Data Summarization for Large Time-varying Flow Visualization and Analysis

Dissertation

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

By

Chun-Ming Chen, B.S., M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2016

Dissertation Committee:

Dr. Han-Wei Shen, Advisor

Dr. Rephael Wenger

Dr. Jen-Ping Chen

Abstract

The rapid growth of computing power has expedited scientific simulations which can now generate data in unprecedentedly high quality and quantity. However, this advancement has not been mirrored in I/O performance, and hence scientific research is facing great challenges in visualizing and analyzing large-scale simulation results. Among areas of scientific research, fluid flow analysis plays an important role in many disciplines such as aerospace, climate modeling and medicine applications. The data-intensive computation required for fluid flow visualization makes it difficult to devise efficient algorithms and frameworks for flow analysis. First, to analyze a time-varying flow field, pathline visualization is typically used to reveal particle trajectories in the flow. Pathline computation, however, has irregular data access pattern that complicates out-of-core computation when the flow data are too large to fit in the main memory. Strategies on modeling the access pattern and improving spatial and temporal data locality are needed. Second, to avoid tremendous I/O latency, the simulated flow field results are typically down-sampled when they are stored, which inevitably affects the accuracy of the derived pathlines. Error reduction and modeling becomes important to enable uncertainty visualization in order for better decision making.

This dissertation addresses the above challenges by data summarization approaches that efficiently process large data into succinct representations to facilitate flow analysis

and visualization. First, a graph modeling approach is employed to encode the data access pattern of pathline computation, with which a cache-oblivious file layout algorithm and a work scheduling algorithm are proposed to optimize disk caching during out-of-core pathline visualization. Second, an incremental algorithm is devised that fits streaming time series of flow fields into higher-order polynomials and estimates errors in a compact distribution model. The benefit of this distribution-based error modeling is demonstrated to enable probabilistic uncertain pathline computation. Finally, a case study of jet engine stall is conducted for large flow simulations. Vortex analysis and various anomaly detection methods are proposed to capture flow instability that may lead to stall. Comparative visualization techniques are then employed to reveal and contrast temporal patterns from the detection results. Positive expert feedback shows the effectiveness and potential of the proposed methods for stall analysis in large-scale flow simulations.

Acknowledgments

I would like to thank my advisor, Dr. Han-Wei Shen, for his guidance, support and inspiring discussions. He saw my ability of research in my early years of Ph.D. study and challenged me to grow. His trainings on logical thinking and effective presentation have influenced me greatly. I would also like to thank Dr. Jen-Ping Chen, for his patient guidance to the field of computational physics and the thoughtfulness during our long-term collaboration. Special thanks to Dr. Rephael Wenger for serving my dissertation committee and providing insightful comments.

Extra special thanks to Don Stredney, Dr. Thomas Kerwin and Bradley Hittle from the Interface Lab of Ohio Supercomputer Center, Dr. Jonathan Woodring from Los Alamos Laboratories, and Dr. Berk Geveci and Robert Maynard from Kitware Ltd., for joining and mentoring me in relevant projects and broadening my view.

Many thanks to all my research collaborators, including, but not limited, to Teng-Yok Lee, Lijie Xu, Boonthanome Nouanesengsy, Abon Chaudhuri, Ayan Biswas, Tzu-Hsuan Wei, Xiaotong Liu, Xin Tong, Soumya Dutta, Wenbin He, Ko-Chih Wang and Gregory Heinlein. Special thanks to Ju-Yu Huang, for being in part of my PhD life.

Finally, I want to thank my family for supporting me to pursuit Ph.D. without worries back home. My sincere gratitudes to my uncle and aunt, Dr. Roger Hill and Dr. Kueichien Hill, for providing me timely academic and non-academic advices and Thanksgiving meals.

Vita

1982	Born - Taichung, Taiwan
2004	B.S. Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan
2009	M.S. Computer Science, University of Southern California, CA
2009-2011	Graduate Research Associate, Ohio Supercomputer Center
Jun-Aug 2012	Summer Intern, Los Alamos National Laboratory
May-Aug 2014	Summer Intern, Kitware Inc., Albany, NY
May-Aug 2015	Summer Intern, Kitware Inc., Albany, NY
2011-present	Graduate Research Associate, The Ohio State University

Publications

Research Publications

Ayan Biswas, Richard Strelitz, Jonathan Woodring, Chun-Ming Chen, Han-Wei Shen, "A Scalable Streamline Generation Algorithm Via Flux-Based Isocontour Extraction". Eurographics Symposium on Parallel Graphics and Visualization (EGPGV), Groningen, Netherlands, 2016

Wenbin He, Chun-Ming Chen, Xiaotong Liu, Han-Wei Shen, "A Bayesian Approach for Probabilistic Streamline Computation in Uncertain Flows". *IEEE Pacific Visualization Symposium (PacificVis),* pp. 214-218, Taipei, Taiwan, 2016

Chun-Ming Chen, Soumya Dutta, Xiaotong Liu, Gregory Heinlein, Han-Wei Shen and Jen-Ping Chen, "Visualization and Analysis of Rotating Stall for Transonic Jet Engine Simulation". *IEEE Transactions on Visualization and Computer Graphics (TVCG),* vol. 22, no. 1, pp. 847-856, 2016

Tzu-Hsuan Wei, Chun-Ming Chen and Ayan Biswas, "Efficient Local Histogram Searching via Bitmap Indexing". *Computer Graphics Forum,* vol. 34, no. 3, pp. 81-90, Cagliari, Italy, 2015

Xin Tong, John Edwards, Chun-Ming Chen, Han-Wei Shen, Chris Johnson, Pak Wong, "View-Dependent Streamline Deformation and Exploration". *IEEE Transactions on Visualization and Computer Graphics (TVCG),* vol. 2, no. 7, pp. 1788-1801, 2016

Xin Tong, Chun-Ming Chen, Han-Wei Shen and Pak Chung Wong, "Interactive Streamline Exploration and Manipulation Using Deformation". *IEEE Pacific Visualization Symposium (PacificVis)* pp. 1-8, Hangzhou, China, 2015

Ayan Biswas, David Thompson, Wenbin He, Qi Deng, Chun-Ming Chen, Han-Wei Shen, Raghu Machiraju and Anand Rangarajan, "An Uncertainty-Driven Approach to Vortex Analysis Using Oracle Consensus and Spatial Proximity". *IEEE Pacific Visualization Symposium (PacificVis),* pp.223-230, Hangzhou, China, 2015

Chun-Ming Chen, Ayan Biswas and Han-Wei Shen, "Uncertainty Modeling and Error Reduction for Pathline Computation in Time-varying Flow Fields". *IEEE Pacific Visualization Symposium (PacificVis),* pp. 215-222, Hangzhou, China, 2015

Chun-Ming Chen and Han-Wei Shen, "Graph-based Seed Scheduling for Out-of-core FTLE and Pathline Computation". *IEEE Symposium on Large Data Analysis and Visualization (LDAV) 2013,* pp.15-23, Atlanta, GA, 2013 (Honorable Mention)

Chun-Ming Chen, Boonthanome Nouanesengsy, Teng-Yok Lee and Han-Wei Shen, "Flow-guided file layout for out-of-core pathline computation". *IEEE Symposium on Large Data Analysis and Visualization (LDAV) 2012,* pp.109-112, Seattle, WA, 2012 (Honorable Mention)

Chun-Ming Chen, Lijie Xu, Teng-Yok Lee and Han-Wei Shen, "A Flow-Guided File Layout for Out-Of-Core Streamline Computation". *IEEE Pacific Visualization Symposium (PacificVis) 2012,* pp.145-152, Incheon, Korea, 2012

Fields of Study

Major Field: Computer Science and Engineering

Studies in:

Computer Visualization	Prof. H.-W. Shen
High Performance Computing	Prof. P. Sadayappan
Artificial Intelligence	Prof. E. Fosler-Lussier

Table of Contents

List of Tables

List of Figures

Figure **Page**

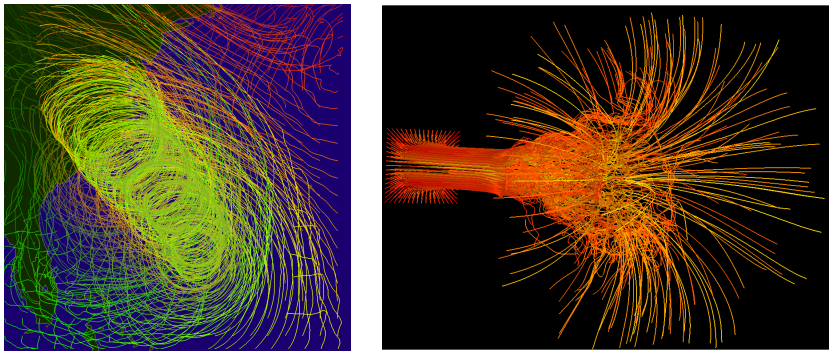
Part I: Background

Chapter 1: Introduction

Scientific research has been benefited by the unprecedented growth of data quality and quantity from simulations, thanks to recent advancements in high performance computer hardware and software. As we expect to embrace exascale supercomputing in the near future [149, 4], computational scientists will be able to conduct more complex simulations in even higher resolutions to reveal even more details with unprecedented numerical precision. However, the fundamental problem of the increasing discrepancy between the leap in computation speed and the slow growth in I/O speed, still remains unsolved [5, 108, 171]. Consequently, as the amount of data continues to grow bigger, domain scientists are in an urgent need of large-scale data analysis algorithms as well as data representations that can overcome the I/O bottleneck.

Visualization has played a crucial role in data analysis, because it allows interaction of data by human for both information exploration and decision making. Visualization can provide effective communication of insights to data that are nontrivial to describe in a few numbers[1]. With the recent improvements of hardware and software in computer graphics and parallel computing, computer visualization has become more powerful and tangible. However, the rapid growth of data size and complexity still outpaces the growth of computer graphics capacity, which makes rendering the entire dataset on screen infeasible.

[1] See Anscombe's quartet for a classic example [8, 38].



(a) (b)

Figure 1.1: Pathline visualization for time-varying flow fields. (a): Flow simulation of Hurricane *Isabel* on the Atlantic ocean. (b): Solar plume on the surface of the Sun.

On one hand, long I/O latency for loading large data from second-level storage hampers interactive rendering; on the other hand, displaying large amount of data without curation can simply exceed what a human can digest. Therefore, visualization often requires a preprocessing step that extracts and summarizes data into a compact representation which preserves important features for the convenience of later analysis.

Flow visualization as a branch of data visualization plays an important role in many scientific disciplines such as automotive, aerospace, climate modeling and medicine applications [119, 120, 172]. A flow field is a field of vector data representing the fluid velocities of the corresponding spatial positions at a given time. A time-varying flow field is a series of flow fields in a period of time. Time-varying flow fields can be generated by CFD (Computational Fluid Dynamics) simulations as a numerical solution of governing equations, or collected from physical experiments [120]. To visualize a flow field, particle tracing is commonly used to reveal the trajectories of particles moving along the vector directions in the flow field. For example, visualization of particle traces allows the scientists to identify and verify the existence of vortices [75]. In time-varying flow fields, particle

tracing generates pathlines (as shown in Figure 1.1), which can then derive other useful visualizations including streaklines, time lines [76], path surfaces as well as FTLE (Finite Time Lyapunov Exponent), a measure of flow separation in flow topology analysis (See Chapter 4). In addition, scalar fields such as density and pressure measures are commonly generated or derived alongside flow fields in CFD simulations, which are also important in understanding and exploring the behavior of the flow.

