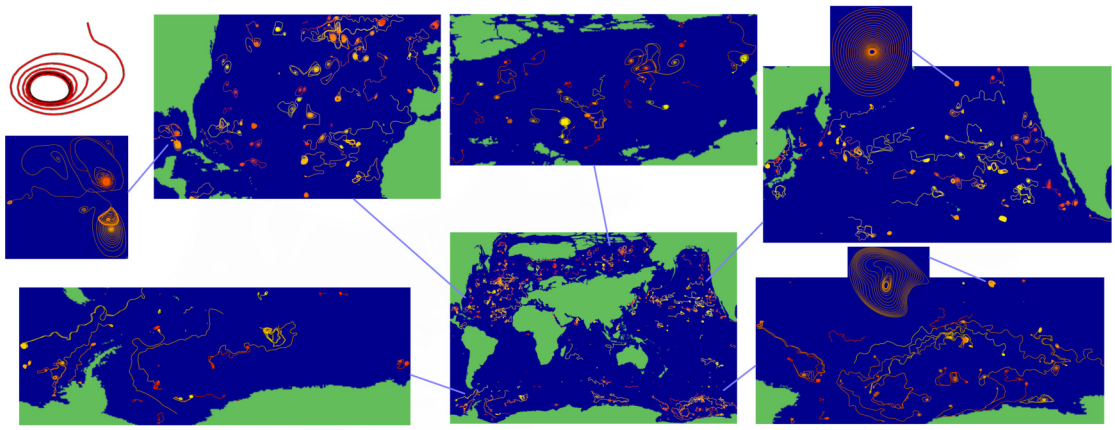


Figure 4.12: Effectiveness of our streamline query. The top left corner shows the target streamline which indicates an vortex. By querying similar streamlines, lots of vortices are found in the Ocean data set.

in the vector field as our method, but it is much slower than our method. The timing performance is reported in Table 4.3. Figure 4.12 shows another example using the Ocean data set.

4.2.3 Hierarchical Streamline Clustering

In addition to streamline query, another way to explore flow fields is through hierarchical streamline clustering. We use a bottom up agglomerative clustering method in our implementation. We begin with the bottom level where each cluster contains only one single streamline. Then at each step, we merge two clusters with a minimum distance until there is only one cluster left and a binary tree representation of the hierarchical cluster is generated. Different methods exist to calculate the distance between clusters such as complete-linkage clustering, single-linkage clustering and the mean distance between elements in each cluster. Complete-linkage clustering takes the maximum pair distance between elements in two clusters. Single-linkage clustering takes the minimum pair distance between elements in

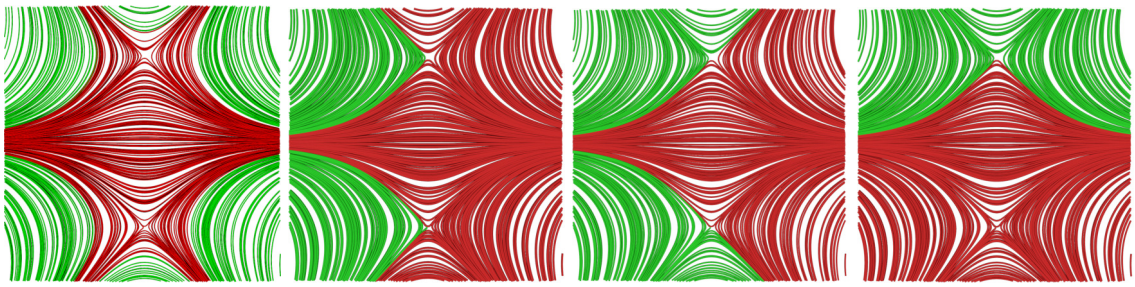
two clusters. The mean distance takes the average pair distance between elements in two clusters. Suppose P and Q are two clusters, these distances are defined as following:

- Complete-linkage: $max\{d(x,y) : x \in P, y \in Q\}$.
- Single-linkage: $min\{d(x,y) : x \in P, y \in Q\}$.
- mean distance: $\frac{1}{|A|\cdot|B|}\sum_{x\in Q}\sum_{y\in P} d(x,y)$.

Our experiments show that directly applying one of these three distance metrics to compute distances between clusters may produce unbalanced binary trees, in the sense that the sizes of clusters at a particular level may differ a lot. To avoid this problem, when computing the distance between two clusters, we add one more term to penalize a merge operation when it generates an unbalanced binary tree. The new formula to compute the distance between two clusters P and Q is:

$$dist(P,Q) = d(P,Q) + w\frac{s_P + s_Q}{S} \tag{4.6}$$

where the first term $d(P,Q)$ is the distance calculated by using one of the three methods discussed above and normalized to be 0 to 1. In the second term, s_P and s_Q are the numbers of streamlines in cluster P and Q and S is the total number of streamlines. w is the weighting parameter between 0 and 1. We call w the balance parameter. When w is larger, the smaller size clusters are forced to be merged early and the final clustering result will be more balanced. When w is smaller, small size clusters which correspond to small features in the data are more likely to be shown in the final clustering result. In our system, user can change w during the exploration to experiment clustering results with different degrees of balance.



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

Figure 4.13: Hierarchical clustering results based on different distance metrics, from left to right: our method, *hausdorff*, *mean of closest point* and *end point distance*. The balance parameter $w = 0.5$. Different clusters are assigned a different color.

To demonstrate the effect of hierarchical clustering based on our distance measures, we start with a simple 2D vector field. Curvature is used as the feature descriptor of streamlines in 2D. To evaluate our distribution-based approach, we also show the hierarchical clustering results using the *hausdorff distance*, *mean of closest point distance*, and *end point distance* respectively. Figure 4.13 shows the hierarchical clustering results. This example also demonstrates one advantage of our distance measure over other traditional point location based distance measures, that is, our method is invariant to translation and rotation of individual streamlines, and mainly focused on the intrinsic nature of shape. Using other metrics, streamlines with similar shapes but different orientations and far from each other will not be grouped into the same cluster, for example, the streamlines in the four corners of the data in the image. In Figure 4.14 we show the hierarchical clustering result for another 2D vector field based on curvature distribution. In this example, vortex and straight flow are separated and assigned to two different clusters since they have different curvature distribution.

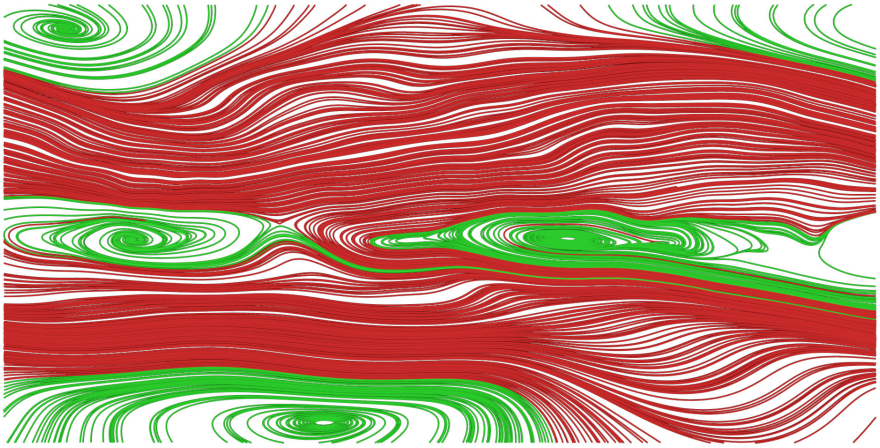
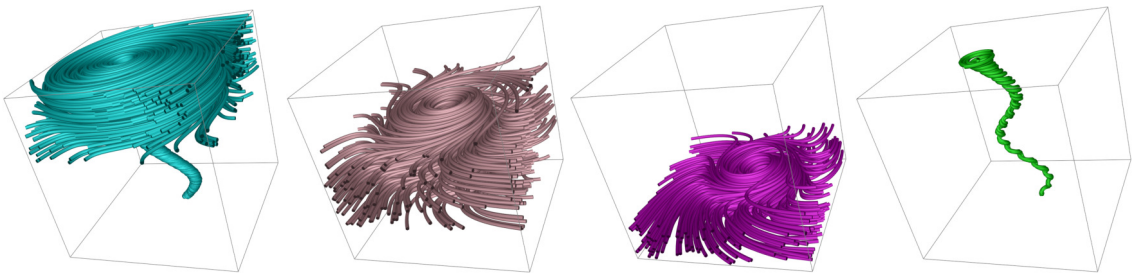


Figure 4.14: Clustering results based on curvature distribution. The balance parameter $w = 0.7$. The green cluster corresponds the vortex flow and the red one corresponds to straight flow.

We also perform experiments on the Tornado data set. The measure used is the magnitude of curl which represents how fast the flow rotates in the field. Based on this measure, tornado can be separated to different parts with different degrees of rotation. Figure 4.15 shows four clusters. We set the balance parameter w to be 0 during the exploration, so that a small cluster corresponding to the region around the tornado center is extracted as shown in Figure 4.15(d).

Besides the 2D flow fields and the simple synthetic 3D flow field, we also tried our hierarchical clustering method on the Plume data set, a more complex 3D flow field. The combination of curvature and torsion is used to cluster the streamlines. The result is shown in Figure 4.16. The whole set of streamlines are grouped into several clusters and each cluster has streamlines with similar shapes. By displaying each cluster separately, the user is able to inspect different part of the flow field and understand the data more easily.



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

Figure 4.15: Hierarchical streamline clustering results for Tornado data set based on the magnitude of curl. The balance parameter $w = 0.0$. The whole set of streamlines are cut into four parts with different swirling intensity.

4.3 Performance

Data Set	Avg. Len.	Advect. (sec)	Feature Comp. (sec)			Seg. (sec)
			Curv.	Curl	Tors.	
Tornado	725	5	4	3	38	55
Isabel	2087	36	21	18	61	232
Plume	2211	33	24	22	153	707
Ocean	2919	60	79	67	596	3495

Table 4.2: Timings for Streamline Computation and Feature Evaluation

Our streamline exploration framework consists of two main stages. The first stage is preprocessing, and can be further divided into four steps: streamline generation, segmentation, feature evaluation, and histogram construction. Currently, we evaluate the three geometric features: curvature, curl and torsion for every sample point. The second stage of the framework is interactive visualization, and it includes similar streamline query and hierarchical streamline clustering. Streamline computation, segmentation and feature evaluation

were done on a Linux server with an Intel Xeon CPU and 24 GB of memory. Histogram construction and interactive visualization were performed on a desktop computer with an Intel(R) Core(TM) i7-2600 CPU 3.4GHz processor with 16GB memory. The performance numbers for streamline computation, segmentation and feature measurements are summarized in Table 4.2.

In Table 4.3, we summarize the timing of streamline query for Hurricane Isabel (Figure 4.11), Plume (Figure 4.10(g)) and Ocean (Figure 4.12) data sets. All the queries used a combination of curvature and torsion. The computational cost for our streamline query depends on the number of streamlines, the number of segments, and the number of bins for the histogram in each segment. We compare the performance of our distribution-based method with other distance measures such as the *hausdorff distance* d_h, the *mean of closest point distance* d_m and the *edit distance* based on curvature and torsion d_e. It can be seen that our method outperformed the other methods by several times in terms of speed. This is important for the users to get a rapid feedback for their queries.

	# Bins		Timing(sec)				
Data Set	Curv.	Tors.	Hist.	Ours	d_h	d_m	d_e
Isabel	43	20	0.28	0.02	281	287	270
Plume	18	8	0.30	0.02	340	351	307
Ocean	12	1	0.96	0.03	561	575	380

Table 4.3: Timings for Streamline Query

Table 4.4 shows the timings for hierarchical streamline clustering based on our method for Tornado(Figure 4.15) and Plume(Figure 4.16). All timings were measured in seconds. For Tornado, we use the curl to separate streamlines based on the degree of rotation. For

Data Set	# Bins			Timing(sec)		
	Curv.	Tors.	Curl	Hist.	Dist. Matrix	Clust.
Tornado	N/A	N/A	7	0.06	1.62	2.92
Plume	18	8	N/A	0.31	12.42	22.39

Table 4.4: Timings for Hierarchical Streamline Clustering

Plume, we use a combination of curvature and torsion. We did not measure the performance of hierarchical clustering based on other distance measures such as the *hausdorff distance* since the computational time would be prohibitively long as we can see from the query results.

4.4 Conclusion and Future Work

In this chapter, we propose a new method to measure the distance between streamlines based on the statistical distributions of geometric measures along the trajectories of streamlines. Compared to some existing methods, our method is invariant to translation and rotation, and we can evaluate the distance between streamlines much faster. Based on our distance metric, we design a framework to interactively explore 3D vector fields through query and clustering.

Our framework can be extended along several directions. First, we will extend our framework to handle time-varying vector data, so that query and clustering can be done for pathlines or streaklines. Second, besides the three geometric measures, curvature, curl and torsion, additional measures including domain-specific physical quantities can be included to explore 3D vector fields. We believe that combining different feature measures can produce more robust query and clustering results, and reveal different features in the vector

field. Third, in order not to miss any important features of the flow and make the query more efficiency, our method can be combined with feature-driven streamline placement. Fourth, the user interface could be further improved. For example, instead of showing a given number of similar streamlines, the user can choose to show the streamlines whose distance to the target streamline is within a threshold, also we could provide some heuristic such as 1D histograms to assist the user to select target streamlines. Fifth, in the hierarchical clustering, we use a balance parameter and the size of clusters to control the clustering results. In the future, other metrics will be investigated to control the clustering results. Furthermore, we will investigate the effect of constructing 2D histograms with different bin size for different streamlines and study how it influences the results and performance.

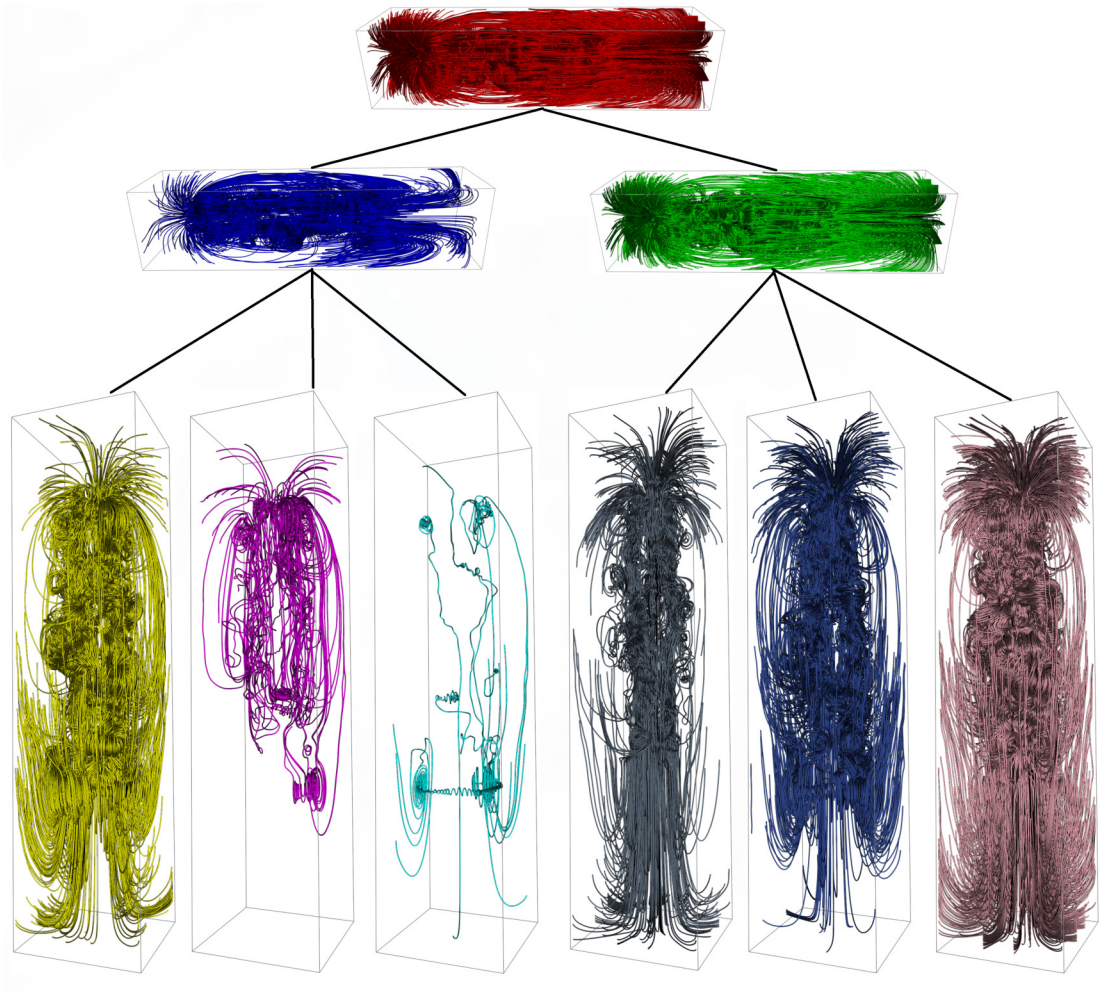


Figure 4.16: Hierarchical streamline clustering results for Plume data set based on the combination of curvature and torsion. The balance parameter $w = 0.7$. The whole set of streamlines are cut into different parts based on their shape difference.

Chapter 5: A Compact Multivariate Histogram Representation for Query-driven Visualization

Distributions play an important role in analyzing and visualizing data generated from various scientific and engineering simulations, as they are particularly effective for data aggregation, summarization, and query. Previously, locally and globally computed distributions have been used in transfer function design for volume rendering [56, 65]. Given the distributions computed from a multivariate dataset, different statistical metrics such as mean, variance and information entropy can be computed to facilitate the analysis and visualization of the data. Point-wise distributions have been used for flow visualization based on information theory [110]. Several query-driven visualization techniques also utilize distributions. By querying regions that contain a certain type of distribution, salient features can be identified from the volume data [52]. In [38], Gosink et al. showed distributions are useful for segmentation and extraction of salient features. Block level distributions are used to track features in time varying data in [39]. Distributions have also been used to predict isosurface statistics [5, 17]. In recent years, the rapid increase in the speed of processors and the number of processing cores empowers scientific simulations to run at unprecedented spatial and temporal resolutions, which in turn produce very large scale datasets. As a result, it is expected that distributions will play an increasingly more important role in supporting the need of large scale data analytics.

Data distributions can be represented using parametric or non-parametric models. For parametric models, it is assumed that data come from a particular type of distribution, which can be described by just a few parameters. Non-parametric models, on the other hand, make no assumptions about the data and hence can model a wider range of distributions. A popular non-parametric model for distributions is the histogram, which is commonly used because the distribution model for data generated from a simulation is often unknown. To support various analysis needs, there have been works on scalable computation of univariate histograms at different levels of detail from large scale data [19] and efficient range distribution query using integral histograms [20, 63, 69]. Most of the works focused on computing 1D histograms from univariate data, while relatively little research has been done for computation and storage of histograms for multivariate datasets.