While "large data" can have different definitions, the National Research Council has recently categorized large data problems into the following five settings according to the computational constraints on memory [109]:

1. Default setting: The data can be stored in random access memory (RAM). This is the setting where most traditional data analysis algorithms are designed with.

2. Streaming setting: The data are streamed into the processor and only a subset can be stored.

3. Disk-based setting: The data can be stored in a machine's disk but are too large to store in RAM.

4. Distributed setting: The data are distributed over multiple machines.

5. Multi-threaded setting: The data are shared by multiple processors in the same RAM of a machine.

In this dissertation, we address non-default settings for large-scale flow visualization, mainly focusing on the disk-based setting and the streaming setting. Below we identify challenges in different stages of flow analysis pipeline from simulation machines towards a user's commodity machine:

- On a user's commodity machine, usually a local desktop, visualizing time-varying flow fields in large scales falls into the category of disk-based setting, where the entire dataset cannot fit into the main memory. In such a case, an *out-of-core* computation scheme is usually applied, where partial data are loaded from the disk into the memory on demand during the computation. Since disk I/O latency is the major bottleneck during out-of-core computation, increasing data locality while reducing the number of disk accesses is essential to increase the overall computation performance. In order to speed up out-of-core pathline visualization, an I/O-friendly pathline computation algorithm and file organization is required, which is nontrivial due to the irregular data access pattern in particle tracing.

- On simulation machines, usually a computer cluster or a supercomputer, although high-resolution datasets can be generated from simulations, it is prohibitive to store the entire large data for post-hoc analysis. A common practice is to store the data in a coarse-grained temporal resolution, where the flow fields are subsampled in the temporal dimension [4, 171]. However, this approach inevitably produces errors when values are interpolated for the skipped time steps in post-hoc analyses. An error modeling method during data reduction is required to enable uncertainty visualization for the lost data in post-analysis. This is especially important for particle tracing in time-varying flow fields since errors can accumulate quickly during the integration steps.

- When a dataset is large, automatic feature extraction becomes important to save manual exploration time. To efficiently extract temporal flow features from large time

series is nontrivial, especially when the data are iteratively streamed into the analysis processors. Moreover, when the feature detection criteria are not well defined, a flexible feature extraction with adjustable visualization parameters is necessary.

To tackle the above challenges, we exploit data summarization approaches that characterize flow data into compact representations while preserving important features. Two representation types, graph-based and statistical summaries, are employed to model essential behaviors in local spatiotemporal regions of a time-varying dataset. These summaries are demonstrated to allow efficient probabilistic flow modeling and uncertainty analysis. Below is an overview of the methods employed in this dissertation.

1.1 Graph-based Flow Modeling for Out-of-core Pathline Visualization

To derive an algorithm for efficient out-of-core computation, it is important to obtain the run-time data access pattern. Although in most applications, the data access pattern can only be obtained at run time [78], for particle tracing, the access pattern instead can be extracted directly from the input flow field, since particles follow the flow directions encoded in the flow field. We model the access pattern of a flow field during particle tracing by a weighted directed graph, where each node represents a spatial block in the flow field domain, and each edge connects spatially and temporally nearby blocks. The weight of each edge models the probability of a randomly positioned particle moving from one block to the other. This flow graph model is shown effective to derive an I/O-effective file layout and seed scheduling, briefly described below.

1.1.1 Graph-based File Layout for Efficient Flow Field Prefetch

Since disk I/O prefers sequential accesses, improving data locality can increase the efficiency of out-of-core computations [71]. By reordering the file layout in a way that conforms to the data access pattern, the overall I/O latency can be reduced through prefetching of data nearby the requested data. We present a file layout algorithm for out-of-core particle tracing by utilizing a flow graph modeling approach which summarizes block-wise data access patterns. A *cache-oblivious* file layout algorithm is developed to produce a sequential ordering of data blocks favoring prefetching with arbitrary prefetch sizes. We will show in Section 3 that using the proposed graph-based file layout, out-of-core particle tracing performs more efficiently compared to using a general space-filling curve layout.

1.1.2 Graph-based Seed Scheduling

Seeds are initial locations of particles to trace in a flow field. When the number of particles to trace is large, there will not be enough memory to store all the temporary particle locations during particle tracing. This problem is typically seen in deriving high-resolution FTLE (Finite-Time Lyapunov Exponent) fields for large flow fields. Therefore, it is necessary to perform the batch mode computation, where particles are organized into different groups and only one group of particles are advected at a time. To avoid reloading the same data blocks used in tracing different groups of particles, the seeds should be grouped in a way that the blocks on the particle paths of a group have minimal overlaps with other groups. Since the above flow graph models the particle movements among blocks in probabilities, in Section 4 we present an algorithm that uses the graph model to predict block-wise particle paths using Markov chains. With the path prediction, we are able to compute seed grouping with minimal block overlaps along the paths of different groups.

A hierarchical clustering method is developed to group the seeds and further schedule the computation order of each group to maximize cached data usage. Results show that using the graph-based seed scheduling method can achieve better I/O performance in out-of-core particle tracing given a large number of seeds.

1.2 Distribution-based Uncertainty Modeling for Time-varying Flow Fields

Unsteady flow simulations can generate time-varying flow fields and other scalar fields commonly in tens or hundreds of thousands of time steps. Storing every time step of the simulation output is very costly due to high simulation-time I/O latency and storage overhead. Therefore, simulation output is often sampled and stored only at certain time steps to reduce the amount of data storage. This inevitably causes information loss, and pathlines integrating subsampled vector fields can become even erroneous.

Uncertainty visualization of noisy or missing data is crucial in many applications including geography [106], astrophysics [95] and climate analysis [66]. Traditional error measures including error bounds, peak signal-to-noise ratio (PSNR) or root-mean-square error (RMSE) have been used to quantify uncertainty in scientific data [18, 105]. Instead of using simple error measures, the use of distributions (e.g., histograms or parametric distribution models) has recently been advocated to model errors from noisy measurement [70, 26]. Distributions encode of the occurrence frequencies of data values, which allow probabilistic computation of uncertainty propagation for further analysis and visualization.

While most existing uncertainty modeling methods are applied to datasets of a single time step, few works focus on uncertainty modeling of temporally down-sampled data in the streaming setting. As a simulation generates temporal data iteratively, incremental

modeling of uncertainty due to data down-sampling should be employed, which processes data sequentially with small memory footprint. In Chapter 5, we present an incremental algorithm to model temporal interpolation error for time-varying flow fields. We will show that with distribution-based error modeling for each time interval of the flow fields, probabilistic pathline computation can be used to estimate the distribution of the particle location for each time step.

1.3 Feature Extraction and Summary Visualization for Flow Stability Analysis

Generally speaking, the goal of data analysis is to answer questions in problem solving, hypothesis verification, or decision making. However, the questions are not always well defined by the user for automatic data analysis. To tackle this, *visual analytics* that closely couples human and automatic analysis by interactive criterion refinement becomes attractive in such a case [166]. Meanwhile, analyzing thousands of time steps of flow simulation outputs is prohibitively time consuming for the user to inspect the data one time step after another. As a result, it is necessary for visual analytics systems to extract possibly interesting features and summarize them into a manageable size for efficient visualization. The user can then locate interesting subsets of the data for detailed analysis and exploration. In Chapter 6, we introduce a visual analytic system for *rotating stall* analysis in large-scale flow simulation of jet engine compressors.

Rotating stall is a temporal phenomenon that starts from subtle air disturbance but can quickly grow and interfere normal engine operation. Early detection of rotating stall is still an ongoing research on jet engine compressor stability. A high-resolution, full-annulous CFD solver has been made in NASA [33], which generates more than a terabyte of data

in a typical simulation run. Since the formulation of rotating stall is still under active research, criteria for detecting its precursors are not well defined. Therefore, a visual analytics framework is designed to extract and visualize suspected regions with potential to stall. We employ general statistical and signal processing techniques to detect anomalies in data and summarize detection results for efficient visualization. From the temporal patterns revealed from the visualization, the domain scientists are able to identify interesting time steps and regions for further detailed study.

Thesis Organization. The rest of this thesis is organized as follows: Chapter 2 provides background of particle tracing and related works on large data visualization. The next three parts cover the approaches outlined in the above three sections. Part II presents the strategies for efficient out-of-core pathline computation, in which Chapter 3 introduces the graph-based file layout method and Chapter 4 elaborates the graph-based seed scheduling method. Part III presents the distribution-based uncertainty modeling for large time-varying flow fields. Part IV studies feature extraction and summary visualization techniques specific for rotating stall analysis in large turbine flow simulation. Chapter 6 discusses the distribution-based feature extraction and summary visualization methods, and Chapter 7 extends the work for identifying and visualizing rotating stall inception with temporal and region-based anomaly detection. Finally, Part V concludes the works and outlines future research.

Chapter 2: Background and Related Work

2.1 Flow Field and Particle Tracing

A flow field typically represents the wind or fluid flow composed of vectors with two or three scalar components to describe the velocities in 2D or 3D. Formally, a 3D flow field is defined as $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, where $f(\mathbf{x})$ defines the velocity at a point $\mathbf{x}$ in $\mathbb{R}^3$. A 3D time-varying flow field is defined as $f : \mathbb{R}^3 \times \mathbb{R} \rightarrow \mathbb{R}^3$, where $f(\mathbf{x},t)$ defines the velocity at a point $\mathbf{x}$ in $\mathbb{R}^3$ at time t.

To analyze a vector field, particle tracing plays an important role in obtaining the trajectories of massless particles moving in the flow. A *streamline* is the trace of a massless particle moving along with the vector directions in a static flow field; in a time-varying flow field, the trajectory of a particle moving along with the flow direction over time is called a *pathline*. Given either a static or time-varying flow field, particle tracing is the building blocks of related flow visualization techniques, such as stream surfaces, path surfaces, streaklines and time lines. Many applications of particle tracing have been shown useful in flow analysis and visualization, including:

- Direct visualization of streamlines or stream surfaces around a vortex helps understand their structures [51].

- Pathline tracing is the essential part of FTLE computation. FTLE (Finite-Time Lyapunov Exponent) measures have been used to extract the Lagrangian Coherent Structure of the flow field, which describes the separation or contraction of the flow [61]. Detailed definition of FTLE will be described in Section 4.1.

Given an initial particle position $\mathbf{x}_0$ and time t_0, the pathline $\mathbf{x}(t)$ is defined by:

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.1}$$

$$\frac{d}{dt}\mathbf{x}(t) = f(\mathbf{x}(t),t) \tag{2.2}$$

Essentially, the tangent of a pathline at any given point is equivalent to the flow vector direction at the corresponding time and that point. In numerical methods, a simple approximation approach to solve the above differential equation is Euler's method, by approximating the particle offset $\triangle\mathbf{x}$ over a time interval h:

$$\triangle\mathbf{x} = \mathbf{x}(t+h) - \mathbf{x}(t) = f(\mathbf{x}(t),t)\cdot h \tag{2.3}$$

To obtain results with a higher accuracy, the Runge-Kutta method derived from high-order Taylor expansion are usually used. The fourth-order Runge-Kutta method with an error of $O(h^5)$ per step is listed below:

$$\begin{aligned}
k_1 &= hf(\mathbf{x}(t),t) \\
k_2 &= hf\left(\mathbf{x}(t)+\frac{k_1}{2},t+\frac{h}{2}\right) \\
k_3 &= hf\left(\mathbf{x}(t)+\frac{k_2}{2},t+\frac{h}{2}\right) \\
k_4 &= hf(\mathbf{x}(t)+k_3,t+h) \\
\mathbf{x}(t+h) &= \mathbf{x}(t)+\frac{1}{6}k_1+\frac{1}{3}k_2+\frac{1}{3}k_3+\frac{1}{6}k_4
\end{aligned}$$

In either Euler's method or the Runge-Kutta method, the integration is computed iteratively from the initial particle location. This iterative procedure of particle tracing makes efficient computation challenging in terms of parallelization and scalability.

2.2 Out-of-core Particle Tracing

When the flow field data size is larger than the main memory capacity and can only be stored in the second-level memory, e.g., hard disks, out-of-core computation is required. In out-of-core computation, partial data are loaded into the main memory on demand and released if not used. The performance of out-of-core computation is mainly affected by the I/O latency at runtime, where different loading and releasing strategies can result in distinct overall I/O latency.

A lot of research has been done on out-of-core computation, where increasing data locality is one common approach to reduce the number of I/O accesses and increase overall performance [12]. Silva et al. [147] provided a review of out-of-core algorithms for general scientific visualization problems. Cox and Ellsworth [39] advocated to store data per spatial block to improve data locality and implement application controlled demand paging to obtain better I/O performance.

For out-of-core streamline computation, Ueng et al. [160] presented a technique to compute streamlines in large unstructured grids with a queuing system. Pugmire *et al.* [127] used a hybrid of static block allocation and load-on-demand to achieve load balance and minimal I/O. Sulatycke and Ghose [150] proposed a multi-threaded system for out-of-core isosurface rendering with a layout that reduced disk seeks. To visualize particle traces in real-time, Bruckschen et al. [20] compute and store pathlines in the preprocessing stage and load the pre-computed particle traces according to the input seed locations. Camp et

al. [24] cached flow data in solid state drives to reduce overhead of repeated I/O accesses of the same data block. In a distributed system, out-of-core computation is also used to reduce memory footprint and I/O latency of particle tracing in both parallelize-over-block [111] and parallelize-over-seed scenarios [25].

2.3 Data Reduction with Uncertainty Modeling

To reduce the size of large data, compression is an obvious approach, where many techniques have been studied and developed for decades. Recent works related to scientific visualization mainly focus on compressing floating-point data values. Lakshminarasimhan et al. [90] presented ISABELA, a floating-point data compression method by fitting sorted values by a spline and compressing the indexes. Kim et al. [86] adapted a bitmap indexing scheme to support efficient in situ data queries. Lindstrom [96] compressed small spatial blocks of floating-points in user-specified bit rates. Wang et al. [163] focused on determining the temporal block ranges according to user interests for compressing large-scale time-varying data.

Although advanced compression techniques have been developed to achieve higher compression rates, a great amount of information will still be lost when the data size becomes extremely large and only a small portion can be stored. Therefore, it becomes important to model the lost data with uncertainty measurements. Traditional approaches of uncertainty modeling are to describe the error measures in error bounds, peak signal-to-noise ratio (PSNR) or root mean square error (RMSE) [18, 105]. Instead of using simple error measures, the use of distributions (e.g., histograms or parametric distribution models) has lately been advocated to model errors from noisy measurement [26], especially by the International Organization for Standardization (ISO) [70].

For data reduction with the support of uncertainty visualization, Thompson et al. [155] presented the idea of Hixel, which stores a histogram per data block to preserve statistical information due to data reduction. They showed the advantages of storing hixels including approximating the topological structures and extracting fuzzy isosurfaces, which provide more informative results. Liu et al. [98] further compacted the distribution representation by Gaussian mixture models (GMMs) in order for stochastic volume rendering on the GPU.

2.4 Distribution-based Uncertainty Visualization Techniques

Distribution-based uncertainty visualization has drawn much attention in recent years and many approaches have been proposed. Brodie et al. [18] and Bonneau et al. [17] provided thorough reviews. Earlier approaches to visualize a 2D distribution dataset where the data value of each 2D grid point is described by a probability distribution include displaying statistical summaries such as means, standard deviations and skews in color, height field, or glyphs [79, 103, 125]. Potter et al. [124] proposed summary plots which extend box plots with moments and histograms in higher dimension visualizations. To visualize 3D distribution datasets, flickering the color according to the distribution samples is used for uncertain volume rendering [98, 155]. Uncertain isosurface extraction provides further understanding of data at a specific isovalue. Pthkow et al. [121] computed the level crossing probability of adjacent points, which was extended to computing cell-wise level crossing probability [123] and nonparametric modeling [122]. Athawale et al. [9] devised closed-form computation of level-crossing probabilities for nonparametric distribution datasets.

2.5 *In Situ* Data Analysis

As the data size continues to grow, transferring data from the simulation machines to visualization machines is getting difficult. As a result, *in situ* data analysis has become attractive in recent years [4, 13, 36, 44, 105, 131]. In general, *in situ* analysis is to perform pre-configured analysis tasks during the simulation, so that only the processed data in a much smaller size is stored. The major benefit is that, since the simulation data are processed while they are still in the memory of simulation machines, this will avoid the enormous I/O overhead to store the entire raw data during the simulation and to load them again in post-processing.

Successful use cases of *in situ* analysis include visualization [44, 171], compression [92, 156], feature extraction [167] and indexing [86]. Peterka et al. [117] presented *in situ* parallel particle tracing methods. Several software frameworks for *in situ* analysis has been proposed. Fabian et al. presented CATALYST [47], which integrates *in situ* visualization with the popular visualization tool, ParaView [10]. Ahrens proposed Cinama [5], which enables image-based rendering for post-hoc analysis.

The *in situ* computation matches the streaming setting of large data analysis as described in Chapter 1, where simulation data are streamed into the processing units for analysis and visualization. Dorier et al. [44] summarized several strategies of where and how the analysis tasks are performed during the simulation. In brief, the analysis tasks can run in the same nodes of the simulation machines, or on separate analysis nodes where data are transferred through high-speed network. All of the strategies avoid storing the entire raw data to the disks. However, the *in situ* algorithms should be designed to process the simulation data as quickly as possible, in order not to jam the normal simulation runs. Global data

access across a wide range of time steps is also difficult due to limited memory capacity to hold too many time steps of raw data.

In this dissertation, the data pre-processing stages are designed to be efficient suitable for *in situ* analysis. For example, in Chapter 5, we present an incremental algorithm to estimate temporal interpolation errors due to data down-sampling, which iteratively processes data per time step and keeps small memory footprint. In Chapter 6 and 7, the data analysis procedures are designed to be efficient and only involve data accesses within limited time intervals.

Part II: Graph-based Flow Modeling

Chapter 3: Flow-guided File Layout for Out-of-core Pathline Computation

The rapid growth of data size in recent years has made scientific visualization more challenging. Although very large-scale data analysis and visualization tend to be performed on supercomputers, scientists still prefer to analyze data on their local machines whenever possible. With more powerful multi-core CPUs and increasing storage capacities, a higher-end desktop computer can now handle much larger data sizes than before. Desktops, though, usually have insufficient memory capacity to hold the entire data at once. Therefore, it is necessary to perform visualization of large data in an out-of-core manner. Out-of-core computation suffers from unavoidable slow disk I/O in terms of low bandwidth and high latency, thus reducing the I/O overhead still remains an essential issue.

A well-designed I/O strategy can reduce I/O latency and thus increase the performance of out-of-core computation. However, a good I/O strategy usually requires the knowledge of runtime data access pattern. For particle tracing, we observed that its runtime access pattern can be predicted from the vector directions of the input flow field itself. Therefore, we introduce a directed graph model, called the *access dependency graph,* or ADG, to model the access pattern of the flow field. A preprocessing of the flow field is done to generate the ADG, where the graph nodes represent spatial data blocks, and an edge connects two blocks if there exist particles traveling between them.

In this chapter, we present a file layout method for effective prefetching in out-of-core particle tracing. Data prefetching has been extensively applied to bridge the widening gap between disk access speed and processor speed. To maximize the benefit of prefetching, data blocks should be placed in disk in an order conforming to the application's data access pattern. For pathline computation, to ensure that the data blocks ahead of the current particle position will be readily available, one can organize the data blocks in disk according to the flow directions, and then prefetch contiguous chunks of data at run time whenever possible. To facilitate efficient out-of-core pathline computation, we present a file layout algorithm for time-varying flow fields. The goal of the file layout is to order the data blocks in a way that the I/O cost is minimized and the utilization of prefetch data is maximized for pathlines seeded at arbitrary locations in the flow field.

As recomputing the file layout for different hardware configuration is highly undesired, we want the generated file layout to always incur smaller I/O latency during pathline computation even for machines of different memory capacities, a common goal shared by most of the cache-oblivious mesh layout algorithms [16, 81, 170, 169]. Compared to these algorithms, which mainly consider undirected connectivities among adjacent mesh cells, the proposed layout algorithm is designed for directed graphs by taking flow directions into account, leading to a higher utilization of the prefetched blocks for efficient pathline computation.

The rest of this chapter is organized as follows. Section 3.1 reviews previous works of file layout methods, followed by the overview of our system in Section 3.2. The graph modeling is detailed in Section 3.3 and the layout optimization algorithm is presented in

Section 3.4. Section 3.5 describes the implementation details for out-of-core pathline computation. In Section 3.6, we discuss the experimental results and compare our method with other conventional layout methods. The chapter is concluded in Section 3.7.

3.1 Related Work

Re-arranging file layouts to improve I/O or cache performance has received much research attention in past two decades. Pascucci and Frank [116] implemented a global static indexing scheme using Z-curves for real time slicing of very large scaled regular grids volumetric data. Niedermeier and Sanders [110] used Hilbert curve [56] to create efficient indexing for mesh-connected computers for parallel computing . Chiang [35] used a cache-oblivious time tree for isosurface extraction of irregular time-varying data. Isenburg and Lindstrom [69] proposed an efficient streaming mesh format that is suitable for effective disk I/O for out-of-core applications. Research has been conducted to improve cache efficiency by reorganizing large-scaled mesh data [16, 81, 170, 169]. In their methods, the mesh is formulated as a graph and the data reorganizing problem is reduced to the graph arrangement problems. Among the above works, Yoon and Lindstrom [169] proposed a novel cache-oblivious cost function using the geometric mean of possible cache sizes, which can generate better layouts than using arithmetic means. Tchiboukdjian et al. [153] proposed a layout algorithm for unstructured meshes with a theoretic complexity analysis that can be represented by overlapping graphs.

Graph modeling of large-scale flow fields has been applied to provide compact representation of the flow behavior. Chen et al. [31] used the Morse Connection Graphs (MCG) model to represent the topological structure of the flow fields. The MCG model facilitates flow field analysis such as simplification, periodic orbits generation and extraction.

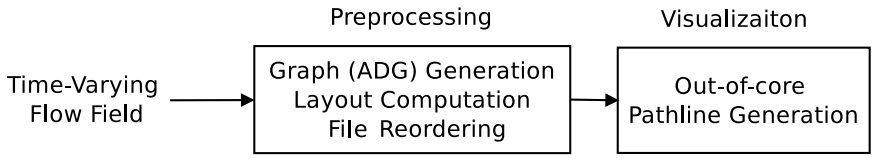


Figure 3.1: System Overview.