Multivariate datasets are frequently encountered in scientific applications. Compared to univariate data, analyzing and visualizing multivariate datasets is much more challenging and hence remains an active topic of research. Computing and storing multivariate histograms is nontrivial because the memory requirement can grow exponentially with respect to the number of variables if stored as multi-dimensional arrays. Furthermore, different analysis tasks may require multivariate histograms that have different combinations of variables with different number of bins. Because of the computation and storage cost, computing multivariate histograms from the raw data at run time is often very expensive.

To solve aforementioned problems, in this chapter, we present a novel technique to store multivariate histograms using their sparse property to reduce the storage cost. Naturally, a multivariate histogram can be represented as a multi-dimensional array. We first transform the large multi-dimensional array to an array of much smaller size based on the sparseness of multivariate histograms. Dictionaries are constructed to encode this transformation and

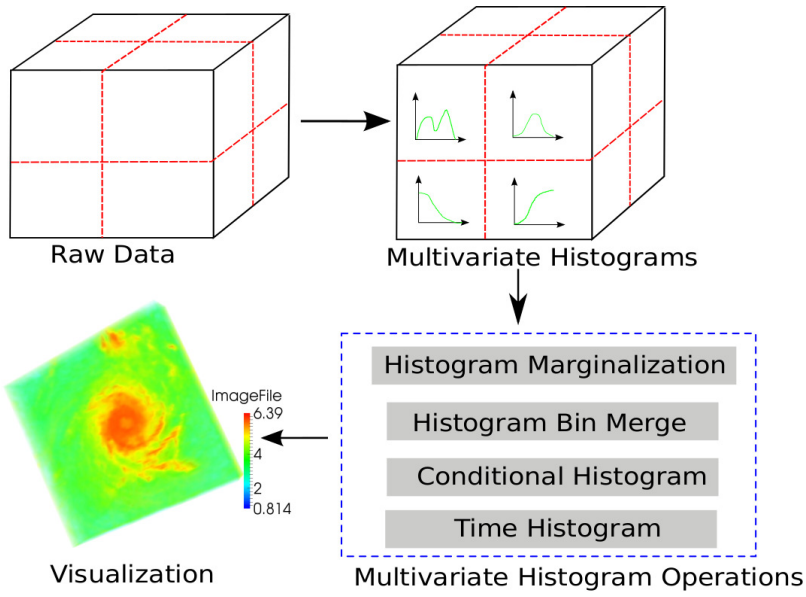


Figure 5.1: Overview of our processing pipeline.

can be used to map the transformed multi-dimensional array back to the original array. Then, we represent the multivariate histogram as a sequence of index and frequency pairs where the indices are represented as bitstrings computed from a space filling curve traversal of the multi-dimensional array. To support different types of applications, we demonstrate several histogram query operations such as marginalization, histogram bin-merging, conditional histogram and time histogram query using our representation. We also present several applications to analyze and visualize multivariate datasets.

Figure 5.1 shows an overview of our processing pipeline. In the data preparation stage, the original raw data is divided into blocks and then a multivariate histogram is computed for each block. A compact data structure, presented in Section 5.1, is used to store the multivariate histograms with reduced storage cost. Then, several query operations are available

to derive other histograms efficiently in the query stage. After the user gets the desired multivariate histograms, different query-driven analysis and visualization can be performed.

5.1 Multivariate Histogram Representation

A nontrivial problem related to multivariate histogram computation is how to represent and store the multivariate histogram. A straightforward way is to store the multivariate histogram as a multi-dimensional array. However, the memory requirement of using a multi-dimensional array would increase considerably as the number of variables and bins increase. Suppose that there are N variables, and we use the same number of bins for all dimensions and the number is B, if the frequencies of the multivariate histogram are stored as floating point numbers, we will need B^N floating point numbers to represent the multivariate histogram. The memory cost grows exponentially with respect to the number of variables.

To reduce the memory requirement for storing a multivariate histogram, we use the sparseness property of multivariate histograms. Given a dataset with P data points, at most P entries in the multivariate histogram have non-zero frequencies while all the other entries are zeros. For large scale datasets, if we divide the entire dataset into smaller blocks and compute a multivariate histogram for each block, the number of data points per data block is even smaller than the number of entries in the multivariate histogram. Considering a block with size of 8^3 and each data point has 3 attributes, if we construct a multivariate histogram with 256 bins for each dimension, at most $8^3/256^3 \approx 0.0031\%$ entries of the multivariate histogram are not zeros.

5.1.1 Data Space Transformation

Given this super-sparse block-wise multivariate histogram, a data space transformation is employed first to reduce the size of the multivariate histogram. The main motivation of this data space transformation comes from the following observation:

- For scientific multivariate datasets, given a block, the value for an arbitrary variable within that block changes smoothly. Also because all block-wise multivariate histograms are computed using the same global value range for the same variable, for an arbitrary dimension, those non-empty entries usually locate in a smaller number of bins along that dimension instead of spreading out a large number of bins along that dimension.

Table 5.1 shows the average number of distinct index values for each variable collected from a number of block-wise multivariate histograms that are computed for each dataset. Three datasets and five variables for each dataset are used. The number of bins are set to 256 for each variable. As it shows, for all those variables, the number of distinct index values is much smaller than 256. Through data space transformation, the extremely large and sparse multi-dimensional array can be converted to a much smaller and denser multi-dimensional array which requires much smaller space compared with the original multi-dimensional array.

Data	V_0	V_1	V_2	V_3	V_4
Isabel	10.4	10.0	5.8	3.2	4.1
Combustion	5.8	12.0	11.1	12.5	9.7
Ion Front	9.9	10.7	1.9	10.7	13.3

Table 5.1: Average Number of Distinct Bin Index for each variable

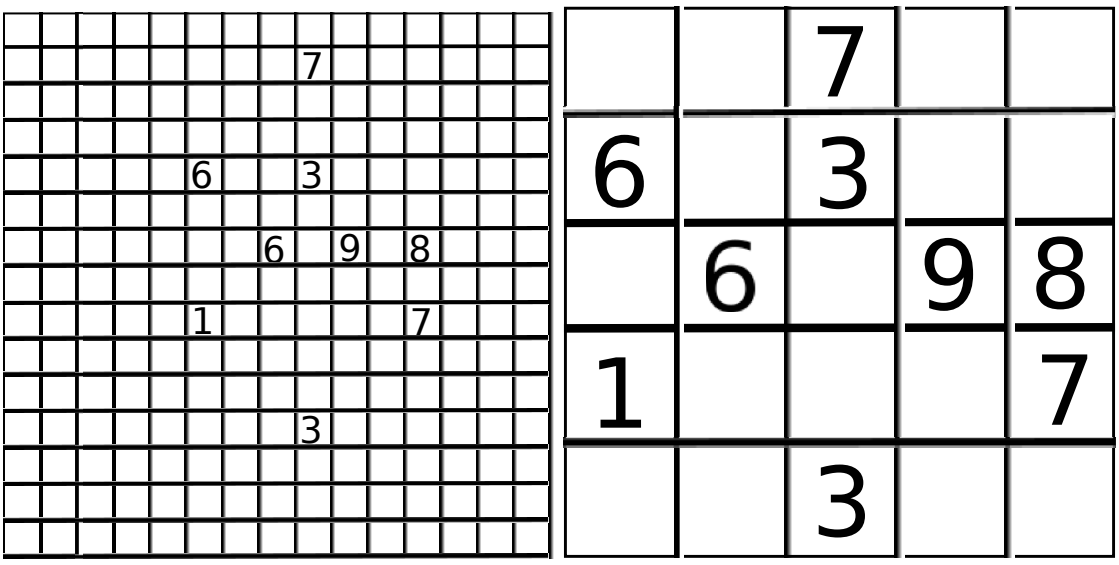
Algorithm 5 Data Space Transformation

```
1: Let N be the number of variables, B_i be the number of bins for the i_th variable, D_i is the
   constructed dictionary for the i_th variable, bin(i) be the bin index for the i_th variable.
2: for i:=1 to N step 1 do
3:    Initialize idx=0
4:    for j:=1 to B_i step 1 do
5:       if All those histogram entries with bin(i)=j are empty then
6:          Remove those histogram entries with bin(i)=j
7:       else
8:          Insert (idx, j) to D_i
9:          idx++
10:      end if
11:   end for
12: end for
```

Let $V = V_0, V_1, ..., V_{N-1}$ denote the set of variables that the multivariate histogram is computed from and B_i represents the number of bins for the i_{th} variable. Then, the multivariate histogram can be represented as a N-dimensional array M_O with size $B_0 \times B_1 \times \cdots \times B_{N-1}$. The data space transformation operation transforms M_O to another N-dimensional array M_T with size $P_0 \times P_1 \times \cdots \times P_{N-1}$, where usually $P_i \ll B_i$ for $0 \leq i \leq N-1$.

A dictionary is constructed to record the transformation for each variable. The dictionary provides an one to one mapping for each variable from the bin index computed from M_T to the bin index computed from M_O. Let D_i denote the dictionary constructed for the i_{th} variable, given a bin index $j \in [0, P_i - 1]$, D_i maps j to a unique value $k \in [0, B_i - 1]$ which is the original bin index, we refer this step as decoding.

Given a multi-dimensional array, to perform transformations and generate dictionaries for each variable, for each variable V_i, let $bin(V_i)$ denote its bin index, for every $bin(V_i) \in [0, B_i]$, if all those histogram entries with the bin index $bin(V_i)$ have zero frequency, we remove all those entries, otherwise, we insert a new entry to the dictionary for the variable V_i. The pseudocode for this algorithm is shown in Algorithm 5. After the transformation,



(a) (b)

Figure 5.2: A multivariate histogram with two variables, each with 15 bins. (a) The original multidimensional array. (b) The transformed multidimensional array.

we have a much smaller array to be indexed. Compared with directly indexing the original multi-dimensional array, indexing the transformed multi-dimensional array requires less number of bits so that reduce the storage overhead. A data space transformation example is shown in Figure 5.2.

5.1.2 Multivariate Histogram Construction

After the data space transformation, we represent the resulting multi-dimensional array as a collection of index and frequency pairs, where each pair corresponds to a nonempty entry in the multi-dimensional array. Similar to the method presented in [18], we also use space filling curve to index the data space. However, before the indexing, we pad the resulting multi-dimensional array from data space transformation to be power of two along all its dimensions, so that we could directly use bit concatenation or bit interleaving to

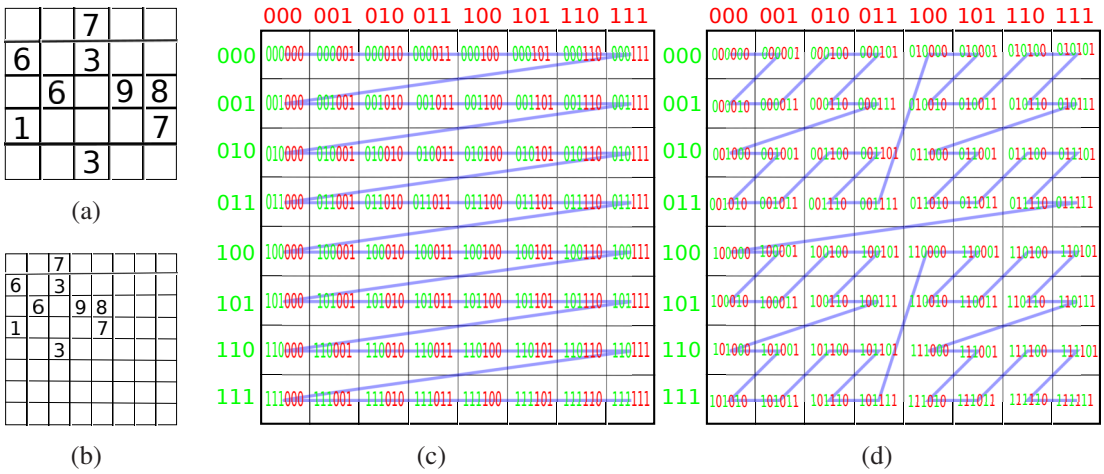


Figure 5.3: (a) The multidimensional array with size 5X5. (b) Non power of 2 dimensions are padded to power of 2. (c) Bit concatenation is used when sweep space filling curve is used. (d) Bit interleaving is used when Z-order space filling curve is used.

compute the index based on whether we use sweep or Z-order space filling curve to index the data space. A 2D example is shown in Figure 5.3. When the sweep space filling curve is used, the computed indices is equivalent to using BESS [37] to index the transformed array without chunking.

The main benefit of this padding step is that after padding, different bits of the index correspond to the bin indices of different variables, so that we could directly manipulate individual bits to operate the bin index of different variables. Since many histogram queries require the bin indices of different variables, this feature allows us to perform those histogram queries with efficient bit-wise operations.

In our representation, a multivariate histogram is represented as dictionaries which are used to record the result of data space transformation, and the index and frequency pairs. In this representation, an index can be decomposed into different parts where each part represents the bin index of a variable after the data space transformation. Hereafter we refer

Index	Frequency
1 8	7
6 7	6
6 9	9
11 8	3
...	...

Vector Representation

Index	Frequency
23(00010111)	7
97(01100001)	6
99(01100011)	9
173(10101101)	3
...	...

SFC Curvilinear abscissa Representation

Index	Frequency
000010	7
010001	6
010011	9
100010	3
...	...

Our Representation Sweep SFC

Index	Frequency
000100	7
001001	6
001101	9
100100	3
...	...

Our Representation Z-order Curve

Figure 5.4: Different representations.

to the bin index that we get from the transformed multi-dimensional array as *transformed bin index*. The dictionary can be used to map the transformed bin index to the bin index computed from the original data space. We refer to the bin index computed from the original multi-dimensional array as *original bin index*. An example of different representations for the multivariate histogram is shown in Figure 5.4. In our representation, the bits corresponding to the first variable are colored red and the bits corresponding to the second variable are colored green. We could manipulate these bits to operate the bin index for different variables. In Section 5.2 we presented several common query operations to obtain different types of histograms by manipulating those bits.

There are several benefits of using our representation:

- Compared with the vector and space filling curve curvilinear abscissa representations, the storage cost of our representation is reduced, because we first do a data space transformation to reduce the size of the multi-dimensional array dramatically and then use a space filling curve to index the transformed multidimensional array. Fewer bits are used to represent the index. Even though we keep dictionaries to record the transformation, the overall storage cost is still reduced in general.

- Compared with Goil's representation, because of the one-to-one mapping from the transformed bin index to the original bin index of our representation, the decoding step is not necessary for all queries and also not needed for all variables. Based on the type of query, we can directly obtain the target histogram without decoding or only having to decode a subset of variables.

5.2 Histogram Query

In this section, we present several query operations using our histogram representation to derive novel histograms for different visualization applications. Here we assume the sweep space filling curve is used. One query example is histogram marginalization which is useful for users to investigate the relationships among a subset of variables. Multiple query operations can also be combined together, for example the user can first perform a histogram marginalization, followed by a histogram bin merge to obtain a lower resolution histogram among the selected variables. Below we describe the algorithms to perform such queries using our representation in detail.

5.2.1 Histogram Marginalization

The first operation that we present here is histogram marginalization. The purpose of this operation is to compute a multivariate histogram that involves an arbitrary subset of variables. We assume that an initial histogram computed from all variables, stored in the representation described above, is already available, but the user wants a multivariate histogram related to only a subset of variables. To compute this new histogram, we marginalize the pre-stored multivariate histogram over the variables that have not been selected by the user. Suppose the pre-computed histogram involves four variables A, B, C and D and the user is interested in the histogram of B and D. To compute the new histogram, we marginalize the multivariate histogram over variables A and C, that is: $P(B = b, D = d) = \sum_a \sum_c P(A = a, B = b, C = c, D = d)$.

As discussed in Section 5.1, the multivariate histogram is stored as dictionaries, along with index and frequency pairs where each index is represented as a bit string. Since in our representation, different bits in the index correspond to the bin indices of different variables, the histogram marginalization operation can be easily performed using a bitwise AND operation, followed by a frequency sum. For the histogram marginalization operation, we do not need to decode the index to map the transformed bin index to the original bin index. The marginalization operation can be directly applied on the transformed bin index. This is because that:

- For a histogram marginalization operation, whether two histogram entries need to be summed together or not only depends on whether these two entries have the same bin index for the marginal variables.

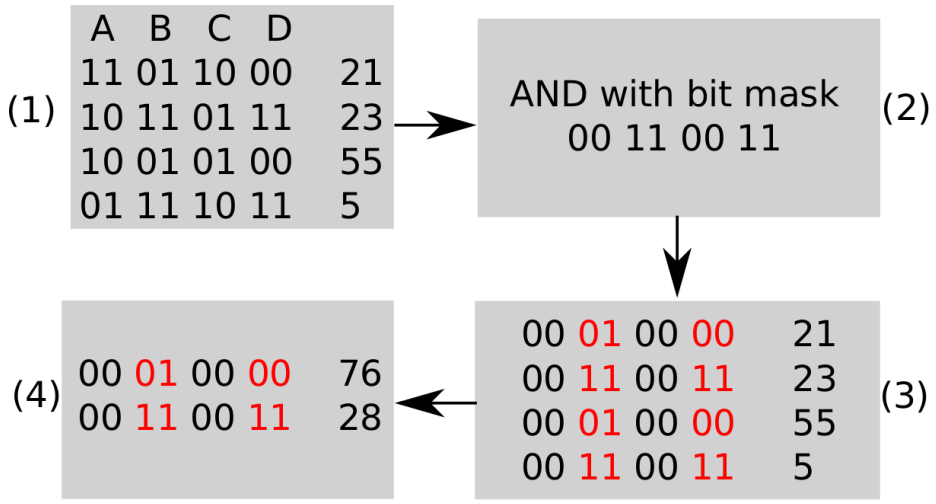


Figure 5.5: Query for histogram P(B,D). (1) A multivariate histogram which has four variables A, B, C, and D. (2) An AND operation is performed with a bit mask. (3) The result after the AND operation. (4) The entries with the same index are summed together to get the marginalization result.

- Given two histogram entries, if the transformed bin indices for a variable are the same, the original bin indices for this variable must be the same.

When using our representation, the marginalization can be done by setting the bits corresponding to the variables that are not selected by the user to 0s and then summing up the frequencies that have the same index. A simple AND operation with a bit mask is used to set the necessary bits to 0s. An example for this marginalization operation is shown in Figure 5.5. This multivariate histogram has 4 variables A, B, C and D. After data space transformation and padding, each variable has 4 bins, so the length of the index is 8 and every 2 bits represent the bin index of a variable. In the example, the user is interested in the multivariate histogram of $P(B,D)$. To support this query, the bit mask used is 00110011 with the bits corresponding to B and D set to 1s and all the other bits set to 0s. Then, for each index, we perform a logical bitwise AND operation with the bit mask to set the bits

corresponding to A and C to 0s as shown in the third step in Figure 5.5. In the last step, we add the frequencies with the same index together to get the marginalization results.