Xu and Shen [168] proposed a different node link graph model used mainly as user interface. A similar model was applied for load-balanced parallel computation of streamlines by Nouanesengsy et al. [112]. Based on the graph model and the initial seeds, the computation workload needed for each block can be estimated, which is then used to optimize data partitioning and balance workload. Reich and Scheuermann [129] approximated FTLE at infinity by applying Markov chains on static unstructured flow fields encoded into a graph representation. Ma et al. [104] proposed a hierarchical graph-based modeling and visualization method for flow field exploration and query. Chen and Fujishiro [34] optimized the domain partitioning of unstructured flow fields for parallel streamline computation. The flow field is modeled as a graph based on an anisotropic local diffusion operator and partitioned with a multilevel spectral graph bisection method that minimizes the sum of edge weights among the partitions.

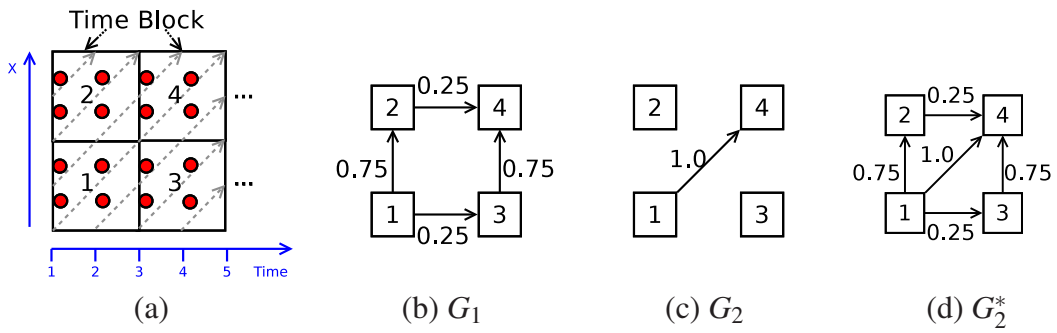
3.2 System Overview

The goal of our system is to optimize and reorder the file layout of a time-varying flow field to benefit later out-of-core pathline computations. Figure 3.1 illustrates the components in our system. Because time-varying flow fields generated from large scale simulations are often too large to fit into a desktop's main memory, a preprocessing stage is necessary to decompose the data into smaller blocks to reduce the working set size as well

as improve the spatial locality of the data. To minimize the I/O overhead, our out-of-core system loads data blocks from disk only when they are needed. Clearly, when performing pathline computation, the only block that is absolutely required is the block that contains the particle's current position. To allow the particle to move forward for a longer distance without having to frequently stop and request more data, we can load more data blocks at the down stream of the flow from the current particle position, i.e., to prefetch data that will soon be needed. Since sequential access of a disk is more efficient than random access, the blocks that will be prefetched should be placed in a close proximity in the file. In the preprocessing stage, our system automatically determines the layout of the data blocks in a file to optimize the efficiency of data prefetching when computing pathlines in the visualization stage. Since the run-time data access pattern is guided by the flow field directions, we model the access dependencies among the data blocks of a flow field as a graph, where each node represents a spatial data block, and each edge represents the probability for the particles to travel from one block to the other. We then linearize the graph nodes and use the order to place the corresponding data blocks in a file. In the following, we explain our algorithm used in the preprocessing stage in more detail.

3.3 Graph-based Data Access Pattern Modeling

To optimize the file layout in the preprocessing stage, the first step is to analyze the data access pattern of particle tracing. Since preprocessing can take significant time, this stage will be done only once for each flow dataset. Therefore, the layout algorithm is designed to accommodate various disk parameters and arbitrary seeding locations, in order for the reordered file to be reused in most cases for different machines.



(a) (b) G_1 (c) G_2 (d) G_2^*

Figure 3.2: The construction of the Access Dependency Graph (ADG). (a): An illustration of a one-dimensional time-varying flow field with five timesteps and four time blocks. Time blocks 1 and 2 contain data in timesteps 1 to 3. Time blocks 3 and 4 contain data in timesteps 3 to 5. The grey dotted arrows show the underlying flow direction. Each time block has four sampling seeds. (b): The ADG of one hop constructed from (a). (c): The ADG of two hops. (d): The union of (b) and (c) for layout optimization.

Since pathline computation requires data from at least two time steps at each integration step, to enhance data locality during pathline computation, the input time-varying flow field is first divided into equal-sized spatial blocks for each time step, and then the blocks in the same spatial location but in consecutive time steps are merged together to form a *time block*. A time block is a unit data block in our out-of-core pathline computation system.

3.3.1 The Access Dependency Graph (ADG)

The goal of file layout design is to provide an ordering of the data blocks which benefits prefetching and hence increases I/O efficiency at runtime. In other words, the layout order should follow the runtime access pattern during the pathline computation. We model the runtime access pattern of pathline computation as a directed graph called *access dependency graph*, or ADG. In the ADG, each graph node represents a time block of the dynamic flow field. A directed graph edge connects two time blocks if there exist pathlines passing through them. Each edge is assigned a weight equal to the conditional probability of seeds

moving from one block to the other. The construction of the ADG is shown in Figure 3.2a. Sampling seeds are densely placed in each time block, and are then traced throughout the flow field. The edge weights are calculated by finding the percentage of particles which, starting from each time block, travels to its immediate neighboring block. Here a neighboring block can be the time block in the same spatial location but with different extents in the time dimension, as a particle moving in time can hit a boundary in either the spatial or temporal domain. The constructed ADG now encapsulates the dependencies of all immediately connected time blocks, as illustrated in Figure 3.2b.

3.3.2 Multi-hop ADG

An ADG that represents the connectivities among neighboring blocks is not accurate enough because it fails to capture the potentially higher dependencies between blocks which are more than one block away. The term *hop* is defined as the number of blocks a particle has traveled through from its initial location, excluding the block where the particle begins. The previously discussed ADG is defined as a 1-hop ADG. An N-hop ADG is constructed by allowing sampling particles to advect through N blocks from their starting block. After all the sampling particles have traveled through N blocks, the number of particles in each ending block is counted and normalized as the edge weight. For simplicity, an N-hop ADG is denoted by G_N. The G_2 graph is shown in Figure 3.2c, where all the four sampling particles in time block 1 travel to time block 4. As can be seen, edge $\langle 1,4 \rangle$ in G_2 has a higher weight than any edge in G_1, which cannot be predicted from G_1. Since the goal of the layout is to place highly dependent blocks together, we cannot ignore the potentially higher edge weights in the multi-hop ADG. We use the union of graphs G_1 to G_N, denoted by G_N^* as the final ADG.

3.4 The File Layout Optimization Algorithm

It is known that the disk I/O is more efficient with sequential access, since the positions of successive I/O accesses present high locality. Therefore, the I/O performance of out-of-core pathline computation can be increased if the flow field file is organized in a way that conforms the data access order. Since an ADG encoding particle movements among blocks provides us the necessary information of the data access pattern during particle tracing, we can use it to estimate the I/O cost for generating a file layout. In other words, the goal to reorder the data blocks of a flow field is to place consecutive blocks along the same particle trace in a close proximity in a file such that they can be prefetched together from disk to main memory with a minimum I/O overhead, i.e, fewer disk reads. It also allows us to increase the utilization of the prefetched blocks, or equivalently, to reduce the miss rates when the required data blocks are not in the prefetched block pool.

The following subsections first describe how to estimate the I/O cost for a layout L, which is then used to derive the layout algorithm. We define a layout L is a one-to-one mapping from a node n in ADG to a location $L(n)$ in the file. For example, $L(4) = 2$ indicates that node 4 is placed in block order 2 of the file from the beginning. Below we first introduce the cost function for a one-hop graph and then extend the cost function to multi-hop graphs.

3.4.1 Cost Function for a Layout Modeled by Single-hop Graphs

To reduce the I/O overhead for out-of-core streamline computation, when a block b is requested to be loaded, the proposed out-of-core system also prefetches several additional blocks immediately following the requested block in the file. Note that loading several consecutive blocks sequentially generally takes less time than loading the same number

of blocks randomly, or in a reverse order. Depending on how the data blocks are laid out in a file, the block that will be needed by the current particle right after block b will be successfully prefetched if the block is placed in the file at a distance within the prefetched data size from b and along the forward direction. Otherwise, if the next required block on the particle trace is placed in the back of block b or in a distance greater than the prefetch block size, another I/O access will be issued when the particle moves to block b. Hence an additional memory cache (the data pool) miss occurs.

According to the prefetching scenario described above, we can estimate the miss rate of data blocks after prefetching, by examining the distance of every possible pair of blocks in the file that are to be accessed contiguously during particle tracing. Formally, given a layout L and the one-hop access dependency graph G_1, the cost of the layout is defined by the *Estimated Miss Ratio* (EMR), as shown in Equation 3.1. In essence, it is the averaged prefetch miss rate for all pairs of directly connected graph nodes.

$$EMR(L,G_1) = \frac{1}{|V|} \sum_{(u,v)\in G_1} Pr_1(v|u) \times PMRF(L(v) - L(u)) \tag{3.1}$$

Here $|V|$ is the number of nodes in G_1. $Pr_1(v|u)$ is the conditional probability of node v to be directly accessed from node u, which is already estimated as the weight of edge (u,v) in G_1.

The term *Prefetch Miss Ratio Function (PMRF)* is a probability function that returns the probability of a miss given the layout distance between two blocks to be accessed sequentially. If we know the exact prefetch size, say B blocks, we can define $PMRF^B$ as in Equation 3.2, which is essentially by checking whether the distance between two adjacent blocks in the file is within the prefetch size B:

$$PMRF^{B}(d) = \begin{cases} 0 & : 0 \leq d < B \\ 1 & : otherwise \end{cases} \tag{3.2}$$

The function $PMRF(\cdot)$ takes d, the distance in blocks of successive data accesses, as the input. d can be negative, meaning the next block to be accessed is located *behind* the current block in the file.

In practice the value B is a tunable parameter defined by the visualization system. Since it is often not known at the stage when the file layout is generated, we instead apply Equation 3.3 to model PMRF:

$$PMRF^{O}(d) = \begin{cases} \frac{lg(d+1)}{lg|V|} & : d \geq 0 \\ 1 & : d < 0 \end{cases} \tag{3.3}$$

which is a monotonically increasing function for positive offsets. The logarithmic function is similar to the geometric metric for cache-oblivious mesh layout by Yoon and Linderstorm [169]. Essentially, PMRF imposes high costs for reverse accesses and large jump from the former block.

3.4.2 Cost Function for a Layout Modeled by Multi-hop Graphs

The cost function based on one-hop graphs only examines the prefetch miss rate between two sequentially accessed blocks. As described in section 3.3.1, since the longer access behavior is not well-modeled by the one-hop graph, the evaluated cost will not reflect the real I/O cost in longer hops. To model the I/O cost more precisely, multi-hop graphs $G_1, G_2, \ldots G_N$ is necessary. In essence, our goal is to model the conditional probabilities of all blocks to be accessed in the next $1 \cdots N$ hops, so that the cost of a given layout can be estimated under prefetching more precisely. The prefetch miss ratio within N hops is estimated by averaging each layout cost based on an individual N-hop graph :

$$EMR(L,\{G_1,\ldots,G_N\}) = \frac{1}{N}\frac{1}{|V|}\sum_{i=1}^{N}\sum_{(u,v)\in G_i} Pr_i(v|u)\times PMRF(L(v)-L(u)) \tag{3.4}$$

Here $Pr_k(v|u)$ is the conditional probability of node v to be accessed from node u in k hops, which is equal to the weight of edge (u,v) in G_k.