5.2.2 Histogram Bin Merge

Histogram bin merge is to combine the frequencies in adjacent bins along certain dimensions of the multivariate histogram to arrive at a new histogram with a different level of value discretization. This operation is used when certain analytical tasks require histograms of different resolutions. We assume that the precomputed multivariate histogram is defined at a higher resolution. The number of bins for each variable before data space transformation is chosen to be power of 2 for convenience reason. Suppose the number of bins before data space transformation for the high resolution multivariate histogram with 4 variables, for example, is $2^{K_1}, 2^{K_2}, 2^{K_3}, 2^{K_4}$, respectively for each variable, a lower resolution histogram with $2^{K_1-1}, 2^{K_2-2}, 2^{K_3-1}, 2^{K_N-0}$ bins for each variable can be accurately computed by summing the frequencies of the consecutive bins groups of size $2^1, 2^2, 2^1, 2^0$. For the histogram bin merge operation, we need to decode first and then perform the operation.

In term of the query procedure, the user specifies how many levels he wants to coarsen from the original multivariate histogram along each dimension by providing a N dimensional vector where N is the number of variables. Histogram bin merge operation is performed by using the same bit-wise AND operation as the histogram marginalization operation except that the bit mask in use is different. Given a high resolution multivariate histogram with N variables with $2^{K_1}, 2^{K_2}, ..., 2^{K_{N-1}}, 2^{K_N}$ bins respectively for each variable, the user specifies a N dimensional vector $L = \{l_1, l_2, ...l_{N-1}, l_N\}$ where $0 \leq l_i \leq K_i$ for $i = 1, ..., N$ to obtain the lower resolution histogram of $2^{K_1-l_1}, 2^{K_2-l_2}, ..., 2^{K_{N-1}-l_{N-1}}, 2^{K_N-l_N}$

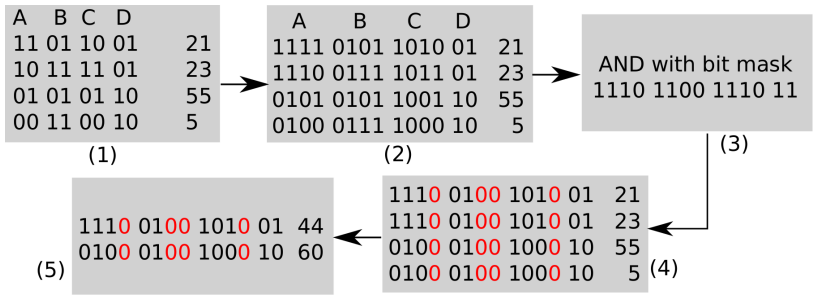


Figure 5.6: (1) A multivariate histogram which has 4 variables. (2) We query for a lower resolution multivariate histogram with $L = \{1,2,1,0\}$, and we decode the bin index for variable A,B and C to get the original bin index. (3) A logical AND operation is performed with a bit mask. (4) The result after the AND operation. (5) The entries with the same index are summed together to get the bin merge result.

bins. When $l_i = 0$, it means we do not need to perform bin merge operation along the i_{th} variable. In our implementation, we only decode the bin index for the variables whose level to coarsen is not zero. Given $L = \{l_1, l_2, ...l_{N-1}, l_N\}$ where $l_2 = 0$ and $l_N = 0$, suppose the number of bits used for variable 2 and N after the data space transformation is m_2 and m_N, the corresponding bit mask is:

$$\underbrace{1,...,1}_{K_1-l_1}\underbrace{0,...,0}_{l_1}\underbrace{1,...,1}_{m_2}\underbrace{1,...,1}_{K_3-l_3}\underbrace{0,...,0}_{l_3},...,\ \underbrace{1,...,1}_{K_{N-1}-l_{N-1}}\ \underbrace{0,...,0}_{l_{N-1}},\underbrace{1,...,1}_{m_N}$$

An histogram bin merge example is shown in Figure 5.6. The multivariate histogram has 4 variables. Before the data space transformation, each variable has 16 bins while after the data space transformation and padding each variable has 4 bins. There are 4 nonempty entries in this multivariate histogram. In this example, $L = \{1,2,1,0\}$ so the bit mask is 11101100111011. Similar to the histogram marginalization operation, a logical AND operation is performed over each index with the bit mask and then we sum together the frequencies with the same index to get the result.

5.2.3 Conditional Histogram

The next operation that we present is the conditional histogram query. Given N variables $V = v_1, v_2, ..., v_N$, a set of variables $U \subset V$ and another set of variables $W \in (V - U)$, this operation computes the distribution of U given the variables in W in particular bin ranges. The conditional histogram is useful when the user wants to investigate the relationship between a set of variables U and another set of variables W when the variables in W are in the specific value ranges. For example, we can compute the entropy of $P(U|W)$ which shows the uncertainty of U given the variables in W in the specific value ranges. Suppose U contains two variables B and C, W contains two variables A and D, and the specific value range for variable A is (min_A, max_A) and for variable D is (min_D, max_D), the conditional histogram query operation is defined as:

$$P(bin(B) = b, bin(C) = c | min_A \leq bin(A) \leq max_A, min_D \leq bin(D) \leq max_D) =$$
$$\sum_{a=min_A}^{max_A} \sum_{d=min_D}^{max_D} P(bin(A) = a, bin(B) = b, bin(C) = c, bin(D) = d)$$

The first step to compute the conditional histogram is to select all the entries that fall into the specific bin ranges for the variables in W. For this selection step, we need to decode for the variables in W to get their original bin index. Then, for each selected entries, a logical AND operation is performed with a bit mask to set all the bits not corresponding to the variables in U to 0s. The final results are computed by summing up the frequencies of the entries that have the same index.

An example is shown in Figure 5.7. This multivariate histogram has 4 variables A, B, C and D. Before the data space transformation, each variable has 16 bins while after the data space transformation and padding each variable has 4 bins. There are 6 nonempty entries in this multivariate histogram. In this example, we compute $P(A|11 \leq bin(C) \leq 15)$.

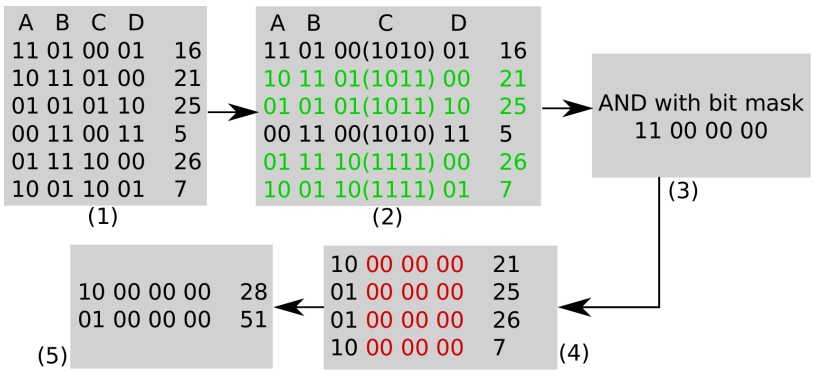


Figure 5.7: Query for $P(A|11 \leq \text{bin(C)} \leq 15)$. (1) A multivariate histogram which has 4 variables A, B, C and D. (2) We decode the bin index for variable C to get the original bin index and then the corresponding entries with $11 \leq \text{bin(C)} \leq 15$ are selected(green color). (3) A logical AND operation is performed with a bit mask for those selected entries. (4) The result after the AND operation. (5) The entries with the same index are summed together

5.2.4 Time Histogram

The last operation that we present is the time histogram query. Our method can be easily extended to time varying multivariate data. For time varying multivariate data, we treat the time step as another variable and set the number of bins to be the number of time steps. When a time varying data is used, we can query the time histograms [3] for an user selected data block. The queried time histogram is useful to show the features presented in this selected data block over time. Time histogram query is similar as histogram marginalization query except that the variable time step must be included in the marginal variables. In term of the query procedure, the user specifies the variables and the data block from which the time histogram is computed. The variable time step must be included in those selected variables, and then the rest of the query is the same as histogram marginalization query.

5.3 Visualization Applications and Analysis

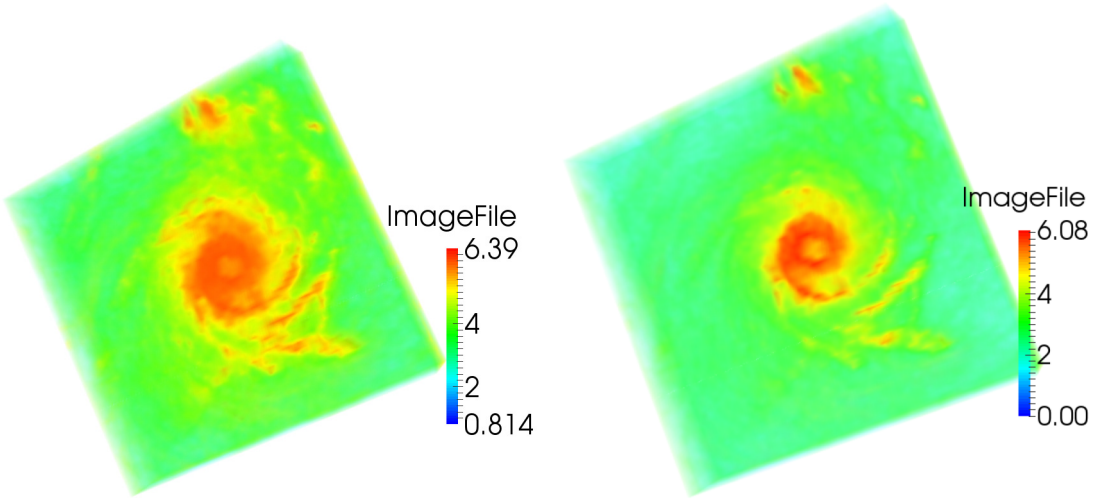
5.3.1 Local Statistical Analysis

Several statistical metrics such as mean, variance and information entropy [110] can be computed from block-wise multivariate distributions to determine whether certain features exist in the blocks. Computing such block-wise distributions from large scale data, however, is expensive. Also, when the user changes the combination of variables or the resolution of the desired histograms, we need to repeat the computation and access the raw data, which can be very costly. With our technique, we do not need to access the raw data, instead, we can efficiently derive histograms from different combinations of variables at different resolutions. Figure 5.8 shows an example of local statistical analysis using our technique. The dataset used is Isabel, which contains a total of 13 variables. Initially, we compute and store the block-wise multivariate histograms from all the 13 variables. At analysis time, we read back these pre-stored multivariate histograms and then compute the required histograms for analysis. First, the user queries the multivariate histograms of selected combination of variables for each block by marginalization, and then the multivariate histograms are coarsened to a desired bin resolution by histogram bin merge.

We show a block-wise entropy field computed from the multivariate histograms of U, V and W velocity fields in Figure 5.8. The left image uses the multivariate histograms at the finest level of detail (256 bins for each variable). Lower bin resolution histograms (128 bins for each variable) are used to generate the right image.

5.3.2 Query Driven Visualization

Given the multivariate histograms computed from each block, query-driven techniques can be used to visualize and analyze the multivariate dataset. During analysis, the user



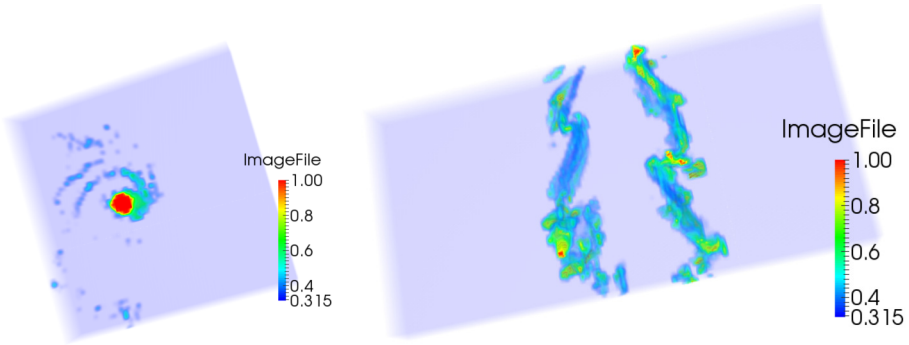
(a) (b)

Figure 5.8: (a). Block-wise entropy field computed from the multivariate histograms of U, V and W velocity at the finest level of detail; (b). Block-wise entropy field computed from the lower resolution multivariate histograms by merging consecutive bins groups of size 2.

can easily retrieve the required histograms to perform query-driven visualization with our techniques. In this section, we present four different query-driven visualization applications based on our histogram representation.

Joint Probability Query

Since the data is decomposed and represented as multiple block-wise multivariate histograms, an useful query would be to query for blocks where the co-occurrence of some variables satisfies a user defined condition. This kind of query requires us to efficiently derive the multivariate histogram for the selected variables. An example of such a query is: $P(200 \leq bin(A) \leq 255, 0 \leq bin(B) \leq 50) > 70\%$. In this query, we query for the blocks where the probability for variable A belonging to bin 200 to 255 and variable B belonging to bin 0 to 50 is bigger than 70%. This query gives us the blocks where the majority of

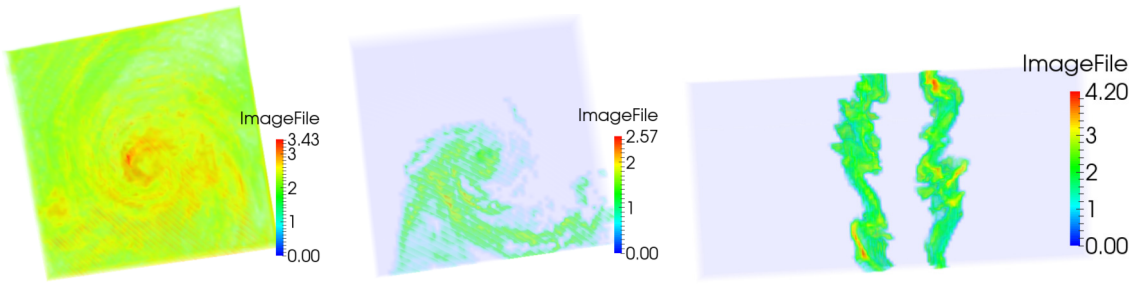


(a) (b)

Figure 5.9: (a). $P(30 \leq \text{bin(Cloud)} \leq 255, 30 \leq \text{bin(QRain)} \leq 255) > 50\%$ OR $P(0 \leq \text{bin(Pressure)} \leq 50) > 50\%$; (b). $P(0 \leq \text{bin(OH)} \leq 96, 105 \leq \text{bin(Mixture Fraction)} \leq 120) > 50\%$.

points have high value for variable A and low value for variable B. We can also use OR to combine multiple queries. For example, $P(200 \leq bin(A) \leq 255, 0 \leq bin(B) \leq 50) > 70\%$ OR $P(100 \leq bin(C) \leq 150) > 50\%$. With our technique, we can quickly answer the query by computing the desired multivariate histogram of the selected variables to get the probability. A fuzzy similarity score proposed by Johnson and Huang [52] can be used to support fuzzy matching. The similarity score has a value range of 0 to 1 while larger value means they are more similar.

Figure 5.9 shows two query results using Isabel and Combustion. In the left image, we query the blocks that the probability of Pressure belongs to bin 0 to 50 or the accumulate probability of Cloud belongs to bin 30 to 255 and QRain belongs to bin 30 to 255 is bigger than 50%. The low Pressure query gives us the hurricane eye while the region corresponding to hurricane wall contained in the higher Cloud and QRain query. In the right image, we query the blocks that the probability of OH belongs to bin 0 to 96 and Mixture fraction belongs to bin 105 to 120, which correspond to where the flame dissipates.



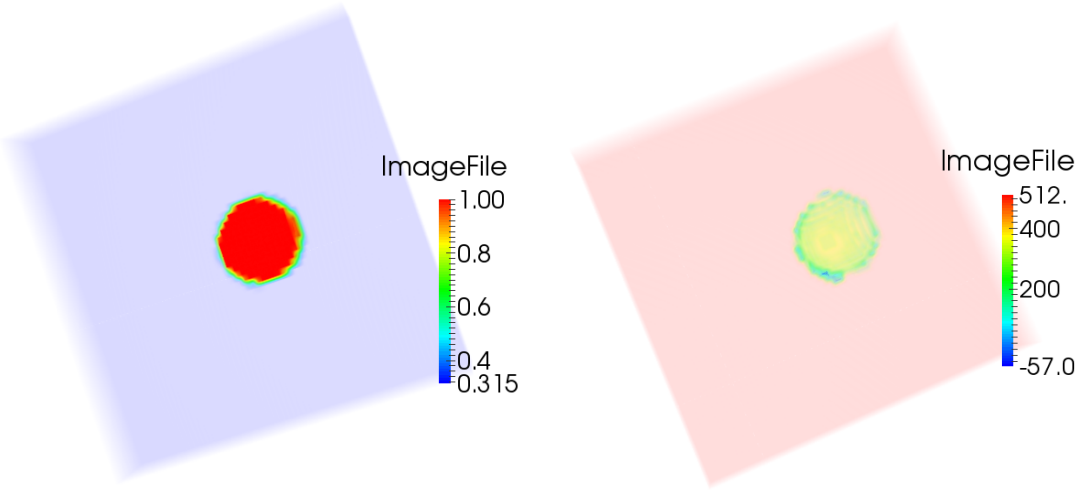
(a) (b) (c)

Figure 5.10: (a). The entropy field computed from P(Temperature$|0 \leq$ bin(QVapor)≤ 10) for Isabel; (b). The entropy field computed from P(Temperature$|200 \leq$ bin(QVapor)≤ 210) for Isabel; (c). The entropy field computed from P(OH$|120 \leq$ bin(Mixture Fraction)≤ 130) for Combustion.

Conditional Probability Query

As described in Section 5.2, we can compute conditional histograms from those block-wise multivariate histograms. Based on the conditional histograms, further statistical information such as information entropy and variance could be derived. For example, if Y is a set of variables and X is another variable, by computing some statistical metrics of the distribution $P(Y|min_x < X < max_x)$, we can infer the relationship between Y and X when X is in a value range $[min_x, max_x]$. For example, the entropy of $P(Y|min_x < X < max_x)$ tells us how uncertain we know about Y when X belongs to value range $[min_x, max_x]$.

Figure 5.10 shows the query results using Isabel and Combustion. The left image shows the conditional entropy of temperature given QVapor belongs to bin 0 to 10. Higher entropy means when QVapor belongs to bin 0 to 10, we are more uncertain about the value of temperature. Lower entropy means we are more certain about the value of temperature. Zero entropy has two meaning, one is that we are very certain about the temperature value and the other is that QVapoer does not have values within bin 0 to 10 in this region. In the



(a) (b)

Figure 5.11: (a). The query result of $P(0 <\text{bin(pressure)}< 150) > 50\%$; (b). The conditional fuzzy isosurfaces computed from P(QRain$= 0.006|0 <$bin(pressure)< 150)

middle image, we show the conditional entropy of temperature given QVapor belonging to bin 200 to 210. The combustion dataset was used in the right image. It shows the conditional entropy of OH given Mixture Fraction belonging to bin 120 to 130.

Conditional Fuzzy Isosurfacing

Thompson et. al. [98] proposed a technique to compute Fuzzy Isosurfaces from block-wise distributions. A likelihood value g is computed given a particular isovalue to indicate the likelihood of the existence of an isosurface for each block. Here, we extend it to multivariate data and compute the conditional fuzzy isosurfaces. Given a variable Y and another variable X, the conditional fuzzy isosurface is the fuzzy isosurface computed from distribution $P(Y|min_x < X < max_x)$. For blocks that do not have the conditional distribution $P(Y|min_x < X < max_x)$ because the variable X does not have values within $[min_x, max_x]$ in

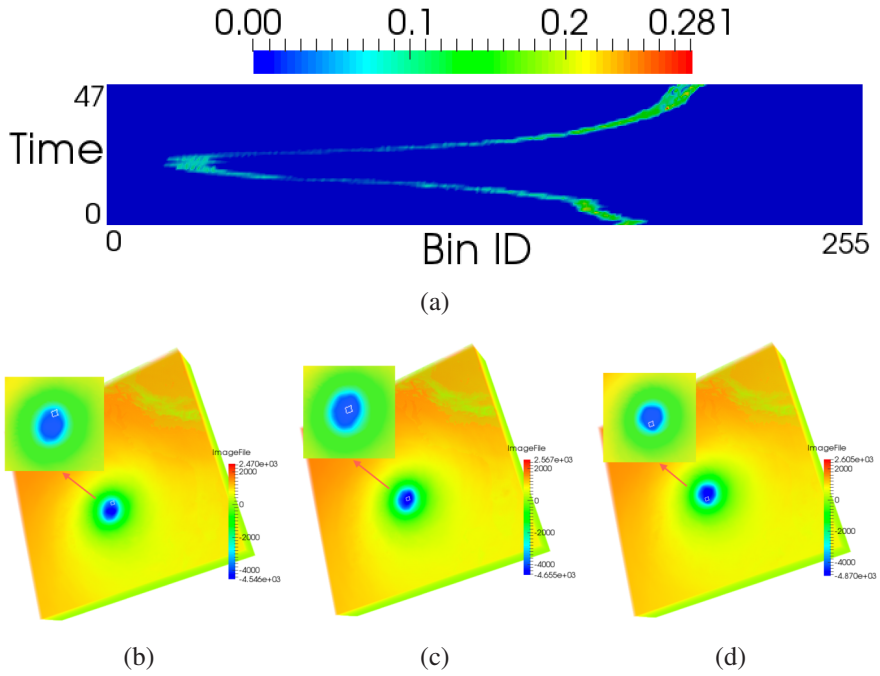
that block, we set the likelihood value g to be the total number of data points in the block which means the block is strongly above the isovalue.

The right image in Figure 5.11 shows the conditional fuzzy isosurface computed from Isabel. We computed the fuzzy isosurface of QRain with an isovalue 0.006 when the pressure belongs to bin 0 to 150. The left image in Figure 5.11 shows the query results of $P(0 < bin(pressure) < 150) > 50\%$ using the method presented in Section 5.3.2. When only the pressure is used, we get the hurricane center in the result. By using the conditional fuzzy isosurfaces to show the regions with high QRain value such as 0.006 given pressure belonging to bin 0 to 150, we get the outside of the hurricane.

Time Varying Data Analysis

If a time varying dataset was used, with our approach, we could easily query the time histogram [3] of different variables from arbitrary data blocks. A marginalization operation over the user selected variable V and the variable time step for the user selected block gives us the time histogram of the variable V for the selected block. The time histogram is concatenated by the 1D histograms for the variable V computed from different time steps. The time histogram computed from the block can be used to indicate the feature behavior presented in this block over time.

The dataset *Isabel* models a strong hurricane in the West Atlantic region. It has 48 times steps and captures the movement of the hurricane. In general, the low pressure area indicates the presence of hurricane eye, so the time histogram computed from pressure can be used to identify when the hurricane passes a local block. An example is shown in Figure 5.12. The top figure in Figure 5.12 shows a time histogram computed from the variable pressure for a data block from Isabel. Low pressure indicates the presence of hurricane eye. From the time histogram, we can see the hurricane passes through the data



(a)

(b) (c) (d)

Figure 5.12: (a). Time histogram; (b)-(d). Volume rendering of the pressure field for time step 17, 20 and 23. The data block from which the time histogram is computed is shown in white.

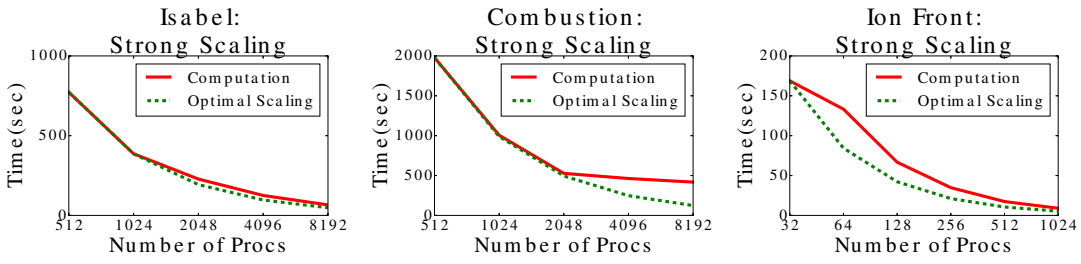
block from time step 17 to 23. The bottom three figures in Figure 5.12 show the volume rendering of pressure variable for time step 17, 20 and 23. The data block is shown in white color, and the hurricane passes through the data block gradually.

5.4 Performance

In our implementation, we divide the entire dataset into axis-aligned blocks of equal size at the finest level of detail specified by the user. Then these data blocks are assigned to parallel processes in a round-robin order and each process computes a multivariate histogram for each assigned data block. We use **boost::dynamic_bitset** to represent the bit indices. The index and frequency pairs are stored using **boost::unordered_map**.

This section presents the performance results of our technique, including the scalability of the distributed multivariate histogram computation, histogram size, and the scalability of different query operations. Three datasets were used. The dataset *Isabel* with a resolution of $500 \times 500 \times 100$ models a strong hurricane in the West Atlantic region in September 2003. This dataset contains thirteen variables and 32 time steps were used. The total size is about 38.74GB. *Combustion* is a dataset which simulates the turbulent combustion. This dataset contains five variables. The resolution of this dataset is $480 \times 720 \times 120$ and 64 time steps were used. The total size is about 49.44GB. The dataset *Ion Front* from the IEEE 2008 Visualization Design Contest [105] is a simulation of an ionization front instability. This dataset has a resolution of $600 \times 248 \times 248$. There are ten variables in this dataset and we use 1 time step for our experiment. The total size is 1407.71 MB.

We compared the sizes of multivariate histograms using our representation with two other representations. The first representation is denoted as vector representation, which stores the non-empty entries as index and frequency pairs while the index is represented as a K-tuple if there are K variables. The i_{th} element in the K-tuple represents the bin id of the i_{th} variable. The elements in the tuple are stored as unsigned short integers. The second representation denoted as *Curv.absci*. also represents the non-empty entries as index and frequency pairs but the index is represented as a single scalar value [18]. We store the scalar values as bit strings. The number of bits used depends on the maximum possible value of the index and each index has the same number of bits. For those two representation, we store the indices and frequencies separately in two files. For our representation, we stored three files, and they store the indices, dictionaries and frequencies respectively. Since the size of the frequency file is the same for all these three representations, we compared the



(a) (b) (c)

Figure 5.13: Strong scaling results of histogram construction.

sizes of the index files generated by the other two representations with the size of the index file plus the size of the dictionary file generated by our representation.

5.4.1 Construction Scalability

The multivariate histogram computation is done in parallel. We tested the scalability of this stage on the Blue Gene/Q *Vesta* system at the Argonne Leadership Computing Facility. *Vesta* contains 2,048 nodes each holding 16 PowerPc A2 1600MHz cores sharing 16GB of RAM and utilizes the General Parallel File System. The total memory is 32 TB.

We used all the three datasets in our test. The dimensions of the multivariate histogram generated from Isabel, Combustion and Ion Front are $256^{13} \times 32, 256^5 \times 64$ and 256^{10} respectively. We set the number of partitions along x, y, and z dimensions to $64 \times 64 \times 16$ for Isabel, $64 \times 128 \times 16$ for Combustion and $64 \times 32 \times 32$ for Ion Front. The strong scaling results of the computation time including data space transformation and multivariate histogram representation computation are shown in Figure 5.13. As shown in the Figure, our multivariate histogram computation shows good scalability up to thousands of processes for all the three datasets.

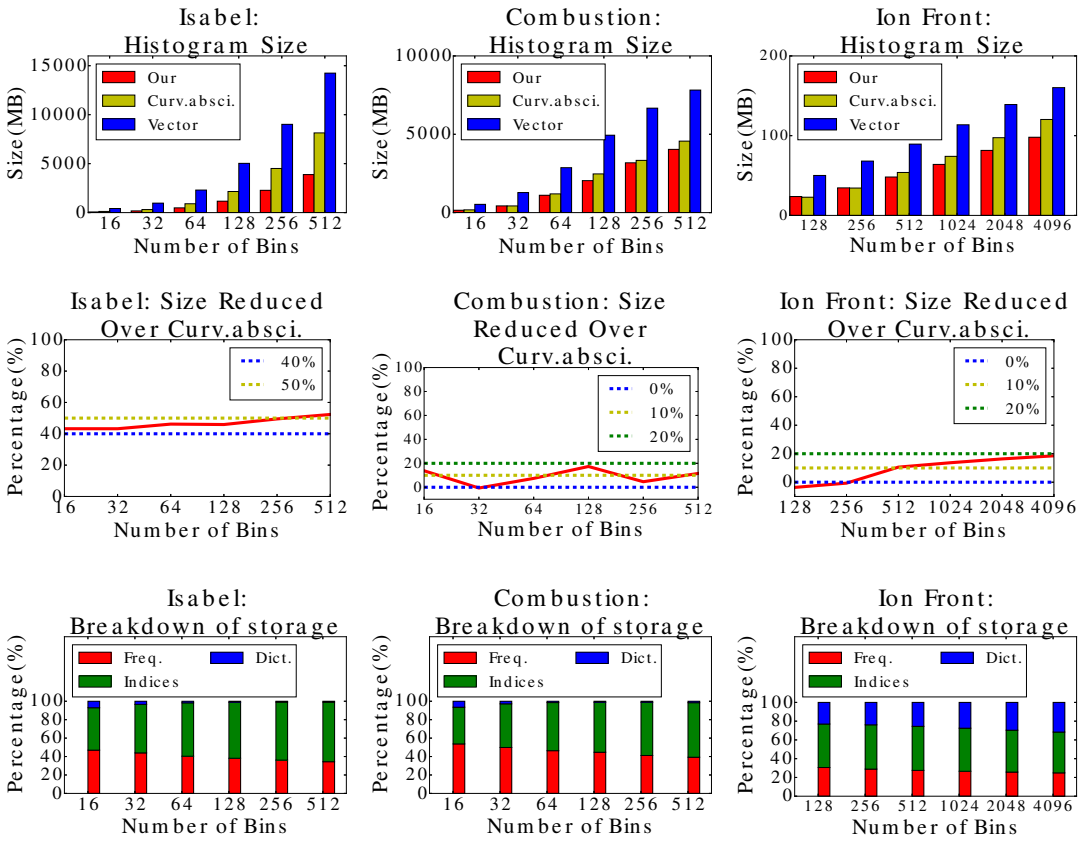
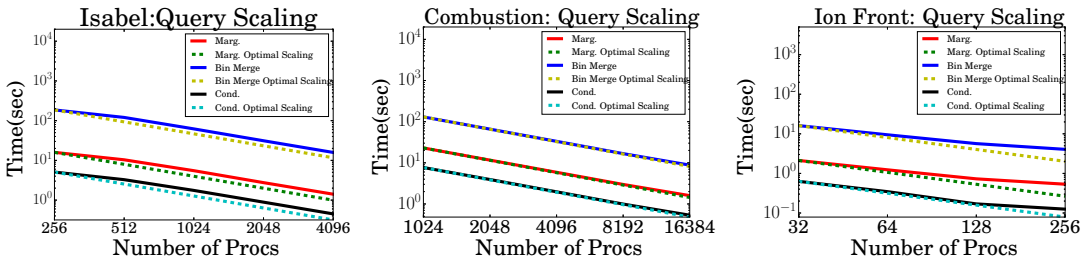


Figure 5.14: **Top**: Storage cost versus number of bins under different representations. **Middle**: Percentage of storage reduced with our representation over *Curv.absci*. representation. **Bottom**: The breakdown of the storage among frequencies, indices, and dictionaries.

5.4.2 Multivariate Histogram Size

In this section, we compared the storage cost by using our representation with the vector and *Curv.absci*. representations. The number of bins along each dimension were the same and in our experiments we tested for 16, 32, 64, 128, 256 and 512 bins for Isabel and Combustion, and 128, 256, 512, 1024, 2048 and 4096 bins for Ion Front.

The comparison of the storage cost for different representations is shown in the top three figures in Figure 5.14. The storage cost by using our representation is consistently smaller



(a) (b) (c)

Figure 5.15: Strong scaling results of query.

than the other two representations. This is reasonable because our representation uses fewer bits than the other two representations to represent the index. The middle three figures in Figure 5.14 show the percentage of storage reduced compared with the *Curv.absci.* representation. These figures indicate generally the percentage of storage cost reduced compared with the *Curv.absci.* representation increases as the number of bins increases. This is because when larger number of bins is used, the size of multivariate histogram is considerably large which make our data space transformation useful. The bottom three figures in Figure 5.14 show the breakdown of the storage costs among frequencies, indices, and dictionaries of our representation.

5.4.3 Query Performance

We implemented all the query operations presented in Section 5.2 in parallel. In our implementation, those histograms are distributed to the compute processes, and then each process performs the query operations on those assigned histogram in parallel. In this section, we study the scalability of the marginalization, bin merge and conditional histogram query operations. We tested the performance on *Vesta*. For the time histogram query, since it is performed only on the user specified data block, no matter how many processes are

used, only the process which has the user specified data block performs this query operation. Instead of studying the scalability, we show the performance of time histogram query given a smaller number of processes. For all tests, the dimensions of the multivariate histogram used for Isabel, Combustion and Ion Front are $256^{13} \times 32, 256^5 \times 64$ and 256^{10} respectively. We measured the response time for all the four query operations described in Section 5.2. For each query operation, we randomly generated 10 test cases, and the reported timings are the average response time.

The scalability results of the marginalization, bin merge and conditional histogram query operations are shown in Figure 5.15. As the figures demonstrated, strong scalability was observed up to a large number of processes for those three query operations and different datasets. The bin merge operation requires more time than the other operations because we need decoding for the bin merge operation while the marginalization operation does not require decoding. Although the conditional histogram operation requires to decode a subset of variables, since it only performs bit-wise operations on the entries which fall into the query bin index ranges, it generally performs better than the other query operations.

For time histogram query, we use 256, 1024 and 32 processes for Isabel, Combustion and Ion Front respectively, and the time required for each dataset is 0.0046 sec, 0.0028 sec and 0.00049 sec.

5.5 Conclusion and Future Work

In this chapter we present a novel compact structure to represent a multivariate histogram. We first use a data space transformation to transform the original large data space to a much smaller data space, and then we index the small data space to compute the representation of the multivariate histogram. By using this representation, the space cost of

storing a multivariate histogram is reduced significantly. Given this new representation, we present several query operations to compute different types of histograms efficiently. We also present several visualization applications using our technique.

In the future, we plan to extend our technique along the following directions. First, we will extend our technique to handle curvilinear and unstructured grid data which are generated from many simulations. Domain decomposition and local histogram computation are more challenging due to their irregular geometric and topological structures. Second, we plan to explore other query operations.

Chapter 6: Multivariate Volumetric Data Analysis and Visualization through Bottom-up Subspace Exploration

Nowadays, computational fluid dynamics(CFD) and weather model simulations commonly produce multiple attributes/variables associated with each grid point in the data domain. Usually, features in multivariate datasets can be better classified with more than one variable. To display features in a multivariate volumetric data, direct volume rendering has been used to visualize and analyze multivariate data. In direct volume rendering, data values are mapped to optical properties such as color and opacity through transfer functions. A good transfer function can reveal important features in the data more effectively. However, identifying good transfer functions is nontrivial and there have been many previous works on this topic [34, 42, 55, 56, 60]. Designing good multivariate transfer functions is much harder than 1D transfer functions because of the increasing number of variables and the complex relationships among the variables in multivariate datasets. In general, a multivariate transfer function can be represented as a multidimensional lookup table which maps different combinations of data values to different optical properties. However, as pointed out by Kniss et al. [57], a major limitation of the multidimensional lookup table is the increasing memory cost as the number of variables increases. In order to resolve the issues of using a multidimensional lookup table, Kniss et al. [57] proposed to represent a multivariate transfer function as several Gaussian components. In the Gaussian transfer

function representation, different features in the dataset are represented as different Gaussian components and each Gaussian component is assigned a unique color. The data point with values that are closer to the center of a Gaussian component will have high opacity, and thus will be shown in the final volume rendering view. The final volume rendering result is visualized by combining the colors and opacities from all Gaussian components. However, it is a nontrivial task to specify those Gaussian components that are potentially corresponding to interesting regions in the data. When users are familiar with the data and know how features are defined based on the data values, users can set those Gaussian components centered around those data values to highlight the corresponding features. However, when users are not familiar with the data, it is difficult for users to specify those Gaussian components to extract salient features.

One way to design a multivariate transfer function is through dimensionality reduction such as Multidimensional Scaling(MDS) [40]. Based on those dimensionality reduction techniques, the high dimensional points are projected to lower dimensions while preserving the distances between each other. Clusters can be identified and selected from the lower dimensions. Each cluster can then be mapped to a Gaussian component by setting the Gaussian mean around the cluster center to support the design of multivariate Gaussian transfer functions. As identified by Parsons et al. [78], for a high dimensional data, clustering while considering all dimensions is ineffective due to the following reasons:

- Not all the dimensions are relevant. When lots of irrelevant dimensions are considered, meaningful clusters in subspaces can be destroyed by those irrelevant dimensions.

- Because of the curse of dimensionality problem, distance measurements in high dimensional space are difficult to interpret [78].

For multivariate scientific data, a feature usually could be well defined with a couple of variables and not necessary all the variables are needed. For example, the Hurricane Isabel dataset which models a strong hurricane in the West Atlantic region contains 13 variables. The feature Hurricane eye in the dataset could be well defined as the low velocity and low pressure region which only involves two variables. Thus, the feature Hurricane eye can be more efficiently identified by exploring the subspace of velocity and pressure. In order to address the above issues, and to efficiently discover features in high dimensional data, our system employs a bottom-up approach to incrementally construct those Gaussian components during exploration. In the bottom-up approach, users start from lower dimension and make selection in lower dimension. Then, the selection can be further refined by extending to a higher dimension. This kind of bottom-up exploration pipeline has been used in several techniques to explore multivariate volumetric datasets [8, 31, 82, 112, 113]. These previous bottom-up approaches mainly rely on data point brushing on parallel coordinate or scatter plot and usually requires users' prior knowledge about the data, so the users know which data points to brush. In this chapter, based on information derived from the data, we present a systemic approach to guide user exploration, so the users do not need to blindly brush on parallel coordinate or scatter plot without any guidance.