The multi-hop layout cost function requires the number of hops N as one of the input parameters. In practice, as N grows, the particle will scatter into more blocks, and hence each individual edge weight in G_N will become smaller. This results in smaller total weights in the accumulated graph. In fact, we have observed that N does not need to be large in order to get a satisfactory prefetching result. More details about the choice of N will be discussed in Section 3.6.

3.4.3 The Layout Optimization Algorithm

The goal of the layout optimization algorithm is to search for a layout assignment $L(\cdot)$ that gives lowest cost due to memory cache misses in Equation 3.4. From the cost equation, although the calculation of the cost involves N graphs $G_1, G_2, ..., G_N$, these graphs can be merged into a single graph and the cost function from Equation 3.5 can be rewritten as below:

$$EMR(L,G_N^*) = \frac{1}{N}\frac{1}{|V|}\sum_{(u,v)\in G_N^*} w(u,v)\times PMRF(L(v)-L(u)), \tag{3.5}$$

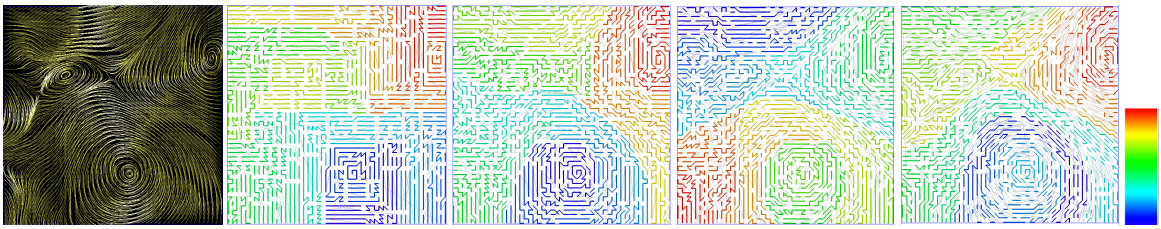
where $G_N^* = \{G_1, G_2, ..., G_N\}$ is the union of graphs $G_1, G_2, ..., G_N$, with summed edge weights from the individual graphs. An example of G_N^* is shown in Figure 3.2d where $N = 2$.

Next, we find the optimal layout that minimizes the miss ratio in Equation 3.5. This optimization problem is similar to the family of graph linear assignment problems, which are

NP-hard [93]. To find an approximated solution, the layout algorithm used in [16, 81, 169] can be adopted. In there method, given an undirected graph, the algorithm recursively reduce the problem size by cutting the input graph into two partitions of similar numbers of nodes, where the sum of the edge weights on the cut is closely minimal. Since the cut edge weights are small, the flow between the two partitions will be small, and thus the subgraph in each partition can be tackled individually. When the size of the partition becomes small enough, i.e., less than a threshold t, the optimal layout for the nodes of the partition is solved via exhaustive search. After the layout for each subgraph is solved, these sub-layouts are concatenated in the bottom-up manner to generate the final layout.

The algorithm above was primarily designed for undirected graphs, while the ADG used in our problem is a directed graph. To accommodate this, before the ADG is to be cut, the graph is first converted into an undirected graph. To perform the conversion, the weight for each edge in the undirected graph is assigned by summing the weights in the corresponding edges of the original graph with same pair of nodes. Consequently, if the flow between a pair of two nodes is small, their edge weight in the converted graphs will also be small. Our implementation sets the threshold t as 4, which yields 4! permutations to find the lowest cost of Equation 3.5 in the bottom of the recursive hierarchy.

To further enforce the final layout to order the blocks in the flow direction in the directed graph, the bottom-up concatenation of the resulting sub-layouts performs an additional optimization. Since there are two possible orders to concatenate the two sub-layouts, the order with smaller cost of Equation 3.5 is chosen. This bottom-up reordering is performed till the root level.



(a) (b) G_1 (c) G_3^* (d) G_5^* (e) G_{10}^* (f)

Figure 3.3: The layout of a 2D slice from dataset *Isabel*. (a): Streamlines showing the flow field, where there are three vortices in the flow field. (b)-(e): The layouts generated by using the multi-hop graphs of G_1, G_3^*, G_5^*, and G_{10}^*. The color of each end point of the line segments reflects the layout order of the related block. (f): Colormap of the layout order (from the bottom color to the top color)

3.4.4 Visual Comparison of Layouts

We use a 2D static flow field to visually demonstrate the benefit of multi-hop graphs for layout computation. The flow field is plotted in Figure 3.3a, which is a 2D slice of the dataset *Isabel* where the flows mostly lay in the X-Y direction. Figures 3.3b-e show the layouts optimized for graphs generated by different numbers of hops. The line segments connect the nodes in the layout order. Line segments that connect non-adjacent blocks are drawn as thinner lines to avoid visual clutter of the visualization. The color of each end point of the line segments reflects the layout order of the related block. The color map is shown in Figure 3.3f.

From Figure 3.3b where the layout was generated from G_1, it can be clearly seen that the domain is partitioned majorly along the X or Y axis. This is because during the graph construction, most particles will move to their neighboring blocks in either X or Y direction. For such a graph, the resulting partition boundaries are usually also axis-aligned because cutting along the diagonal direction will take roughly double edges than that along

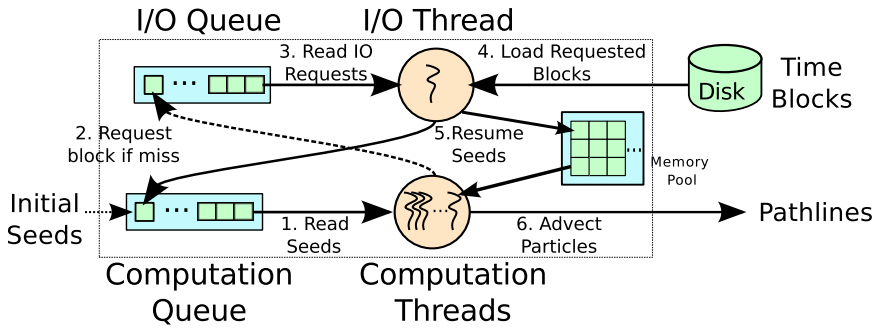


Figure 3.4: System for out-of-core parallel particle tracing.

the axes and therefore induces a higher cost. This results in that the generated layout does not follow the flow directions as shown in Figure 3.3a.

In Figures 3.3c-d, we can see that the generated layouts start to follow the flow directions better since the partition boundaries match the streamlines in Figures 3.3a. By closely inspecting the layout orders in Figure 3.3e, nevertheless, the layout for G^*_{10} becomes fragmented and a larger number of distant jumps occur. As this fragmented layout can be related to the complexity of the graph, it might cause higher miss ratio. We will further quantitatively evaluate the layouts in Section 3.6.2.

3.5 The Out-of-core Parallel Pathline Computation System

To compute pathlines out-of-core, we employ a multi-threaded system to overlap I/O time with computation time. In the system there are two types of threads, an I/O thread and several pathline computation threads. Each type is associated with a queue to manage I/O or computation requests, as illustrated in Figure 3.4. The I/O thread also manages a memory pool which caches prefetched or previously used data blocks. At the beginning of the program, the initial seeds are divided into separate tasks based on their block region.

All tasks are then placed into the computation queue. The computation threads remove tasks from the computation queue when they require more work, and check whether the data block required to advect particles in the task are ready in the memory pool. If not, an I/O request is placed into the I/O queue, and the particles in the task are temporarily suspended.

The I/O thread loads data blocks from disk based on the requests in the I/O queue. If the memory pool is full, currently unused blocks are released in the least-recently-used (LRU) manner. When the I/O is complete, the computation request is then pushed back to the computation queue and the previously suspended particles can proceed. In order to reduce I/O accesses while utilizing the generated file layout, the data reader uses prefetching to reduce the number of I/O accesses. Given a prefetch size, the prefetching loads consecutive data blocks ahead of the requested block in a single disk access. With an optimized file layout, the prefetched data cached in the memory pool will be used in the near future and hence will increase the hit rate of the cache.

3.6 Results

Three time-varying datasets were used in the experiments: *Isabel*, *Plume* and *Climate*. *Isabel* is a simulation of the hurricane Isabel from September 2003 over the West Atlantic region. The dimensions are $500 \times 500 \times 100$ with 48 time steps. *Plume* is a simulation of the thermal downflow plumes on the surface layer of the sun, with dimensions of $252 \times 252 \times 1024$ with 29 time steps. *Climate* represents a climate simulation over the Indian and Pacific Ocean. The dimensions are $2699 \times 599 \times 50$ with 25 time steps. We took the velocity fields representing the wind or flow speed. The first two datasets are provided by

Table 3.1: The ADG statistics for each dataset. The three numbers for the last column represent respective ADG $G_1/G_3^*/G_5^*$

Dataset	# Blocks	# Edges (millions)
Isabel	336,896	2.6 / 7.1 / 12.3
Plume	458,752	3.0 / 9.5 / 16.5
Climate	616,512	3.1 / 11.3 / 22.0

the National Center for Atmospheric Research and the last is by Pacific Northwest National Laboratory in the United States.

For all tests, the datasets were first reorganized in time blocks, each of which contains two consecutive time steps of 16^3 spatial blocks with two layers of ghost cells in the spatial domain. The use of ghost cells requires extra storage overhead, but can facilitate interpolation of data values across block boundaries. A time block spans time steps between one and two, two and three, and so on, for efficient temporal interpolation within the block. The resulting file sizes for dataset *Isabel*, *Plume* and *Climate* in time blocks are 47.2 GB, 64.2 GB and 86.3 GB, respectively.

The performances of the layouts optimized for ADG G_1, G_3^* and G_5^* were compared, as well as the use of the Z-order space-filling curve (Z-curve) layout as the baseline. The Z-curve layout, which is widely used for multi-dimensional data [116], traverses the space in different dimensions at each recursion level. The time dimension in the forward direction was assigned the finest traversing step of the Z-curve layout.

3.6.1 Preprocessing

Here we report the preprocessing time and related statistics for the computation of each ADG and the corresponding file layout. In the experiments, each ADG edge weight was

Table 3.2: The preprocessing time. The three numbers for the timings represent respective ADG $G_1/G_3^*/G_5^*$

Dataset	ADG Generation	Layout Computation	File Reordering
Isabel	8 / 28 / 45 min	6 / 7 / 8 min	15 min
Plume	16 / 58 / 104 min	12 / 14 / 16 min	21 min
Climate	17 / 74 / 145 min	23 / 25 / 26 min	43 min

computed by regularly placing one particle per 2^3 voxels at each time step. For consistency, Runge-Kutta 4th-order integration scheme was used for particle tracing in both graph construction and pathline computation. The out-of-core system described in Section 3.5 was used to trace particles and construct the ADG. The preprocessing was performed on a Linux workstation with an Intel Xeon CPU, 24 GB of RAM, and a RAID-5 disk array. To speed up, each ADG was generated using eight computation threads. Table 3.1 lists the statistics of each generated ADG. The preprocessing times, including graph generation, layout computation and file reordering, are listed in Table 3.2.