In our system a scatter plot matrix is used. Compared with Parallel Coordinates, all pairwise variable relationships can be shown in scatter plot matrices while only the relationship between adjacent coordinates are able to be shown in parallel coordinate. In our system, we show all single variable and two variables subspaces in a matrix juxtaposition view. Users can navigate through the different subspaces using the matrix widget. Since scatter plots are resulted from projecting high dimensional data points to lower dimensions, the high dimensional information can be lost [82]. In [82], the authors proposed a technique

to use a combination of 2D and 3D scatter plots to solve this problem. However, when we face datasets with the number of dimensions higher than three, it is nontrivial for us to visualize those high dimensional data points. In this chapter, we proposed two weights: size weight and scatter weight associated with each projected data point as a complementary information for the loss of high dimensional information due to projection. Based on those two weights, data point filter and weighted kernel density estimation operations are used to help users to explore multivariate datasets.

Since a bottom-up exploration strategy is used in our system, when users do not have much priori knowledge about the data, several questions need to be addressed to achieve effective exploration:

- **Which subspace to explore?** The first question need to be addressed is which subspace to explore. Given N variables, there are $N(N-1)/2$ two dimensional subspaces and N one dimensional subspaces. Given such a large number of subspaces, it is difficult for users to choose which subspace to explore next.

- **What data points to be brushed in the selected subspace?** Once a subspace (X,Y) is selected for exploration, users need to decide what data points to be brushed, i.e. what values should be the mean and variance of variable X and Y in the Gaussian function. A good selection should lead users to discover interesting features hidden in the data.

In our system, we derived information from the data sets to assist user exploration. In order to guide users to choose the next subspace for exploration, an entropy based approach is used. Data points filter and weighted kernel density estimation based on size and scatter weights are used to help users to select data points in the current exploring subspace.

Our system contains several interactive widgets that are linked together to assist user exploration. The layout of our system is shown in Figure 6.1. The upper triangle of the matrix widget (Figure 6.1.B) is used to show all single and two variables subspaces in a juxtaposition view. The lower triangle of the matrix widget shows the entropy value computed from the joint histogram defined in the corresponding subspace. Users can navigate through different subspaces by double clicking the corresponding subspace. All the subspaces are linked together and dynamically updated based on users' selection in one subspace. Parallel coordinate plots (PCP) (Figure 6.1.A) are used to show the Gaussian functions corresponding to the user-identified features. Users can use the PCP to adjust the variance of the currently specifying Gaussian component. A multidimensional scaling plot (Figure 6.1.F) shows the two dimensional embedding of all data points by considering all the dimensions. The high dimensional project view gives users a global view of all the data points and which regions have been covered by the specified Gaussian components so far. The data points that are already covered by a specified Gaussian component are colored by the color assigned to that Gaussian component. A zoom-in view of the user-selected subspace is shown in a window widget (Figure 6.1.D). Density estimation result is shown in this zoom-in window widget. A 2D color map widget (Figure 6.1.E) is used to show the color map for volume rendering which allows users to visually link the volume space with the transfer function space. All the 2D images such as density estimation results, subspace plots and entropy submatrix use the same color map (Figure 6.1.G) where white indicates low and red indicates high values. The main contributions are:

- We presented an effective and intuitive system for users to interactively explore multivariate volumetric data and identify interesting features existing in different subspaces.

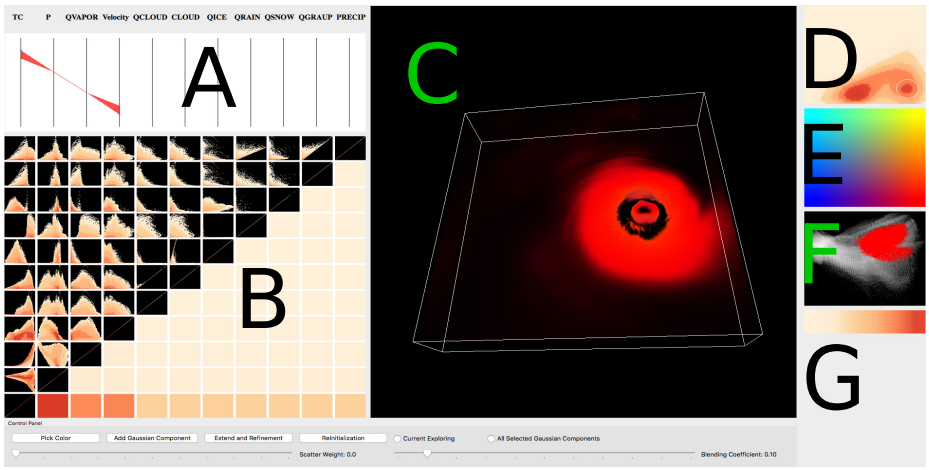


Figure 6.1: The layout of our system interface.

- In order to provide users with the high dimensional information which is lost due to projection, we proposed two weights: scatter weight and size weight associated with each data point. Then, based on those two weights, users can use data point filter and weighted kernel density estimation to identify features and gradually construct Gaussian Transfer Function.

- Besides the scatter and size weight, we derived additional information such as information entropy from the data to assistant user exploration.

6.1 System Overview

As described previously, features represented as clusters in subspaces may not be effectively detected if all variables are considered. In this chapter, we propose a system to address the aforementioned problem. Information such as size and scatter weight are derived from the data set to guide user exploration, and multivariate transfer functions are

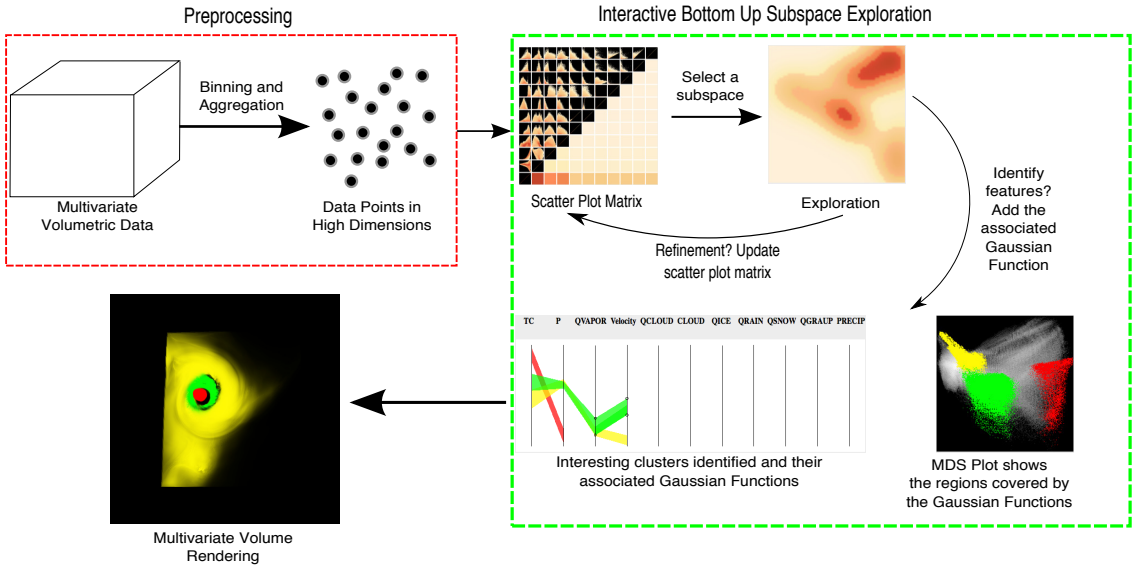


Figure 6.2: The pipeline of our system. In the preprocessing stage, given a multivariate volumetric dataset, a reduced set of data points are generated through data binning and aggregation. In the interactive exploration stage, users can perform bottom-up subspace exploration and identify interesting clusters with our system. The end product of the interactive exploration is multiple Gaussian components where each one is corresponding to a user-identified cluster. Finally, all the selected clusters can be visualized using multivariate volume rendering.

constructed accordingly during exploration. All the user identified clusters can be rendered through multivariate volume rendering to reveal interesting features in the data.

Figure 6.2 shows an overview of our system pipeline. Given a multivariate volumetric data set, each grid point corresponds to a multivariate data point. A multivariate volumetric data set with N grid points amounts to N multivariate data points in total. We apply binning and aggregation in value space to quantize and reduce the number of data points, which makes our subsequent interactive exploration more efficient. Given those data points after quantization and aggregation, all single and two variables subspaces are inspected in a juxtaposition view. As a data point in 2-dimensional or 1-dimensional subspace can map

to many data points in the original high dimensions due to projection, two weights are assigned to each data point. These two weights severe as the complementary information for the loss of high dimensional information due to projection, and are used to guide user exploration. The color channel is used to encode the weight information in the scatter plot matrix visualization. The end product of this bottom-up subspace exploration is multiple Gaussian components where each one is corresponding to a user-identified cluster in a subspace. All the selected clusters can be visualized using multivariate volume rendering to depict the features of the data.

6.2 Preprocessing

6.2.1 Data Reduction

With the increased computing power, the resolution of data generated from simulation also continues to grow. Taken a moderate size scientific dataset as an example. Hurricane Isabel which models a strong hurricane in the West Atlantic region in September 2003 has a resolution of $500 \times 500 \times 100$. This amounts to $25,000,000$ multivariate data points which poses a challenge for interactive subspace exploration. Given this large number of data points, directly performing our bottom-up subspace exploration can be difficult. In our subspace exploration process, kernel density estimation(KDE) is used to estimate the density to discover potential clusters existing in different subspaces. Given n data points, the time complexity of calculating k evaluation points using KDE is $O(kn)$. Given a constant number of evaluation points, the time complexity is linearly proportional to the number of data points. When the number of multivariate data points is large, it prohibits us to achieve interactive subspace exploration and cluster identification. To resolve this issue, in the preprocessing stage, we use data binning and aggregation to quantize and reduce

the number of multivariate data points. In the following section, we discuss in detail the data binning and aggregation, and the reduced set of data points generation in both full dimensions and low subspaces.

Data Binning and Aggregation

Data binning and aggregation are widely used techniques for data preprocessing and reduction. Data binning divides a continuous domain into intervals, and then the original value is mapped to the interval that contains the data value. It is a form of quantization. In our system, we bin the data in the value space. Suppose the multivariate data set contains n grid points and m variables, then the original multivariate data points denoted as P contains n data points in m dimensional value space. Given the range of the i_{th} variable, denoted as $[a_i, b_i]$, and the width of the bins for the i_{th} variable is h_i, a data point $p_j \in P$ that has values $(p_{j,1}, p_{j,2}, ..., p_{j,m})$ is quantized by using the following equation:

$$Binning(p_j) = (\left\lfloor \frac{p_{j,1} - a_1}{h_1} \right\rfloor, \left\lfloor \frac{p_{j,2} - a_2}{h_2} \right\rfloor, ..., \left\lfloor \frac{p_{j,m} - a_m}{h_m} \right\rfloor) \tag{6.1}$$

After quantization, the data points in P with their values fell into the same bin for each dimension have exactly the same value. Then, we perform data aggregation so that those data points in P which have the same value after quantization are represented as a single point q in the multidimensional value space. The new data point q is defined as:

$$q = Binning(p_i), i \in U \tag{6.2}$$

where U denotes a set of data points in P that have the same value as q after quantization. We assign a weight w to q:

$$w = |U| \tag{6.3}$$

the weight represents the number of original multivariate data points in P that have the same value as q after quantization, and we use Q to denote the set of data points after binning and aggregation.

6.2.2 Subspace Data Points Generation

Given the set of data points Q, a scatter plot matrix that contains all single and two variables subspaces is constructed. Each subspace plot in the scatter plot matrix shows the scatter plot of the data points generated for this subspace. In this section, we discuss subspace data points generation and the two weights: size weight and scatter weight associated with each data point in a subspace. We use 2-variable subspace as the example for illustration, data points in single variable subspaces can be generated accordingly.

Let $S^{x,y}$ denote the projected data points generated for the subspace defined by variable x and y. $S^{x,y}$ can be generated by removing the dimensions other than x and y for all data points in Q, which can be seen as an orthogonal projection in value space. We define two weights associated with each projected data point in $S^{x,y}$. Let $s_k^{x,y} \in S^{x,y}$ be the k_{th} data point. Many data points in Q could be mapped to a single data point in $S^{x,y}$ due to the orthogonal projection. Let U_k denote the set of data points in Q that are mapped to the data point $s_k^{x,y}$ in the x, y subspace, the size weight w_{size} and scatter weight $w_{scatter}$ for this projected data point are defined as:

$$w_{size} = \sum_{i \in U_k} w_i \tag{6.4}$$

$$w_{scatter} = |U_k| \tag{6.5}$$

where w_i is the weight associated with the data point $q_i \in U_k$ computed in equation 6.3.

The size weight w_{size} for a particular data point $s_k^{x,y}$ represents how many data points in the original set of data points P(before binning and aggregation) are mapped to the point

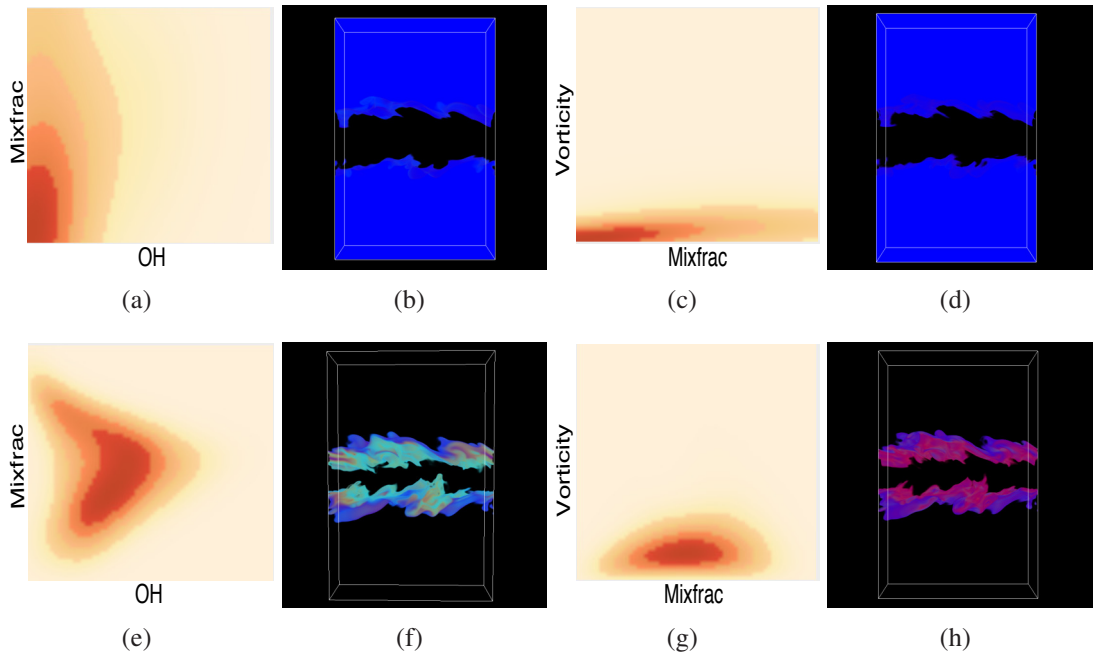


Figure 6.3: (a) and (c) shows the density estimation results when only size weight is considered and the volume regions corresponding to the dense regions are shown in (b) and (d) respectively. The dense region corresponds to the unimportant background region in this dataset. (e) and (g) shows the density estimation results when both size weight and scatter weight are considered and the volume regions corresponding to the dense regions are shown in (f) and (h). More interesting regions are identified.

$s_k^{x,y}$. A large weight means the data point $s_k^{x,y}$ corresponds to a cluster of a large size. The scatter weight $w_{scatter}$ for a particular data point $s_k^{x,y}$ represents how many data points in Q(after binning and aggregation) are mapped to it. A large weight generally means the cluster defined around the data point $s_k^{x,y}$ is more likely to be further refined in the extended subspace, since its high-dimensional mapping is more scattered.

In our subspace exploration pipeline, weighted kernel density estimation is used to identify possible clusters existing in a subspace. If only the size weight w_{size} is considered in the density estimation, the following problems will arise:

- The density estimation result will bias to the size of cluster, so clusters with relatively smaller size may not be easily discovered.

- When the data contains many unimportant data points such as background that have exactly the same value, the density estimation can not efficiently identify meaningful features that exist in the data. This problem is illustrated in Figure 6.3. Figure 6.3 (a) and (c) show the density estimation results when only size weight is considered and the dense region corresponds to background region in the data set as shown in (b) and (d). Figure 6.3 (e) and (g) show the density estimation results when both size and scatter weight are considered and the dense region corresponds to a more interesting region in the data set as shown in (f) and (h).

If only the scatter weight $w_{scatter}$ is considered during density estimation, some clusters that are corresponding to a very few number of data points in Q might be missed during exploration. In our system, we combine the size weight and scatter weight:

$$w = \alpha w_{size} + (1 - \alpha) w_{scatter} \tag{6.6}$$

w is a new weight for each projected data point in the subspaces and is used in density estimation. α is the blending coefficient between 0 and 1 which determines the influence of size weight. Figure 6.4 shows the effect of using different blending coefficients. As we can see, when α is small such as below 0.2, a small cluster can be easily identified as shown in the leftmost figure. When the value of α increases, a large cluster(usually corresponding to background) is then detected as shown in the rightmost figure.

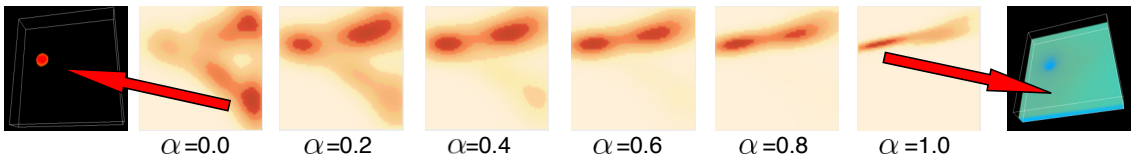


Figure 6.4: The middle six images show the density estimation results of the Temperature and Pressure subspace with different blending coefficients α after filtering out data points with scatter weights larger than a user-defined threshold. When the value of α is small such as 0.0 and 0.2, the influence of size weight on the density estimation result is small, so some small size clusters can be identified. In this example, a small size cluster corresponding to the Hurricane center can be easily identified with a small value of α. When the value of α is large, the influence of size weight on the density estimation result is large. In this example, a large size cluster shows up as the value of α increases.