3.6.2 Sequential Particle Tracing Test

The I/O performance of the out-of-core pathline computation can be affected by both the design of the out-of-core system and the layout for the flow field. To verify the effectiveness of the file layout using a prefetching system, seeds were randomly placed in the domain and traced one particle at a time using various prefetch sizes. The prefetch size represents how many blocks ahead of the requested block are to be loaded into the memory in a single I/O call. Given a prefetch size p, the data loader reads $p+1$ blocks contiguously, including the requested block. Tracing one particle at a time ensures that the resident blocks in the memory pool are not prefetched by other seeds, which affects the accuracy of

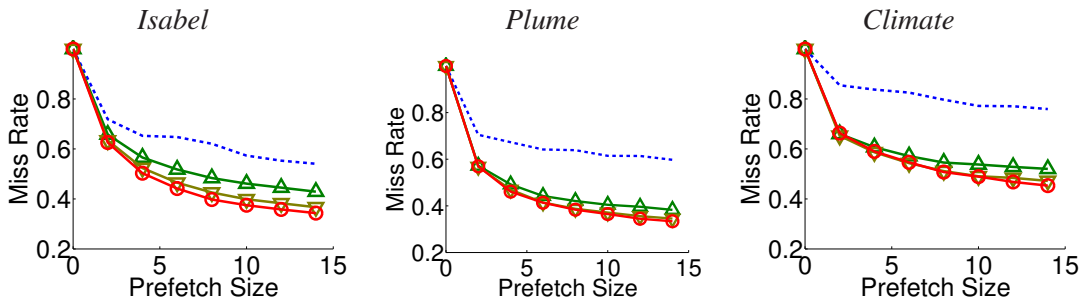


Figure 3.5: Layout validation through the miss rate of sequential particle tracing. The curves represent the Z-curve ($\cdots$) and the flow-guided layouts optimized for G_1 ($\triangle$), G_3^* ($\bigtriangledown$), and G_5^* ($\bigcirc$).

the miss rate. Tracing one particle at a time avoids data blocks being prefetched by other particles and thus reflects the layout performance. During sequential particle tracing, the number of disk accesses was counted, and in the end the miss rate was calculated by dividing the total number of disk accesses to load missing blocks by the number of blocks the seed has traversed. If the layout is good, more prefetched blocks will be used and hence the cache misses are reduced.

The results for the three datasets are shown in Figure 3.5. As can be seen, the graph-based layouts always achieve lower miss rates than the Z-curve layout. Among these layouts, the ones optimized for G_3^* and G_5^* achieve lower miss rates than that using only G_1 when the prefetch size is larger than two. This is because the ADG with higher number of hops contains more information about the runtime access pattern.

3.6.3 Runtime Performance

We tested the layout performances for out-of-core pathline computation based on different seeding scenarios. The first was *uneven seeding*, where particles were seeded in a small region, to visualize the flow starting from a specific region. The other was *uniform*

seeding, where particles were evenly distributed in the domain, which provides an overview of the flow field. For each seeding scenario, small and large seed sizes were tested. The experiments were run on a Linux machine with a 7200 RPM disk and 4 GB of RAM. Four computation threads were used with the Intel Core i7 - 2600 CPU. In all tests, the memory pool size was limited to 2 GB, which was able to hold around 14,000 time blocks. Based on different scenarios where scientists would like to understand the flow field, we tested the computation performance in two different seeding strategies:

1. *Uneven seeding* – Seeds are placed within a local region, which mimics a cubic probe used to visualize particles originated from a specific region.

2. *Uniform seeding* – Particles are evenly seeded in the entire domain, which provides a global overview of the flow field.

For each seeding strategy, 64 (4^3) and 4096 (16^3) seeds were used for different seeding densities. The experimental results of each seed strategy are described below.

A. Uneven Seeding In this seeding strategy, particles were seeded in a 32^3 region at the first time step. Several randomly selected regions were tested for each layout. During pathline computation, the miss count and the actual number of blocks loaded due to prefetching were counted, as shown in Figure 3.6. The miss count corresponds to the total number of I/O accesses due to a memory cache miss. A reduced miss count indicates better I/O efficiency. In the results using small seed size, we can see that the miss count of our layouts are all less than that of the Z-curve layout (blue bars) when the prefetch size is larger than one. Also, for each prefetch size, the miss count trends downward as the layout uses more hops. For large seed size, the same trend can be seen, with layouts which use more hops having the lowest miss count, except for the *Plume* dataset with similar miss counts.

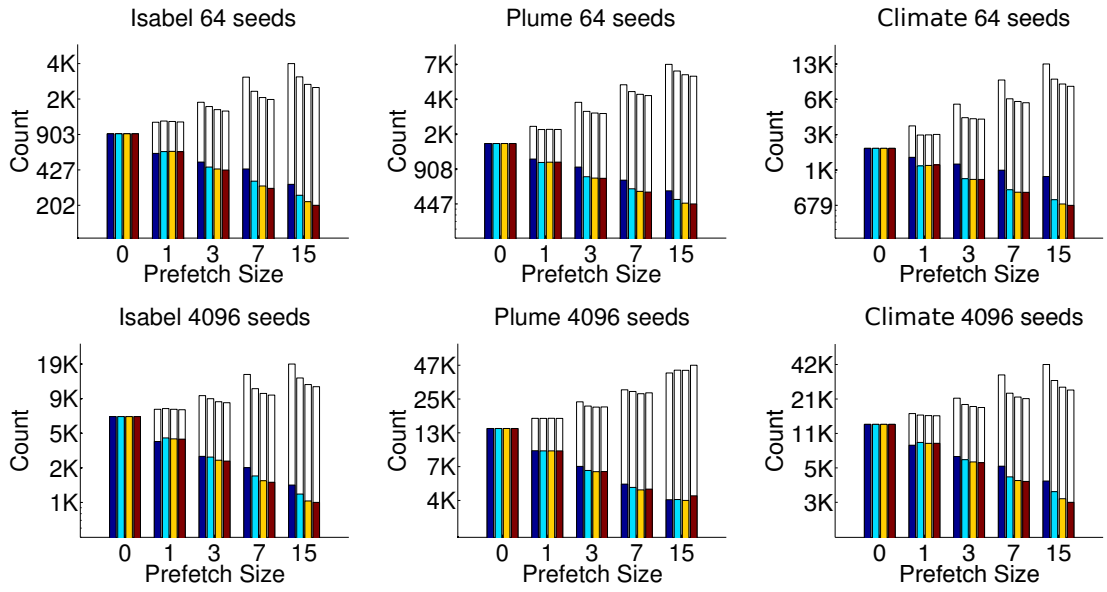


Figure 3.6: The miss count (color bars) and the total number of loaded blocks (white bars) for uneven seeding. Note that the box counts are plotted in logarithm scale. The bars in each group from left to right represent the result from the Z-curve layout and our layouts optimized for G_1, G_3^* and G_5^*.

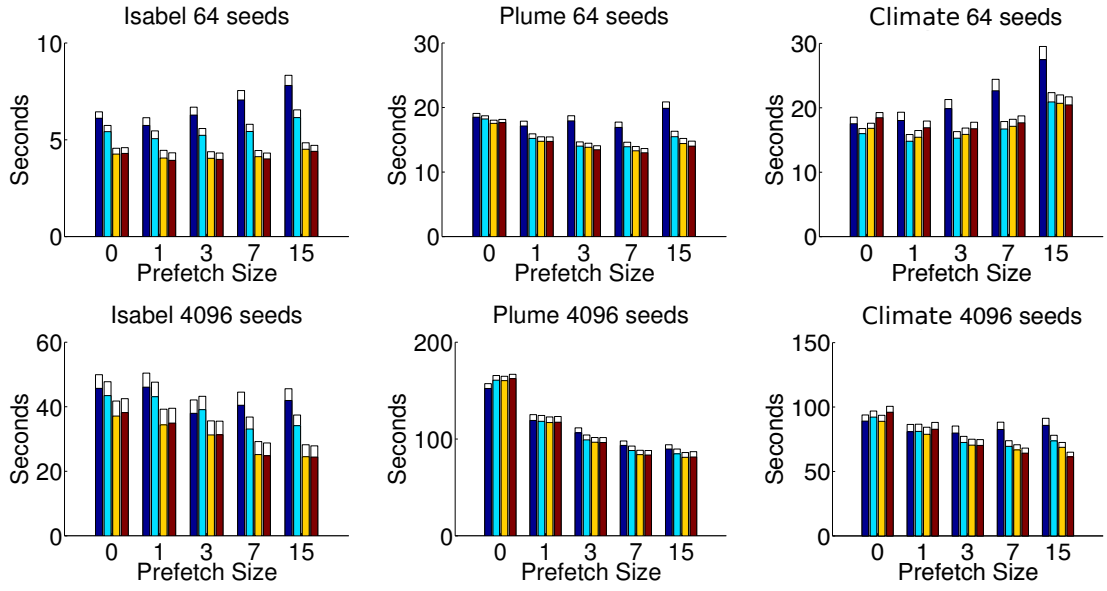


Figure 3.7: The block loading time (color bars) and wall time (white bars) of uneven seeding. The bars in each group from left to right represent the result from the Z-curve layout and our layouts optimized for G_1, G_3^* and G_5^*.

Figure 3.7 shows the accumulated I/O time (color bars) and total running time (white bars). The results of all test cases show that the total running time is very close to the I/O time, indicating that the I/O cost dominates the out-of-core pathline computation. When the prefetch size is larger than one, our layout achieves better performance versus the Z-curve layout. In most cases, better performance is gained from layouts which use a higher number of hops for the ADG. For each individual layout, we can see that as the prefetch size increases, the computation time first decreases, and then goes up again. This is because with an increased prefetch size, the miss rate should decrease, and the number of disk accesses is also reduced. However, when the prefetch size is too high, the miss rate cannot be further reduced, thus unnecessary disk reads will increase I/O time. Since our layout for *Climate* dataset generally has higher miss rates than our other datasets as shown in Figure 3.5, the best prefetch size to achieve shorter I/O time in *Climate* dataset is thus smaller than that for other datasets. From the results we can empirically determine that the best layout is our layout using G_5^*, and the proper prefetch size is within 3 and 7.

B. Uniform Seeding In this test, particles were evenly distributed in the entire domain. Unlike the previous seeding scenario where most particles share similar starting paths and mostly pass though the same blocks, this test is more I/O intensive because each seed will request data blocks in disparate regions of the flow field. Figure 3.8 shows the miss counts and the total number of loaded blocks, and Figure 3.9 shows the I/O and total running time. From the results we see trends similar to the uneven seeding results. When the prefetch size is larger than one, our layouts achieve higher performance than the Z-curve layout, and a decreasing runtime can be seen when using higher number of hops, except for using the *Climate* dataset or the 4096 seed set size. From the results, the layout optimized for G_3^*

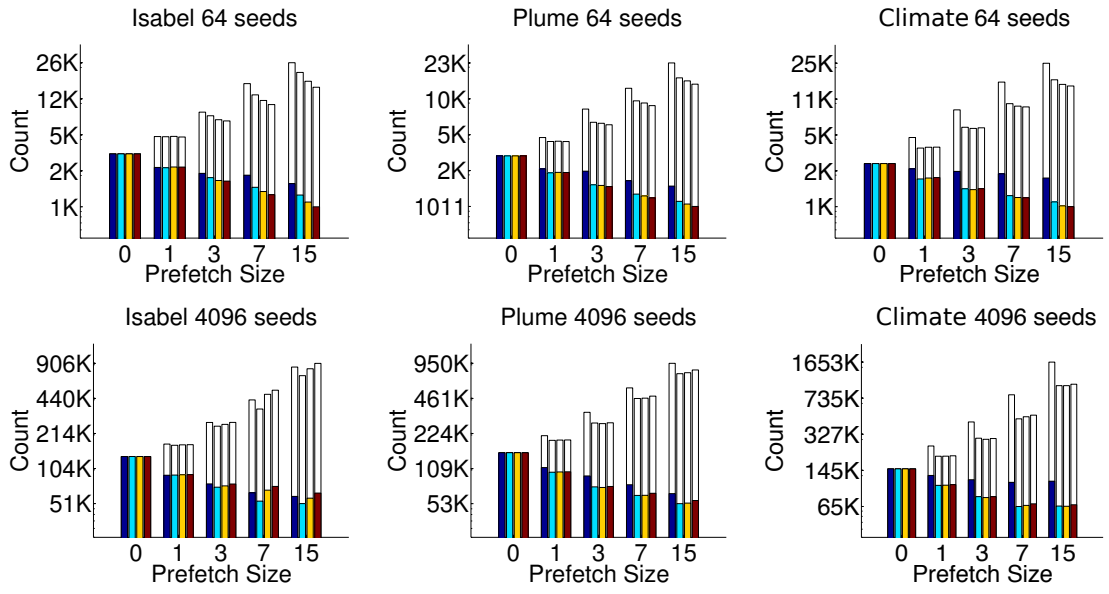


Figure 3.8: The miss count (color bars) and the total number of loaded blocks (white bars) for uniform seeding. Note that the box counts are plotted in logarithm scale. The bars in each group from left to right represent the result from the Z-curve layout and our layouts optimized for G_1, G_3^* and G_5^*.

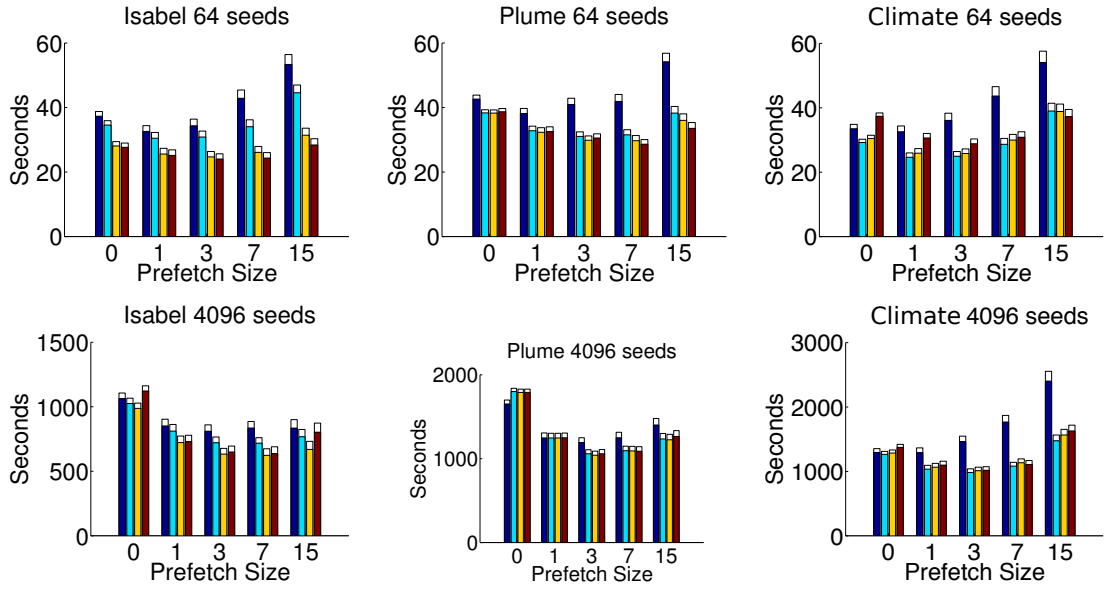


Figure 3.9: The block loading time (color bars) and wall time (white bars) of uniform seeding. The bars in each group from left to right represent the result from the Z-curve layout and our layouts optimized for G_1, G_3^* and G_5^*.

Table 3.3: The percentage of the runtime reduction of our layout using G_3^* over the Z-curve layout with the prefetch size of 7.

	Uneven seeding		Uniform seeding	
Dataset	64 seeds	4096 seeds	64 seeds	4096 seeds
Isabel	41%	38%	39%	25%
Plume	21%	10%	29%	31%
Climate	24%	19%	31%	36%

achieves the best total time in most cases, and the proper prefetch size is still within 3 and 7.

Table 3.3 lists the percentage of runtime reduction when using the layout computed from G_3^* , compared to the running time of the Z-curve layout, both under the prefetch size of 7. As shown in the table, 10%-41% running time reduction can be achieved.

3.7 Summary

In this chapter, we have presented a data layout method and a runtime prefetching scheme for out-of-core pathline computation. We compared our layout with the Z-curve layout in different seeding scenarios using various parameters, and showed that with the knowledge of the access pattern encoded in ADGs, the proposed layout algorithm can achieve higher I/O efficiency. An ADG of 3 and 7 hops achieves best I/O performance. Although ADG construction with higher numbers of hops dominates the preprocessing time to compute layouts, we expect to gain better performance using CPUs with a higher number of cores or on GPU, since the bottleneck of ADG construction is in the computation of particle advection.

Chapter 4: Graph-based Seed Scheduling for Out-of-Core Pathline Computation

In this chapter, we present a scheduling algorithm for computing high resolution FTLE fields from densely seeded pathlines on a multicore desktop machine. An FTLE field is a scalar field proposed by Haller [61] to quantify the separation rate of neighboring flow trajectories in a time-varying flow field over a finite time interval. FTLE measures are effective for depicting the separation of flows that are not easily seen by directly inspecting the velocity field itself. Although the FTLE field is useful to reveal important structures, its computation cost is much higher than most flow visualization tasks. This is because to measure the separation rate of flow in a local region, it is necessary to compute the *flow map*, by tracing particles densely seeded in the domain, e.g., from every grid point. Moreover, particles are seeded at every time step to obtain a sequence of FTLE fields. Many methods have been proposed to speed up the flow map generation [136, 49, 21, 64]. Without trading accuracy for speed, some methods compute densely seeded pathlines in parallel using hardware acceleration such as GPGPU [37, 50] or clusters [111]. While these methods provide out-of-core strategies to achieve high throughput by overlapping I/O and computation, they assume that the intermediate particle locations can be stored in memory during the flow map computation so that these particles can be advected together concurrently. As the datasets become larger, however, this assumption is not valid any

more, because the particles seeded in every grid point and every time step of the data will also require large amount of memory space.

To tackle the issue of processing excessive amount of particles, one can perform batch mode computation by advecting a smaller group of seeds at a time, under the memory space constraint. The whole flow map computation is thus divided into a number of rounds, where in each round only a selected group of seeds are stored in memory and advected until they meet the termination condition. Since memory capacity is limited, pathlines are computed out-of-core. Generally, efficient out-of-core computation requires full utilization of data while they are in memory. Since particles from different seeding locations and time steps may converge or follow similar paths, in order to prevent the same data from being loaded multiple times in different rounds, a good scheduling scheme that determines the grouping of particles to be advected together is thus needed.

To maximize the data usage and minimize the I/O cost for out-of-core FTLE computation, the single-*hop* ADG previously presented in the previous chapter is used, which can effectively model the local flow behavior in neighboring regions in both space and time. With a graph created from the flow directions, we analyze the data dependency during particle tracing for the entire time-varying flow field using discrete-time Markov chains. After the global data dependency is predicted, we group similar paths together and schedule the seeds accordingly to perform batch computation of FTLE. Experiments show that the proposed scheduling can achieve fewer disk accesses compared to a general space-filling curve ordering of particle advection.

4.1 Background and Related Works

In this section, we provide a brief overview of FTLE (Finite-time Lyapunov Exponent) and discuss the related techniques to speed up its computation. FTLE is a scalar which measures the separation of flow in its neighborhood along a finite time period. Haller [61] showed that the ridges of the FTLE field can be used to identify the Lagrangian coherent structures (LCS). The LCS divides distinct regions in the dynamical system and is useful for the visualization and analysis of transport barriers [145]. Pobitzer et al. provided an overview of the techniques for topology-based visualization of unsteady flow fields [118].

The calculation of FTLE depends on an accurate measurement of flow divergence, which is computed by tracing particles over an infinitesimal distance for each point of interest. Given the integration interval, we generate a mapping between the start and end positions of particle transportation, usually called the *flow map*. Formally, given a time-varying velocity field $\mathbf{v} = f(\mathrm{x},t)$ with $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, its flow map is defined as:

$$\phi_{t_0}^{t_0+T} : \mathbb{R}^n \to \mathbb{R}^n,\ \mathbf{x}_0 \mapsto \phi_{t_0}^{t_0+T}(\mathbf{x}_0) = \mathbf{x}(t_0+T;t_0,\mathbf{x}_0), \tag{4.1}$$

where $\mathbf{x}(t_0+T;t_0,\mathbf{x}_0)$ is the location of a particle trajectory (or pathline) starting from point $\mathbf{x}_0$ and time t_0 after traveling a time interval T under the velocity field. Since we consider the differences of trajectories within an infinitesimal neighborhood, the gradient of the flow map $\nabla\phi_{t_0}^{t_0+T}$ is used to form the *right Cauchy-Green deformation tensor*, defined as

$$\Delta_{t_0}^{t_0+T} = \left(\nabla\phi_{t_0}^{t_0+T}\right)^* \nabla\phi_{t_0}^{t_0+T} \tag{4.2}$$

where $(\cdot)^*$ represents matrix transpose. The square root of its maximal eigenvalue, $\sqrt{\lambda_{max}(\Delta_{t_0}^{t_0+T})}$ corresponds to the maximal local change in distance due to deformation. Taking the exponent, FTLE is defined as

$$\sigma_{t_0}^{t_0+T} = \frac{1}{T}\ln\sqrt{\lambda_{max}(\Delta_{t_0}^{t_0+T})}. \tag{4.3}$$

The FTLE values depend on the input starting time t_0 and integration time T. Usually an appropriate integration time T is assigned by the scientists and the FTLE fields regarding every starting time t_0 is computed.

In practice, in order to generate high resolution FTLE fields, it is necessary to construct the flow map by seeding and tracing particles as densely as possible. Since this computation requires extremely large amounts of processing time, several accelerative approaches have been proposed. One strategy is to approximate the flow map by adaptively reducing the spatial seeding density based on the surrounding flow divergence. Garth et al. [49] presented an adaptive refinement algorithm to approximate FTLE given an error constraint. Sadlo et al. filtered the computation of FTLE on expected ridge regions [136], and a grid advection technique to track the ridges was then proposed [137]. Kasten et al. [83] introduced *localized* FTLE that reuses the FTLE values from the previous steps. To gain further speed up, though with more error introduced, Brunton and Rowley [21] and Hlawatsch et al. [64] both presented similar ideas to approximate the flow map by hierarchically interpolating short integrals over time. Agranovsky et al. [3] extended Hlawatsch et al.'s work to *in situ* computation.