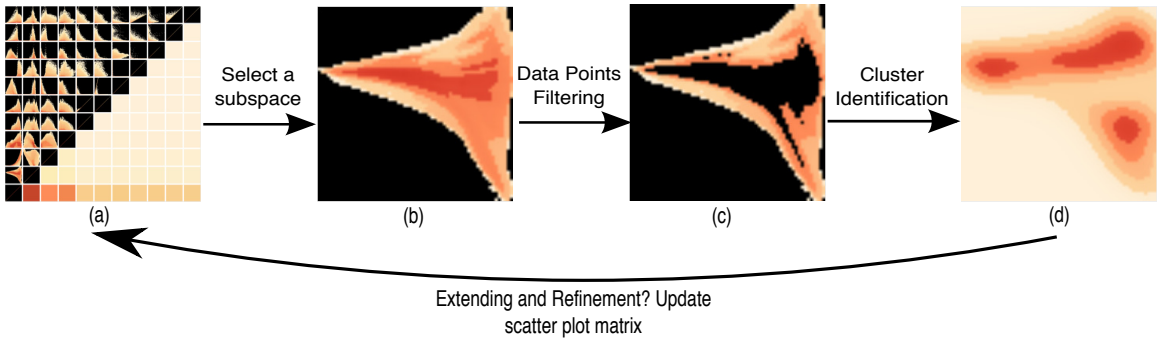


Figure 6.5: Basic steps of our bottom-up subspace exploration.

6.3 Bottom-up Subspace Exploration

In this section, we discuss our interactive bottom-up subspace exploration and cluster identification method. Our interactive bottom-up subspace exploration mainly contains the following steps as shown in Figure 6.5:

- **Step 1: Subspace Selection:** The first step is to pick a subspace to explore. When the number of variables is large, the number of subspaces we have is also large and thus

make the selection of which subspace to explore difficult. We provide an entropy-based method to guide users to choose the next subspace to explore. The details of the entropy-based selection is discussed in Section 6.3.1.

- **Step 2: Data points filtering:** After users select a subspace to explore, the data points with scatter weight larger than a user-defined threshold are filtered, since those data points are more scattered when extending to a larger subspace. After removing those points, density estimation could be applied to the remaining data points to identify clusters that can be well defined in the currently exploring subspace. In our system, users can adjust the threshold with a slider bar.

- **Step 3: Cluster identification:** Weighted kernel density estimation is applied to find the dense regions that are potentially corresponding to interesting clusters. As described in Section 6.2.2, the weight is decided by blending scatter and size weights. In our system, we provide a slider bar to allow users to easily adjust the value of blending coefficient. Details of the cluster identification is described in Section 6.3.2.

- **Step 4: Extending and refinement:** Users can further refine the currently identified cluster by extending to a larger subspace. Details of the extending and refinement is described in Section 6.3.3.

6.3.1 Entropy-based Subspace Selection

The first step of our exploration is to select the next subspace to explore. Difficulties arise when there are a larger number of subspaces. In our system, for each subspace, we evaluate the information contained in the variables that define the subspace using information entropy. Then, this information is used to guide users' choice of the next subspace to

explore. The subspace with a higher entropy tends to contain more information about the data and thus should be explored first. Given a probability distribution P(X) for a random variable X, the Shannon entropy is defined as:

$$H(X) = -\sum P(X) log P(X) \tag{6.7}$$

When a group of variables $X_1, ..., X_n$ are considered, the joint entropy is defined as:

$$H(X_1, ..., X_n) = -\sum P(X_1, ..., X_n) log P(X_1, ..., X_n) \tag{6.8}$$

In the preprocessing stage, we do data binning and aggregation to quantize the data points and then those subspace data points are generated through orthogonal projection. If we consider the subspace of variable U and V, and suppose the number of bins used for variable U and V during the preprocessing stage is B_u and B_v, then each projected point in the subspace of U and V corresponds to a non-zero entry of the joint histogram of U and V with B_u and B_v bins respectively. The size weight w_{size} of a projected point is the frequency of the corresponding non-zero entry. Given the normalized size weight W_{ns} such that the summation of all projected points' normalized size weight equals to one, then the information contained by a subspace S can be evaluated by using the following formula:

$$H(S) = -\sum W_{ns} log W_{ns} \tag{6.9}$$

which is the entropy of the joint distribution of variable U and V. The entropy values of different subspaces are shown in the lower triangle of the scatter plot matrix subplot as shown in Figure 6.5.A.

6.3.2 Cluster Identification

Density Estimation

Once users select the subspace to explore, users can use our system to identify interesting clusters existing in the currently exploring subspace. In our system, we use weighted kernel density estimation to estimate the density and the dense regions are potentially corresponding to interesting clusters. Kernel density estimation(KDE) is a nonparametric technique to estimate the probability density function from a number of data points. Given a set of N data points: $X_1, X_2, ..., X_N$ and a weight w_i associated with each data point X_i, a weighted kernel estimator based on this set of data points is defined as:

$$f(x) = \frac{1}{h}\sum_{i=1}^{N} w_i K(\frac{x - X_i}{h}) \tag{6.10}$$

where K is the kernel function and h is the kernel bandwidth. The weight is computed by blending size weight and scatter weight as described in Section 6.2.2. In our system we use the Gaussian kernel function:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{6.11}$$

When the data points are in multi-dimensions, multivariate kernel density estimation need to be used:

$$f(x) = \sum_{i=1}^{N} w_i \frac{1}{det(H)} K(H^{-1}(x - X_i)) = \sum_{i=1}^{N} w_i K_H(x - X_i) \tag{6.12}$$

where H is the bandwidth matrix and in our system, we assume H is a diagonal matrix:

$$H = diag(h_1, h_2, ..., h_d) \tag{6.13}$$

where d is the number of dimensions and h_i is the bandwidth for dimension i.

The bandwidth is an important parameter for KDE which decides the quality of the density estimation. A good bandwidth selection should minimize the mean integrated squared

error of the kernel density estimation with the true underlying probability density function. To determine the bandwidth for our kernel density estimation, we employ Silverman's rule of thumb [93].

Cluster Representation

Once users identify a dense region in the density estimation result that is potentially corresponding to an interesting cluster, the users can select the cluster and a Gaussian component g centered at the user-selected cluster is constructed to represent the selected cluster. Users can click on the density estimation subplot to select the corresponding cluster. Let set U denote all the variables that define the current subspace, set W denote all the variables in the dataset and then set $V = \{W - U\}$ contains all the variables that have not been added to g. The Gaussian component g associated with a cluster selected in the currently explored subspace is defined as:

$$g = \{\vec{m}, H\} \tag{6.14}$$

where $\vec{m}$ is a vector which denotes the means of variables in U and H is a diagonal matrix of which each diagonal element denotes the variance of a variable in U. The mean value is chosen based on users' click position on the density estimation subplot and the variance can be adjusted by dragging the corresponding Gaussian component on the parallel coordinate subplot. To visualize the user-selected cluster using volume rendering, given an original multivariate data point p_i, all the data values for variables from U form a sample data vector $\vec{v}_i$. The multivariate Gaussian transfer function maps the data point p_i to an opacity value:

$$GTF(\vec{v}_i) = a \times e^{-\frac{(\vec{v}_i - \vec{m})^T H^{-1} (\vec{v}_i - \vec{m})}{2}} \tag{6.15}$$

where a is a constant to scale the opacity value. The data points that have values closer to the cluster center will have high opacity, and therefore will be highlighted in the volume

rendering result. Each Gaussian component is also assigned a unique color. Each Gaussian component covers a subset of data points in Q, a data point is denoted as covered by a Gaussian component g if it is within two standard deviations of the mean of g. Once users decide to add a Gaussian component g to the Gaussian Transfer Function, the data points covered by g are removed from Q since they are already classified and the scatter plot matrix subplot is also updated. In this way, the points that are shown in the scatter plot subplot are always the points which are not covered by our specified Gaussian Components and users can focus on exploring those points.

Suppose the user identifies n interesting clusters through our system, and the i_{th} cluster has a Gaussian component g_i associated with it and is assigned a unique color C_i. All the n clusters can be rendered simultaneously to depict interesting features of the data. The final color C and opacity α are obtained by combining the color and opacity generated by each Gaussian component together:

$$C = \frac{\sum_{i=1}^{n} \alpha_i C_i}{\sum_{i=1}^{n} \alpha_i}, \qquad \alpha = \sum_{i=1}^{n} \alpha_i \tag{6.16}$$

6.3.3 Extending and Refinement

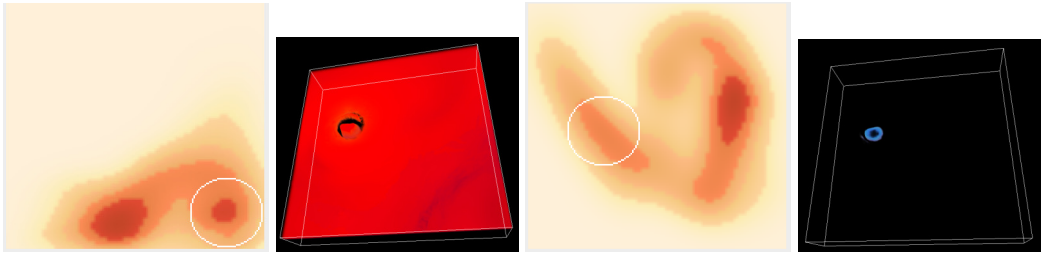
Once users identify an interesting cluster in the currently exploring subspace, the users can further refine the cluster by extending the subspace to a larger subspace. Whether a specified cluster needed to be further refined can be decided based on the following two criteria:

- **Scatter:** Scatter indicates how likely a cluster can be further refined. The cluster defined by data points with larger scatter weight usually needs to be further refined.

- **Overlapping:** Given the previously specified Gaussian components $G_1, ..., G_i$ and the currently specifying Gaussian component G_c, if the region covered by G_c has a large overlap with the region covered by all the previously specified Gaussian components, usually, an extending and refinement operation is needed to further refine this region to minimize the overlap or we can simply discard the currently exploring region and explore other regions that have a smaller overlap with the previously specified Gaussian components. This will prevent us from exploring the same region multiple times. A Multidimensional Scaling subplot is used here to assist users to identify how much overlap there is between G_c and all those previously specified Gaussian components. All the data points covered by those previously specified Gaussian components and the currently specifying Gaussian components are highlighted in the MDS subplot, so the users can visually identify how much overlap there is.

The cluster refinement includes three steps: removing data points that are not covered by the selected cluster, updating the subspace matrix, and refining the cluster in an extended subspace.

Removing data points that are not covered by the selected cluster. Once users identify and select a cluster in the current subspace, a Gaussian component is constructed to represent the selected cluster as described in Section 6.3.2. We remove the data points which are not covered by the Gaussian component. As defined in Section 6.3.2, a data point is said to be covered by a Gaussian component g if its value is within two standard deviations of the mean of g. The remaining data points are then used for the subsequent exploration.



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

Figure 6.6: Bottom-up dense region refinement. We start with an empty Gaussian component g={}. Initially we select the subspace of Temperature and Velocity. (a). The density estimation of the data points in the subspace of Temperature and Velocity. The dense region within the white cycle is selected by the user. (b). The volume rendering highlights the selected region. The Gaussian component is also updated. Temperature and Velocity are added to g. Now, g is updated to $\{(Temperature, 0.75, 0.0049), (Velocity, 0.2, 0.0049)\}$, where 0.75 and 0.2 are the means for temperature and velocity respectively, 0.0049 is the variance for temperature and velocity. Then, data points are filtered if they are not covered by g. A new set of data points are generated and the subspace matrix is updated. (c). The user selects the Pressure and QVapor subspace in the updated subspace matrix. Now, the current subspace is defined by Temperature, Pressure, QVapor and Velocity. The user explores the subspace by filtering out data points with scatter weights larger than a user-defined threshold and then performs density estimation. The region within the white cycle is selected by the user. (d). The volume rendering highlights the selected region. Pressure and QVapor are added to g. g is extended to a 4 dimensional Gaussian component, (Temperature,0.75,0.0049), (Pressure,0.3,0.0049),(Qvapor,0.5,0.0049), (Velocity,0.2,0.0049).

Updating the subspace matrix. With the new set of data points, the subspace matrix is updated. The size weight and scatter weight are recalculated for each data point. Suppose the selected cluster is in the subspace defined by variables A and B, then all subspaces involving A and B are also removed from the scatter plot matrix subplot. The entropies for the remaining subspaces are also recalculated.

Refining clusters in the extended subspace. With the updated subspace matrix, the user can navigate through different new subspaces to refine the cluster and identify interesting clusters.

An example of this bottom-up cluster refinement is shown in Figure 6.6. Initially, the user explores the subspace of Temperature and Velocity and a cluster defined in this subspace is selected. Then, the user further refines the cluster by extending the subspace to include the variables Pressure and QVapor. From the volume rendering image, we can see the region around the hurricane eye is selected based on the previous selection in Temperature and Velocity subspace. The corresponding Gaussian function g is also extended from two dimensions to four dimensions.

6.4 Results

In this section, we demonstrate the effectiveness of using our system to analyze and visualize several multivariate volumetric datasets. Three datasets were used. For the data binning and aggregation, the number of bins was set to 64 for each variable.

6.4.1 Hurricane Isabel Data Set

The Hurricane Isabel data set was used for this case study. Eleven variables were used for the experiments: Temperature, Pressure, Wind speed magnitude (Velocity), Water vapor mixing ratio (QVAPOR), Cloud moisture mixing ratio (QCLOUD), Total cloud moisture mixing ratio (CLOOUD), Cloud ice mixing ratio (QICE), Rain mixing ratio (QRAIN), Snow mixing ratio (QSNOW), Graupel mixing ratio (QGRAUP) and Total precipitation mixing ratio (PRECIP). Time step 7 was selected for our experiment.

Figure 6.7 shows a typical exploration using the Hurricane Isabel dataset. Initially, a scatter plot matrix constructed from all the eleven variables is presented. All the single variable and two variables subspaces are shown in a matrix juxtaposition view, so we can easily compare different subspaces. The entropy values for different subspaces are shown in the lower triangle of the matrix subplot. We can navigate through different subspaces and then

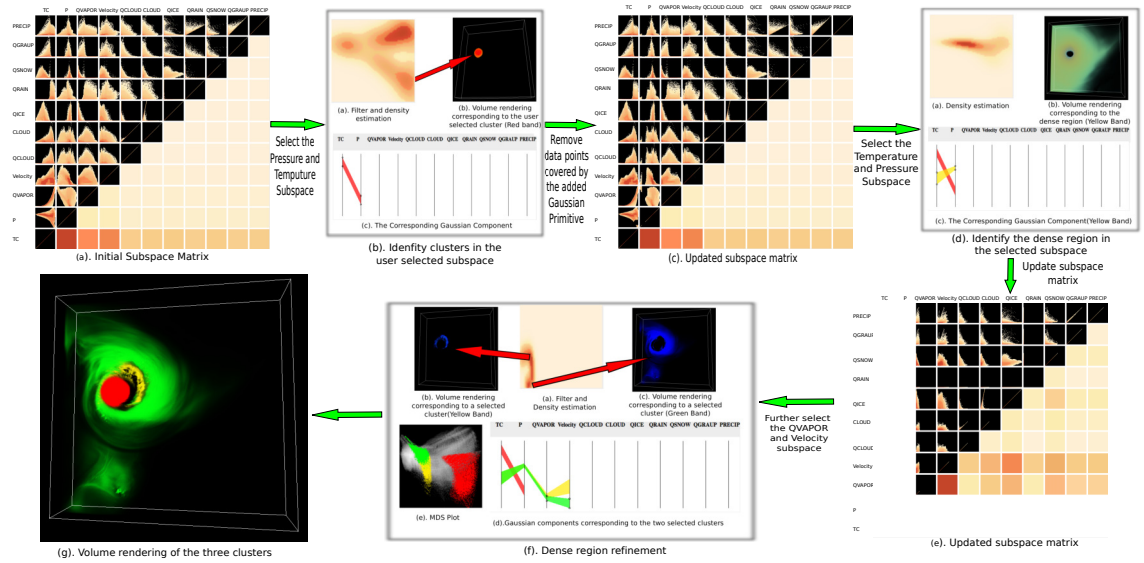


Figure 6.7: Experiments on the Hurricane Isabel dataset. Details about the exploration process are discussed in Section 6.4.1.

a zoom-in view of the selected subspace is shown so that we can investigate the selected subspace in more detail. By investigating the lower entropy triangle, the Pressure and Temperature subspace is selected to explore because it has the highest entropy. Figure 6.7.(b) shows the exploration process in the selected subspace. A filter operation is applied first to remove data points with scatter weights larger than 0.6, and then a weighted kernel density estimation is performed on the remaining data points with a blending coefficient 0.1 to find clusters that can be well defined in the current subspace. Figure 6.7.(b.a) shows the density estimation result. Roughly three clusters can be identified from the density estimation result. We can then click and select different dense regions to see the corresponding volume rendering result. In the example here, the Hurricane eye corresponding to one dense region is identified and we add its associated Gaussian component to the Gaussian Transfer Function. After the specified Gaussian component is added to the Gaussian Transfer Function,

the data points that are covered by the Gaussian component are removed and the scatter plot matrix is updated as shown in Figure 6.7.(c). Then, we begin another exploration from the updated scatter plot matrix. We continue to explore the Temperature and Pressure subspace as this subspace still has the highest entropy. Figure 6.7.(d) shows the exploration process. Weighted kernel density estimation is directly applied to identify the dense region. Then, a Gaussian component is placed centered at the dense region to select this region. The Gaussian component is shown as the yellow band in the parallel coordinate. The volume region corresponding to the dense region is shown in Figure 6.7.(d.b). Figure 6.7.(e) shows the updated scatter plot matrix by removing the data points that are not in the selected dense region. Subspaces which contain Temperature and Pressure are removed. After updating, the lower entropy triangle is also updated. We then navigates to the QVapor and Velocity subspace which has the highest entropy to further refine the selected region in Temperature and Pressure subspace. Now, the subspace is extended to contain Temperature, Pressure, QVapor, and Velocity. The data points with scatter weights larger than 0.6 are filtered out first. Figure 6.7.(f.a) shows the density estimation result on the remaining data points with a blending coefficient 0.1. Two dense regions are identified from the result. The volume regions corresponding to those two regions are shown in Figure 6.7.(f.b) and Figure 6.7.(f.c). Their associated Gaussian components are shown in the parallel coordinate as the yellow and green bands. The two associated Gaussian components are now in four dimensions given the four-dimensional subspace. The regions covered by those three Gaussian components are shown in the MDS subplot as shown in Figure 6.7.(f.e). Figure 6.7.(g) shows the final volume rendering result by rendering all the three clusters together.

As we can see from the results, features in a multivariate dataset usually can be well defined with a few number of variables instead of all the variables. In the above example,

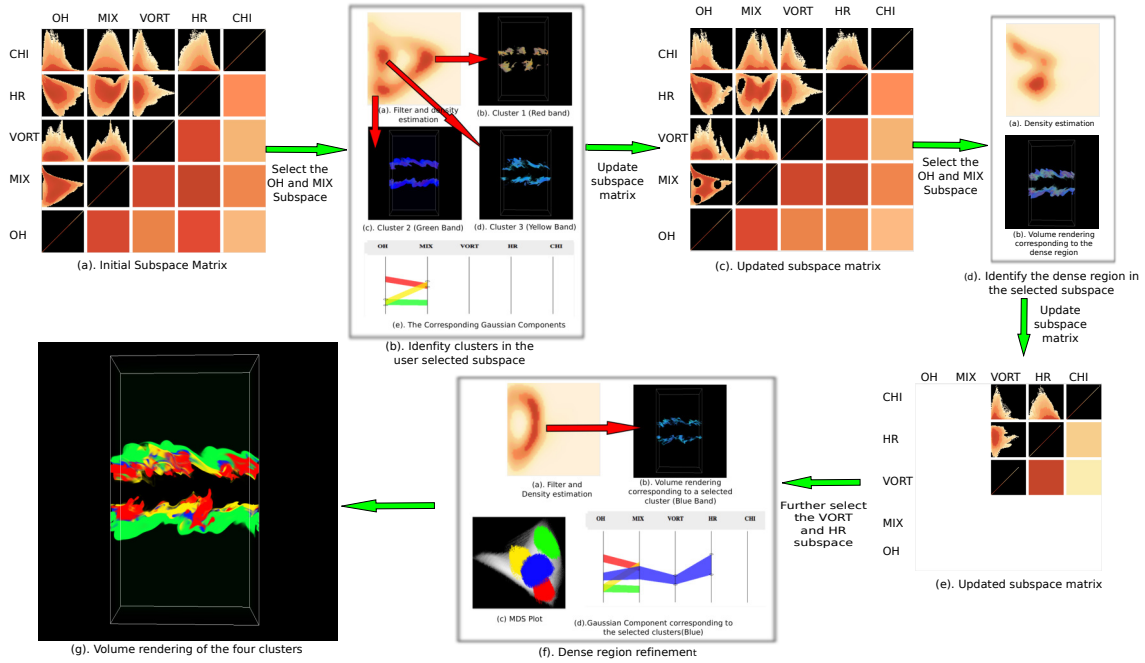


Figure 6.8: Experiments on the Turbulent Combustion dataset. Details about the exploration process are discussed in Section 6.4.2.

the Hurricane eye is identified easily in the Pressure and Temperature subspace without considering the other 9 variables.

6.4.2 Turbulent Combustion Data Set

The Turbulent Combustion simulation data set is produced by Dr. Jackqueline Chen at Sandia Laboratories through US Department of Energy's SciDAC Institute for Ultrascale Visualization. The data has a resolution of $480 \times 720 \times 120$. There are five variables in this data set: Mixture Fraction of Hydroxyl Radical(OH), Mixture Fraction(MIX), Vorticity(VORT), Heat Release Rate(HR) and Scalar Dissipation Rate(CHI). Time step 41 was selected for our experiment.

Figure 6.8 shows a user exploration process to identify interesting features that exist in the data. The initial scatter plot matrix is shown in Figure 6.8.(a). Initially, the subspace of MIX and OH is selected according to the lower entropy triangle. The data points with scatter weights larger than a user-defined threshold 0.9 are filtered out. Weighted kernel density estimation is then applied to the remaining data points with a blending coefficient 0.1 and Figure 6.8.(b.a) shows the result. Three dense regions could be identified from the density estimation result. Two dense regions are corresponding to MIX value around 0.5. The Mixture Fraction variable represents the portion of fuel and oxidizer. Its values ranges from 0 and 1 while 0 represents pure oxidizer and 1 represents pure fuel. The value of MIX generally quantifies different characteristics of the flame: a fully burning flame is characterized by the region which has a larger chemical reaction rate than the turbulent mixing rate. Local extinction occurs when the turbulent mixing rate exceeds the chemical reaction rate. As reported in [2], the stoichiometric mixture fraction 0.42 corresponds to the flame. The top two dense regions that are identified are corresponding to the flame area. As can be seen, the flame area is further divided into two regions based on the value of the OH. As described in [2], some areas of the region corresponding to stoichiometric mixture fraction might be only weakly burning, and might reignite independently or with the help of the neighboring burning flame elements. This area can be characterized by low radical, temperature and heat release rate. By taking into account OH, the flame is further divided into two regions: the region with a lower OH value is corresponding to a flat region while the region with a higher OH value is corresponding to a more turbulent region as shown in the volume rendering view. The third cluster is corresponding to low MIX and low OH region. Figure 6.8.(b.e) shows the Gaussian components for the three clusters. Figure 6.8.(c)

shows the updated scatter plot matrix after adding those three Gaussian components to the Gaussian Transfer Function.

Next, we continue exploring the OH and MIX subspace according to the lower entropy triangle. Weighted kernel density estimation is directly applied and the result is shown in Figure 6.8.(d.a). Figure 6.8.(d.b) shows the volume region corresponding to the dense region. In order to further refine the dense region in a larger subspace, the scatter plot matrix is updated by removing the data points that are not inside the dense region. Figure 6.8.(e) shows the updated scatter plot matrix. The subspaces involve MIX and OH are removed. The lower entropy triangle shows the VORT and HR subspace has the highest entropy, so we navigate to the VORT and HR subspace to further refine the selected region in MIX and OH subspace. Now the subspace is extended to four dimensions. The data points with scatter weights larger than 0.7 are filtered out first. Figure 6.8.(f.a) shows the weighted density estimation result on the remaining data points with a blending coefficient 0.1. The volume region corresponding to the dense region is shown in Figure 6.8.(f.b). The corresponding Gaussian component is shown in Figure 6.8.(f.d) as the blue band. Notice that, now the Gaussian component is extended to four dimensions. The regions covered by those four Gaussian components are highlighted in the MDS subplot as shown in Figure 6.8.(f.c). Figure 6.8.(g) shows the volume rendering results by rendering all the four clusters together.

6.4.3 Ionization Front Instability Data Set

In the third case study, we use the Ionization Front Instability dataset. The data was created by Mike Norman and Daniel Whalen which aimed at understanding the effect of instabilities where radiation ionization fronts scatter around primordial gas [106] [107].

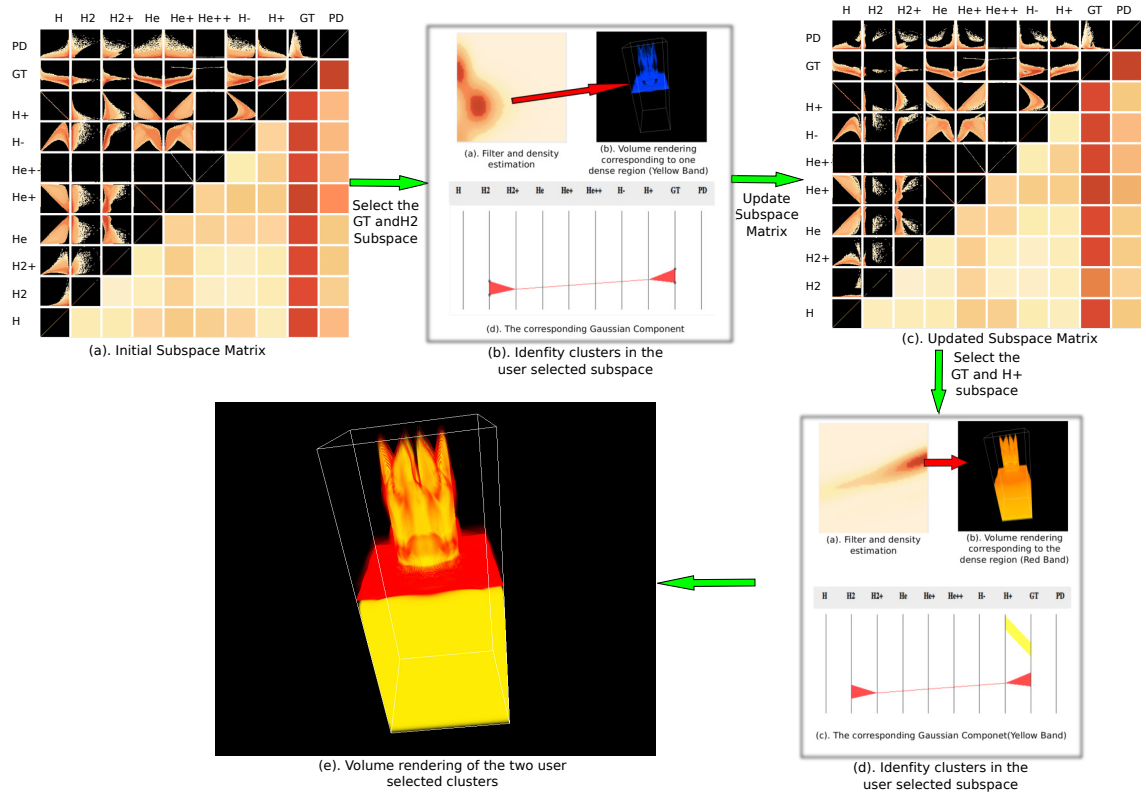


Figure 6.9: Experiments on the Ionization Front Instability dataset. Details about the exploration process are discussed in Section 6.4.3

Ionization front is the region that separates gas in an ionized state from gas in a neutral state. The resolution of the data is $600 \times 248 \times 248$. It contains ten variables which include particle density, gas temperature and eight chemical species including the mass abundance of $H, H^+, He, He^+, He^{++}, H^-, H_2, H_2^+$.

To discover interesting features in the data, an exploration process is shown in Figure 6.9. The initial scatter plot matrix is shown in Figure 6.9.(a). First, we explore the gas temperature and H_2(gaseous hydrogen) subspace based on the lower entropy triangle. The data points with scatter weights larger than 0.6 are filtered out first. Figure 6.9.(b.a)

shows the weighted density estimation result on the remaining data points with a blending coefficient 0.2. Roughly, two dense regions can be identified. One dense region that is corresponding to relative lower temperature and higher H_2 is shown in Figure 6.9.(b.b). Its associated gaussian component is shown as the red band in Figure 6.9.(b.c). This region is corresponding to the shocked state. Figure 6.9.(c) shows the updated scatter plot matrix after adding the Gaussian component to the Gaussian Transfer Function. Then, we explore the gas temperature and H^+(ionized hydrogen) subspace. The data points with scatter weights larger than a user-defined threshold 0.2 are filtered out. Weighted kernel density estimation is then applied on the remaining data points with a blending coefficient 0.1. Figure 6.9.(d.a) shows the result. Figure 6.9.(d.b) shows the volume region corresponding to the dense region. The temperature of this region is around 15,000K which is corresponding to the ionized gas region [74]. This region also has a high value for H^+ as shown in the transfer function view. Figure 6.9.(d.d) shows the regions that are covered by the two Gaussian components in the MDS view. Figure 6.9.(e) shows the volume rendering results of all the two clusters.

6.5 Performance

Data Set	Size(MB)	# Samples before binning and agg.	# Samples after binning and agg.
Isabel	1239.8	25,000,000	567,235
Comb.	791	41,472,000	1,669,290
Ion	1407.7	36,902,400	213,824

Table 6.1: **Data Set:** The size, number of samples before binning and aggregation and number of samples after binning and aggregation for the three data sets we used.

Data Set	# Samples	$T_l(sec)$	$T_c(sec)$	$T_s(sec)$
Isabel	567,235	1.38	13.25	1.02
Comb.	1,669,290	1.69	14.53	1.29
Ion	213,824	3.06	19.95	0.37

Table 6.2: **System Setup Time:** The system setup time for Isabel, Combustion and Ion Front. The system setup time are mainly spent on data loading (T_l), binning and aggregation (T_c) and initialize the scatter plot matrix subplot (T_s). The second column shows the number of data points generated after binning and aggregation.

Data Set	$T_{ds}(sec)$	$T_{pcp}(sec)$	$T_{extend}(sec)$	$T_{add}(sec)$
Isabel	0.33	0.32	0.53	0.1 - 1.21
Comb.	0.97	1.09	1.01	0.07 - 3.93
Ion	0.11	0.13	0.18	0.11 - 0.406

Table 6.3: **System Response Time:** The system response time when users perform the following operations: T_{ds} : brushing 100% percentage of the data points on the density estimation subplot; T_{pcp} : brushing 100% percentage of the data points on the parallel coordinate subplot; T_{extend} : Extending to a larger subspace when 50% percentage of data points are selected; T_{add} : Adding a specified Gaussian component to Gaussian Transfer Function.

Our system was implemented in C++. The interface was designed using QT and the rendering part was implemented with OpenGL. In this Section, we measure and report the performance of our system. The machine we used has an Intel Core i7-4870HQ CPU @ 2.5GHz with 16 GB memory. We tested the performance of our system with three datasets: Hurricane Isabel, Combustion, and Ion Front. Table 6.1 shows the size of the three data sets used and the number of data samples before and after binning and aggregation. Table 6.2 shows the system setup time for those three datasets. The second column shows the number of samples generated after binning and aggregation. T_l represents the time to load data into

memory. T_c denotes the time to do binning and aggregation and T_s represents the time to initialize the scatter plot matrix subplot. For the multidimensional scaling subplot, we precompute the low dimensional embedding and save it as a texture. During system startup, the time to initialize MDS subplot is negligible(less than 0.1 second), so we did not report this time here.

We also measured the system response time during exploration. T_{ds} and T_{pcp} denote the system response time when users brush on the density estimation and parallel coordinate subplot. These two brushing operations will change the mean and variance of the current Gaussian component, and then the points that are covered by the Gaussian component are computed. If more data points are covered, the longer the system will respond. In Table 6.3, we reported the time when brushing 100% percentage of the data points. T_{extend} denotes the system response time when performing the extending operation to extend to a larger subspace. The response time taken by the extending operation is related to the number of data points that are covered and the number of remaining variables. In our experiments, we report the time when roughly 50% data points are covered. T_{add} denotes the system response time when users add a specified Gaussian component to Gaussian Transfer Function. The time taken by this operation is influenced by the number of data points needed to be removed, the remaining number of data points and the remaining number of variables. In table 6.3, we reported a range instead of a single number to represent the system response time.

6.6 Conclusion

In this chapter, we present a novel system for users to analyze and visualize multivariate volumetric data through bottom-up subspace exploration. In our system, we use information derived from the data to assist user exploration. An entropy-based method is used to help users to identify which subspace to explore. Size weight and scatter weight are proposed as a complementary information for the loss of high dimensional information due to projection. Based on those two weights, we employ weighted kernel density estimation and bottom-up subspace exploration in our system to assist users to explore and identify interesting features in the data. A Gaussian component is constructed to represent each user-selected cluster. Finally, all these selected clusters can be rendered together to reveal the feature of the data.

One limitation of our approach is that as the number of variables increases, the number of subspaces need to show in the scatter plot matrix also increases. When a large number of subspaces need to be shown, the size of the scatter plot matrix could be very large and make it difficult to be shown in a limited screen size. This limitation prohibits our approach to scale to a large number of variables. One way to solve this limitation is to choose only a subset of subspaces to show in the scatter plot matrix subplot. For example, we could rank subspaces based on their entropy and then only show the top K subspaces.

In the future, we plan to extend our system to support time-varying data. It would be interesting to investigate whether the subspace exploration is helpful for feature detection and tracking in time-varying data.

Chapter 7: Conclusion

Since the computing power of high performance machines has been increasing exponentially in the last few decades, the size of data generated by various scientific simulations is growing rapidly. As the size of data continues to increase, it is important to design efficient techniques to visualize and explore scientific data sets that are produced from simulations in many science and engineering disciplines. This dissertation presents a generalized distribution-based query-driven framework to analyze large scale scientific data sets. We applied our proposed framework to two general forms of scientific data: flow fields and multivariate scalar fields.

Chapter 3 and 4 present our methods for analyzing flow fields by applying our proposed framework. In order to obtain statistical summarization of flow fields, we first compute geometries from flow fields and then compute the statistical summarization from those computed geometries. When the data size is large, the time taken to compute geometries is expensive. One way to achieve efficient computation is to utilize distributed memory parallelism. Then, traditional algorithms need to be revised to fully utilize the computing power and achieve high parallel efficiency. It is nontrivial to design a parallel algorithm to compute stream surfaces efficiently due to the strong dependency in computing the positions of the particles forming the stream surface front. Also, because of the adaptive refinement of the stream surface front, it is difficult to estimate the workload associate with

each seeding curve or seeding curve segment, which makes the load balancing computation of stream surfaces nontrivial. In Chapter 3, we describe a new parallel algorithm based on the traditional front advancing algorithm to support efficient and scalable computation of stream surfaces. In our algorithm, the seeding curves are divided into segments and then assigned to different processes. During the computation, each process integrates the seeding curve segments assigned to compute different stream surface patches in parallel. Several strategies are applied to improve the overall performance of our algorithm. Loading data from other processes instead of from disk reduces the I/O cost. The number of data loads is reduced by utilizing cache. Load imbalancing problem is solved by employing runtime seeding curve segment subdivision and work stealing. The strong scalability of our algorithm is demonstrated by using several large-scale flow field data sets.

After we compute those geometries such as streamlines and stream surfaces from flow fields, those computed geometries need to be visualized to help users understand and analyze the data. However, a direct visualization of those densely computed geometries is usually not useful because of visual cluttering and occlusion. In Chapter 4, a statistical distribution representation is computed for each streamline and then we propose a query-driven framework to explore and analyze flow field data sets. A new method to measure the similarity between streamlines based on the statistical distributions of geometric measures along the trajectories of streamlines is proposed. Compared to the previous methods, our proposed similarity measurement is invariant to translation and rotation, and also we can evaluate the similarity between streamlines much faster. Based on the presented similarity measurement metric, we designed a framework to interactively explore 3D vector fields through streamline query and clustering which solves the visual cluttering and occlusion problems.

Chapter 5 and 6 present our methods to analyze multivariate scalar fields by applying our proposed framework. In Chapter 5, we present a query-driven framework to explore and analyze multivariate scalar fields through multivariate histogram. In order to reduce the storage cost, we propose a novel compact structure to store a multivariate histogram. We also present several histogram query operations based on our new histogram representation to efficiently derive other different types of histograms. Finally, several query-driven visualization applications to analyze and visualize the data using our technique are presented. In Chapter 6, we present an interactive system to allow users to design multivariate transfer function through bottom-up subspace exploration effectively. The original data is summarized through data binning and aggregation. Then, an interactive system is designed based on those summarized data points. By using the proposed system, users can design effective multivariate transfer functions to highlight regions of interest in large-scale multivariate scalar data sets.

As we are approaching the age of exaflop computing, much more data than we have today will be generated. For data at such a large scale, it is important to design efficient techniques for visualization and analysis. The methods introduced in this dissertation present several different approaches to visualize and analyze large scale data such as using distributions to summarize the data and then analyzing the data based on those distributions. Future research in visualization and analysis will be quite important for us to face the continuing increase of data generated from various simulations.

Bibliography

[1] Sean Ahern, Arie Shoshani, Kwan-Liu Ma, Alok Choudhary, Terence Critchlow, Scott Klasky, Valerio Pascucci, Jim Ahrens, E. Wes Bethel, Hank Childs, Jian Huang, Ken Joy, Quincey Koziol, Gerald Lofstead, Jeremy S Meredith, Kenneth Moreland, George Ostrouchov, Michael Papka, Venkatram Vishwanath, Matthew Wolf, Nicholas Wright, and Kensheng Wu. *Scientific Discovery at the Exascale, a Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*. 2011.

[2] H. Akiba, K. l. Ma, J. H. Chen, and E. R. Hawkes. Visualizing multivariate volume data from turbulent combustion simulations. *Computing in Science Engineering*, 9(2):76–83, March 2007.

[3] Hiroshi Akiba, Nathaniel Fout, and Kwan-Liu Ma. Simultaneous classification of time-varying volume data based on the time histogram. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'06, pages 171–178, 2006.

[4] S. Arens and G. Domik. A survey of transfer functions suitable for volume rendering. In *Proceedings of the 8th IEEE/EG International Conference on Volume Graphics*, VG'10, pages 77–83, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[5] C.L. Bajaj, V. Pascucci, and D.R. Schikore. The contour spectrum. In *Visualization '97., Proceedings*, pages 167–173, Oct 1997.

[6] Donald J. Berndt and James Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *KDD Workshop*, pages 359–370, 1994.

[7] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943.

[8] J. Blaas, C. Botha, and F. Post. Extensions of parallel coordinates for interactive exploration of large multi-timepoint data sets. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1436–1451, Nov 2008.

[9] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, 1999.

[10] S. Born, A. Wiebel, J. Friedrich, G. Scheuermann, and D. Bartz. Illustrative stream surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1329–1338, Nov 2010.

[11] Anders Brun, Hae-Jeong Park, Hans Knutsson, and Carl-Fredrik Westin. Coloring of dt-mri fiber traces using laplacian eigenmaps. In *EUROCAST '03: Computer Aided Systems Theory*, pages 518–529, 2003.

[12] J. J. Caban and P. Rheingans. Texture-based transfer functions for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1364–1371, Nov 2008.

[13] D. Camp, H. Childs, C. Garth, D. Pugmire, and K.I. Joy. Parallel stream surface computation for large data sets. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 39–47, 2012.

[14] D. Camp, C. Garth, H. Childs, D. Pugmire, and K.I. Joy. Streamline integration using mpi-hybrid parallelism on a large multicore architecture. *Visualization and Computer Graphics, IEEE Transactions on*, 17(11):1702–1713, 2011.

[15] David Camp, Hank Childs, Amit Chourasia, Christoph Garth, and Kenneth I. Joy. Evaluating the Benefits of An Extended Memory Hierarchy for Parallel Streamline Algorithms. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, 2011.

[16] David Camp, Hari Krishnan, David Pugmire, Christoph Garth, Ian Johnson, E. Wes Bethel, Kenneth I. Joy, and Hank Childs. Gpu acceleration of particle advection workloads in a parallel, distributed memory setting. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, pages 1–8, 2013.

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

[18] J. Chanussot, A Clement, B. Vigouroux, and J. Chabod. Lossless compact histogram representation for multi-component images: application to histogram equalization. In *Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International*, volume 6, pages 3940–3942, July 2003.

[19] A Chaudhuri, Teng-Yok Lee, Bo Zhou, Cong Wang, Tiantian Xu, Han-Wei Shen, T. Peterka, and Yi-Jen Chiang. Scalable computation of distributions from large scale data sets. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 113–120, Oct 2012.

[20] A Chaudhuri, Tzu Hsuan Wei, Teng Yok Lee, Han Wei Shen, and T. Peterka. Efficient range distribution query for visualizing scientific data. In *Pacific Visualization Symposium (PacificVis), 2014 IEEE*, pages 201–208, March 2014.

[21] Chun-Ming Chen and Han-Wei Shen. Graph-based seed scheduling for out-of-core ftle and pathline computation. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, pages 15–23, Oct 2013.

[22] Chun-Ming Chen, L. Xu, T. Lee, H. Shen, Lijie Xu, Teng-Yok Lee, and Han-Wei Shen. A flow-guided file layout for out-of-core streamline computation. In *Pacific Visualization Symposium (PacificVis), 2012 IEEE*, pages 145–152, Feb 2012.

[23] Li Chen and I. Fujishiro. Optimizing parallel performance of streamline visualization for large distributed flow datasets. In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pages 87–94, March 2008.

[24] Scott Cohen and Leonidas Guibas. The earth mover's distance under transformation sets. In *ICCV '99: Proceedings of the International Conference on Computer Vision*, volume 2, pages 1076–, 1999.

[25] I. Corouge, S. Gouttard, and G. Gerig. Towards a shape model of white matter fiber bundles using diffusion tensor mri. In *IEEE International Symposium on Biomedical Imaging: Nano to Macro*, pages 344 – 347 Vol. 1, 2004.

[26] C. Correa and K. L. Ma. Size-based transfer functions: A new volume exploration technique. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1380–1387, Nov 2008.

[27] C. Correa and K. L. Ma. The occlusion spectrum for volume classification and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1465–1472, Nov 2009.

[28] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[29] David Degani, Arnan Seginer, and Yuval Levy. Graphical visualization of vortical flows by means of helicity. *AIAA Journal*, 28(8):1347–1352, 1990.

[30] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 53:1–53:11, 2009.

[31] H. Doleisch. Simvis: Interactive visual analysis of large and time-dependent 3d simulation data. In *2007 Winter Simulation Conference*, pages 712–720, Dec 2007.

[32] Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the Symposium on Data Visualisation 2003*, VISSYM '03, pages 239–248, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[33] D.Sujudi and R.Haimes. Integration of particles and streamlines in a spatially-decomposed computation. In *Proceedings of Parallel Computational Fluid Dynamics*, 1996.

[34] Shiaofen Fang, T. Biddlecome, and M. Tuceryan. Image-based transfer function design for data exploration in volume visualization. In *Visualization '98. Proceedings*, pages 319–326, Oct 1998.

[35] Nissim Francez. Distributed termination. *ACM Trans. Program. Lang. Syst.*, 2(1):42–55, 1980.

[36] Christoph Garth, Xavier Tricoche, Tobias Salzbrunn, Tom Bobach, and Gerik Scheuermann. Surface techniques for vortex visualization. In *Proceedings of the Sixth Joint Eurographics - IEEE TCVG Conference on Visualization*, pages 155–164, 2004.

[37] S. Goil and A. Choudhary. Sparse data storage schemes for multidimensional data for olap and data mining. *Technical Report CPDC-9801-005, Northwestern University*, December 1997.

[38] L.J. Gosink, C. Garth, J.C. Anderson, E.W. Bethel, and K.I. Joy. An application of multivariate statistical analysis for query-driven visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 17(3):264–275, March 2011.

[39] Yi Gu and Chaoli Wang. Transgraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2015–2024, Dec 2011.

[40] 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, Sept 2012.

[41] M. Haidacher, D. Patel, S. Bruckner, A. Kanitsar, and M. E. Gröller. Volume visualization based on statistical transfer-function spaces. In *Visualization Symposium (PacificVis), 2010 IEEE Pacific*, pages 17–24, March 2010.

[42] Taosong He, Lichan Hong, A. Kaufman, and H. Pfister. Generation of transfer functions with stochastic search techniques. In *Visualization '96. Proceedings.*, pages 227–234, Oct 1996.

[43] B. Heckel, G. Weber, B. Hamann, and K.I. Joy. Construction of vector field hierarchies. In *Vis '99: Proceedings of IEEE Visualization*, pages 19 –25, 505, 1999.

[44] J. Heinrich and D. Weiskopf. Continuous parallel coordinates. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1531–1538, Nov 2009.

[45] J. P M Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Visualization, 1992. Visualization '92, Proceedings., IEEE Conference on*, pages 171–178, 1992.

[46] Mathias Hummel, Christoph Garth, Bernd Hamann, Hans Hagen, and Kenneth I. Joy. Iris: Illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1319–1328, November 2010.

[47] A. Inselberg and Bernard Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Visualization, 1990. Visualization '90., Proceedings of the First IEEE Conference on*, pages 361–378, Oct 1990.

[48] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91, 1985.

[49] Heike Janicke, Alexander Wiebel, Gerik Scheuermann, and Wolfgang Kollmann. Multifield visualization using local statistical complexity. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1384–1391, 2007.

[50] Jinhee Jeong and Fazle Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69–94, 1995.

[51] J. Johansson, P. Ljung, M. Jern, and M. Cooper. Revealing structure within clustered parallel coordinates displays. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 125–132, Oct 2005.

[52] C. Ryan Johnson and Jian Huang. Distribution-driven visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):734–746, September 2009.

[53] W. Kendall, M. Glatter, Jian Huang, T. Peterka, R. Latham, and R. Ross. Terascale data organization for discovering multivariate climatic trends. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12, Nov 2009.

[54] Wesley Kendall, Jingyuan Wang, Melissa Allen, Tom Peterka, Jian Huang, and David Erickson. Simplified parallel domain traversal. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 10:1–10:11, 2011.

[55] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: methods and applications. In *Visualization, 2003. VIS 2003. IEEE*, pages 513–520, Oct 2003.

[56] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization*, VVS '98, pages 79–86, 1998.

[57] J. Kniss, S. premoze, M. Ikits, A. Lefohn, C. Hansen, and E. Praun. Gaussian transfer functions for multi-field volume visualization. In *Visualization, 2003. VIS 2003. IEEE*, pages 497–504, Oct 2003.

[58] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the Conference on Visualization '01*, VIS '01, pages 255–262, Washington, DC, USA, 2001. IEEE Computer Society.

[59] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, July 2002.

[60] Andreas H. König and Eduard M. Gröller. Mastering transfer function specification by using volumepro technology, 1999.

[61] Erwin Kreyszig. *Differential geometry*. Dover Publications, 1991.

[62] Solomon Kullback. *Information Theory and Statistics (Dover Books on Mathematics)*. Dover Publications, 1997.

[63] Teng-Yok Lee and Han-Wei Shen. Efficient local statistical analysis via integral histograms with discrete wavelet transform. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2693–2702, Dec 2013.

[64] S. Liu, B. Wang, J. J. Thiagarajan, P. T. Bremer, and V. Pascucci. Multivariate volume visualization through dynamic projections. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 35–42, Nov 2014.

[65] C. Lundstrom, P. Ljung, and A Ynnerman. Local histograms for design of transfer functions in direct volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 12(6):1570–1579, Nov 2006.

[66] R. Maciejewski, I. Woo, W. Chen, and D. Ebert. Structuring feature space: A non-parametric method for volumetric transfer function generation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1473–1480, Nov 2009.

[67] Mahnaz Maddah, W. Eric L. Grimson, Simon K. Warfield, and William M. Wells. A unified framework for clustering and quantitative analysis of white matter fiber tracts. *Medical Image Analysis*, 12(2):191 – 202, 2008.

[68] S. Martin and Han-Wei Shen. Histogram spectra for multivariate time-varying volume lod selection. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 39–46, Oct 2011.

[69] S. Martin and Han-Wei Shen. Transformations for volumetric range distribution queries. In *Visualization Symposium (PacificVis), 2013 IEEE Pacific*, pages 89–96, Feb 2013.

[70] Tony McLoughlin, Robert S. Laramee, and Eugene Zhang. Easy integral surfaces: A fast, quad-based stream and path surface algorithm. In *Proceedings of the 2009 Computer Graphics International Conference*, pages 73–82, 2009.

[71] E Merzari, W Pointer, A Obabko, and P Fischer. On the numerical simulation of thermal striping in the upper plenum of a fast reactor. *Proceedings of ICAPP*, 2010.

[72] B. Moberts, A. Vilanova, and J.J. van Wijk. Evaluation of fiber clustering methods for diffusion tensor imaging. In *Vis '05: Proceedings of IEEE Visualization*, pages 65 – 72, 2005.

[73] C. Muller, D. Camp, B. Hentschel, and C. Garth. Distributed parallel particle advection using work requesting. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, pages 1–6, Oct 2013.

[74] S. Nagaraj and V. Natarajan. Relation-aware isosurface extraction in multifield data. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):182–191, Feb 2011.

[75] B. Nouanesengsy, Teng-Yok Lee, and Han-Wei Shen. Load-balanced parallel streamline generation on large scale vector fields. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):1785–1794, 2011.

[76] M. Novotny and H. Hauser. Outlier-preserving focus+context visualization in parallel coordinates. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):893–900, Sept 2006.

[77] Steffen Oeltze, Helmut Doleisch, Helwig Hauser, Philipp Muigg, and Bernhard Preim. Interactive visual analysis of perfusion data. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1392–1399, 2007.

[78] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: A review. *SIGKDD Explor. Newsl.*, 6(1):90–105, June 2004.

[79] D. Patel, M. Haidacher, J. P. Balabanian, and E. M. Gröller. Moment curves. In *Visualization Symposium, 2009. PacificVis '09. IEEE Pacific*, pages 201–208, April 2009.

[80] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, Han-Wei Shen, Teng-Yok Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 105–112, 2011.

[81] Tom Peterka, Robert Ross, Boonthanome Nouanesengsy, Teng-Yok Lee, Han-Wei Shen, Wesley Kendall, and Jian Huang. A study of parallel particle tracing for steady-state and time-varying flow fields. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, pages 580–591, 2011.

[82] H. Piringer, R. Kosara, and H. Hauser. Interactive focus+context visualization with linked 2d/3d scatterplots. In *Coordinated and Multiple Views in Exploratory Visualization, 2004. Proceedings. Second International Conference on*, pages 49–60, July 2004.

[83] F. Porikli. Integral histogram: a fast way to extract histograms in cartesian spaces. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 829–836 vol. 1, June 2005.

[84] Luis M. Portela. *Identification and Characterization of Vortices in the Turbulent Boundary Layer. Volume I.* PhD thesis, Stanford University, 1997.

[85] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G.H. Weber. Scalable computation of streamlines on very large datasets. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12, 2009.

[86] C. Rossl and H. Theisel. Streamline embedding for 3d vector field exploration. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):407 –420, 2012.

[87] Oliver Rübel, Prabhat, Kesheng Wu, Hank Childs, Jeremy Meredith, Cameron G. R. Geddes, Estelle Cormier-Michel, Sean Ahern, Gunther H. Weber, Peter Messmer, Hans Hagen, Bernd Hamann, and E. Wes Bethel. High performance multivariate visual data exploration for extremely large data. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 51:1–51:12, Piscataway, NJ, USA, 2008. IEEE Press.

[88] Y. Rubner, C. Tomasi, and L.J. Guibas. A metric for distributions with applications to image databases. In *ICCV '98: Proceedings of the International Conference on Computer Vision*, pages 59 –66, 1998.

[89] G. Scheuermann, T. Bobach, H. Hagen, K. Mahrous, B. Hamann, K.I. Joy, and W. Kollmann. A tetrahedra-based stream surface algorithm. In *Visualization, 2001. VIS '01. Proceedings*, pages 151–553, Oct 2001.

[90] M. Schlemmer, M. Heringer, F. Morr, I. Hotz, M.-H. Bertram, C. Garth, W. Kollmann, B. Hamann, and H. Hagen. Moment invariants for the analysis of 2d flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1743 – 1750, 2007.

[91] David W. Scott. On optimal and data-based histograms. *Biometrika*, 66(3):pp. 605–610, 1979.

[92] Kuangyu Shi, Holger Theisel, Hans-Christian Hege, and Hans-Peter Seidel. Path line attributes - an information visualization approach to analyzing the dynamic behavior of 3d timedependent flow fields. In *Proceedings of Topology-Based Methods in Visualization*, 2007.

[93] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.

[94] K. Stockinger, E.W. Bethel, S. Campbell, E. Dart, and Kesheng Wu. Detecting distributed scans using high-performance query-driven visualization. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 39–39, Nov 2006.

[95] K. Stockinger, J. Shalf, Kesheng Wu, and E.W. Bethel. Query-driven visualization of large data sets. In *Visualization, 2005. VIS 05. IEEE*, pages 167–174, Oct 2005.

[96] T. Stöter, T. Weinkauf, H.-P. Seidel, and H. Theisel. Implicit integral surfaces. In *Proc. Vision, Modeling and Visualization*, pages 127–134, November 2012.

[97] A. Telea and J.J. Van Wijk. Simplified representation of vector fields. In *Vis '99: Proceedings of IEEE Visualization*, pages 35 –507, 1999.

[98] D. Thompson, J.A Levine, J.C. Bennett, P.-T. Bremer, A Gyulassy, V. Pascucci, and P.P. Pebay. Analysis of large-scale scalar data using hixels. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 23–30, Oct 2011.

[99] Fan-Yin Tzeng, Eric B. Lum, and Kwan-Liu Ma. A novel interface for higher-dimensional classification of volume data. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 66–, Washington, DC, USA, 2003. IEEE Computer Society.

[100] Allen Van Gelder. Stream surface generation for fluid flow solutions on curvilinear grids. In *Proceedings of the 3rd Joint Eurographics - IEEE TCVG Conference on Visualization*, pages 95–106, 2001.

[101] J.J. van Wijk. Implicit stream surfaces. In *Visualization, 1993. Visualization '93, Proceedings., IEEE Conference on*, pages 245–252, 1993.

[102] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[103] Y. Wang, W. Chen, J. Zhang, T. Dong, G. Shan, and X. Chi. Efficient volume exploration using the gaussian mixture model. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1560–1573, Nov 2011.

[104] Jishang Wei, Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. A sketch-based interface for classifying and visualizing vector fields. In *PacificVis '10: Proceedings of IEEE Pacific Visualization Symposium*, pages 129 –136, 2010.

[105] D. Whalen and M. L. Norman. Competition data set and description. In *2008 IEEE Visualization Design Contest*, 2008.

[106] Daniel Whalen and Michael L. Norman. Ionization front instabilities in primordial h ii regions. *The Astrophysical Journal*, 673(2):664, 2008.

[107] Daniel Whalen, Brian W. O'Shea, Joseph Smidt, and Michael L Norman. Photoionization of clustered halos by the first stars. *AIP Conf. Proc.*, 990:381–385, 2008.

[108] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. G. R. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Laurent, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rübel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang. FastBit: Interactively Searching Massive Data. *Journal of Physics Conference Series, Proceedings of SciDAC 2009*, 180:012053, June 2009. LBNL-2164E.

[109] Kesheng Wu, W. Koegler, J. Chen, and A. Shoshani. Using bitmap index for interactive exploration of large datasets. In *Scientific and Statistical Database Management, 2003. 15th International Conference on*, pages 65–74, July 2003.

[110] Lijie Xu, Teng-Yok Lee, and Han-Wei Shen. An information-theoretic framework for flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1216–1224, Nov 2010.

[111] H. Yu, C. Wang, C. Shene, and J. Chen. Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1353–1367, 2012.

[112] X. Zhao and A. Kaufman. Multi-dimensional reduction and transfer function design using parallel coordinates. In *Proceedings of the 8th IEEE/EG International Conference on Volume Graphics*, VG'10, pages 69–76, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[113] L. Zhou and C. Hansen. Transfer function design based on user selected samples for intuitive multivariate volume exploration. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pages 73–80, Feb 2013.

[114] L. Zhou and C. Hansen. Guideme: Slice-guided semiautomatic multivariate exploration of volumes. *Computer Graphics Forum*, 33(3):151–160, 2014.