As parallel computation platforms become more pervasive, instead of reducing the sampling seeds, another strategy is to use GPUs and APUs (Accelerated Processing Units) to compute FTLE in parallel for mid-scale datasets [37, 50], or to use HPC platforms to handle very large datasets [111]. Guo et al. [60] extended *DStep* [85], a MapReduce-like particle tracing framework, to achieve high scalability on HPC for FTLE computation.

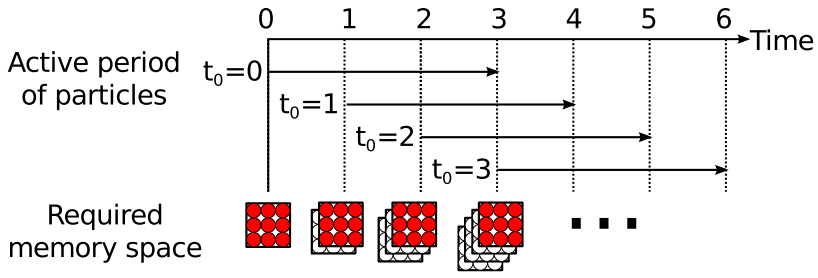


Figure 4.1: Illustration of a typical out-of-core particle tracing strategy for flow map generation. Assuming the integration step is 4, starting from the 4^{th} time step (time step 3) the system has to store all particle locations from the current (red circles) and the previous three time steps (white circles) in memory.

4.2 The Current Challenge and Proposed Framework

In this section, we discuss the practical challenge to compute high-resolution FTLE fields with a limited memory capacity. Then we provide an overview of the proposed out-of-core particle tracing system to tackle the problem.

4.2.1 The Challenge of parallel FTLE computation on a memory-limited system

To speed up FTLE computation, existing hardware-accelerated methods [37, 111] typically divide and assign pathline computation tasks into a number of processing units. When the data size is large, instead of waiting for the entire flow field to be loaded from the secondary storage device to the main memory, most methods use the strategy that loads and advects particles iteratively from the first time step to the last during forward particle tracing. Since particles will never go backwards in time, once all particles have passed a time step, the corresponding flow field data can be released for future time steps. Double-buffering can be used during the computation - one for particle advection in the current time step, and the other for the I/O thread to preload data in the next time step. Figure 4.1

illustrates this strategy. This method is I/O efficient because each time step of data will be loaded only once for the entire computation.

Despite its I/O efficiency, the above method is in fact difficult to scale. As shown in Figure 4.1, this method assumes that all the intermediate particle positions are stored in memory. Storing the active particle positions, however, will require many times the size of a time step of the flow field. For example, to generate a flow map with 20 integration steps for all time steps, the number of active particles from time steps 1 to time step 20 will continue to accumulate till the 20^{th} time step. For a dataset of dimension 500^3, around 40 gigabytes of memory will be needed to store all the temporary particles in the format of vectors in single-precision floating points. The memory requirement soon becomes infeasible for a normal desktop computer to proceed.

Since particles in the entire domain cannot be traced at the same time due to the limited memory capacity on a desktop computer, we can only allow a smaller portion of the seeds to be resident in memory at a time. For this reason, it is important to break the particles into groups and schedule the execution of the particles carefully to maximize the computation performance and minimize the I/O overhead. Below we present the main idea of our algorithm.

4.2.2 The Proposed Framework

When the memory capacity is limited, only a small group of seeds can be traced to compute pathlines at a time. We call the completion of computing pathlines for such a seed group a *round*. Since in each round not all the flow field data are to be used, the particle advection is performed in an out-of-core manner. Similar to the out-of-core system

presented in the previous chapter, the data are decomposed and stored in time blocks, each of which is a unit data block for one data access.

Although the out-of-core system has a memory pool to cache the loaded data blocks, we cannot assume those cached blocks remaining in one round can be reused in later rounds. Since reloading the same data block multiple times during the computation increases the number of disk I/O accesses and thus reduces the I/O efficiency, a good grouping and scheduling of seeds for each round is an important factor to improve the overall performance of out-of-core FTLE computation. Therefore in this work we employ a preprocessing stage to schedule the seeds for all rounds, as described below.

The seed scheduling is determined before flow map computation. It is a quick preprocessing aiming at reducing the potential number of disk accesses at run time. To do this, as how flow data are segmented into blocks, seeds in the seeding domain are also first divided into blocks, defined as *seed blocks*. In each round the flow map computation processes a group of seed blocks. The benefit for using the seed blocks is that it reduces the problem size for scheduling analysis, and the flow map generated in blocks is easier to store and index. For simpler analysis and implementation, we define the seed blocks with the same partitioning as the flow data blocks.

The next goal is to determine the optimal grouping of the seed blocks, where all seeds in one group will be advected in the same round. The basic idea is that the seed blocks that share similar flow trajectories will be grouped together, in order to reduce the number of data blocks to load in each round. The method to predict the particle trajectories from each seed block will be discussed in section 4.3.1. The seed group size is controlled by available memory size designated for storing the active seeds during particle advection. Figure 4.2 illustrates an example of seed grouping over different time steps.

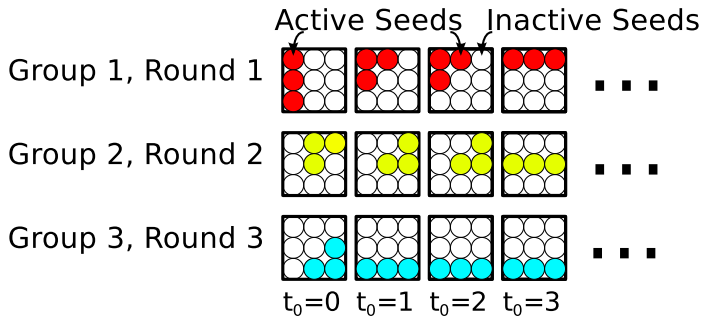


Figure 4.2: An example of seed grouping. With limited memory capacity for storing active particles, the initial seeds are divided into several groups and advected separately per group. A group can contain seeds from different time steps.

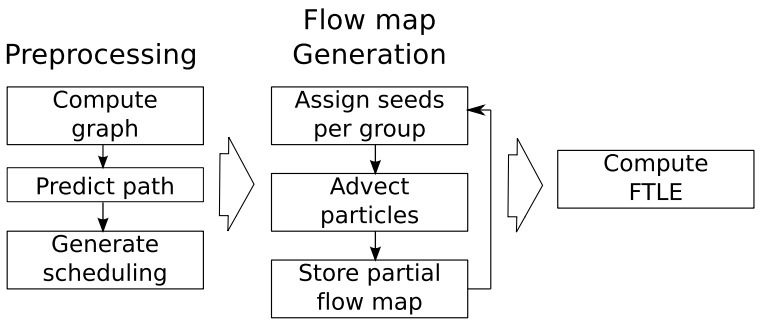


Figure 4.3: An overview of our out-of-core FTLE computation framework.

In addition to determining the optimal grouping of the seed blocks, it is necessary to determine the computation order of the groups. This computation order scheduling is to maximize the possibility of data blocks to be reused in consecutive rounds once they are loaded in the memory. Our algorithm is designed in a way that combines optimal seed grouping and scheduling together to ensure a high chance of data reuse, for arbitrary the seed group sizes. The detail will be described in section 4.3.2. The overview of the out-of-core FTLE computation framework is shown in Figure 4.3.

4.3 The Graph-based Seed Scheduling Method

The goal for scheduling seed advection is to reduce the number of data accesses and thus increase the performance of flow map generation. The idea is to minimize the number of blocks being reloaded due to memory misses. In the following sections, We first describe the path prediction method that estimates data blocks to be visited from each seeding block, utilizing a graph model and Markov chains. With the path information, we optimize seed grouping and scheduling by minimizing the total number of data block accesses for all possible seed group sizes.

4.3.1 Flow modeling and path prediction

To minimize the total number of data block accesses from different rounds of particle advection, it is necessary to have the information about what blocks a particle will visit from each seeding location. The naive way to obtain the exact path is through tracing all seeds with full integration steps, but the costly computation does not serve our purpose in the preprocessing stage. In order to provide a low-cost solution, we compute the flow trajectories in local regions and use the information to predict the global flow behavior.

4.3.1.1 Flow modeling

As stated in Section 4.2.2, during the out-of-core flow map generation the time-varying flow field is decomposed into blocks over all time steps. Since our goal is to minimize the number of block accesses, the flow field can be analyzed and modeled at the granularity of blocks. The inter-block dependencies of the flow field can be modeled as a directed weighted graph, using the access dependency graph defined in Section 3.3.1, which regards a time-varying flow field as a 4D dataset and aggregates all blocks from all time steps into

the graph. We will use the 1-*hop* access dependency graph, G_1 (denoted as G hereafter) to model the access dependencies between neighboring blocks and predict the longer-term access dependencies along the particle paths for the entire flow field.

4.3.1.2 Path Prediction by discrete-time Markov chains

Since in G the sum of all outgoing edge weights for each node is either one or less than one (when particles move into a sink or to the domain boundary), the graph weight $w(i, j)$ can be seen as the transition probability of a random walk (or a Markov chain) from node i to node j. That is, if there are particles uniformly distributed in block i, then in the next *step*, particles at the percentage $w(i, j)$ will move to block j. We can propagate the particles further using discrete-time Markov chains, by assuming that the probability of particles moving from one node to another in any step only depends on its previous step. By repeatedly propagating the probabilities through Markov chains from each initiating block, we can collect the visited nodes as the predicted dependent blocks related to their initiating block. The predicted paths are then used in our scheduling algorithm described later. Below we first describe the prediction algorithm.

The flow graph G with n nodes is first represented as an $n \times n$ adjacent matrix A. We use the format of column-stochastic matrix, where the j^{th} column of A represents the transition probabilities *out* of the vertex v_j, and thus A_{ij} represents the weight of edge from node j to node i in the graph G. We also denote the percentage of particles distributing among all nodes as an $n \times 1$ column vector x, where x_i represents the percentage of particles in node i. Note that the sum of values in vector x in any step is always less or equal to one:

$$\sum_{1 \le i \le n} x_i \le 1. \tag{4.4}$$

The propagation of particles among the nodes from step s to step $s+1$ can be written as a matrix-vector multiplication:

$$x^{s+1} = Ax^s. \tag{4.5}$$

We can also stack several different column vectors together to form a probability distribution matrix X and propagate the particle distribution at once using a matrix-matrix multiplication:

$$X^{s+1} = AX^s. \tag{4.6}$$

Essentially, the j^{th} column vector X_j^{s+1} represents the particle distribution propagated from X_j^s.

Since the final goal is to collect the dependent blocks with particles initiating from each seeding block separately, we assign the initial stacked probability distribution matrix as an $n \times n$ identity matrix, i.e.,

$$X^1 = I_{n\times n}. \tag{4.7}$$

In other words, we assign full probability (one) in distinct nodes for each column vector of X, and then iteratively propagate the distribution using Equation 4.6. After several steps of propagation, the j^{th} column vector X_j^s will represent the probability distribution of particles initiating from block j, since initially $X_{jj}^1 = 1$.

To collect the block dependencies, during each iteration we locate each non-zero elements X_{ij} in the matrix X, indicating that node i will be visited from node j in some step. To record these visited nodes we create an empty $n \times n$ matrix V and assign $V_{ij} = 1$ when X_{ij} is not zero. After running the above method for several steps, the matrix V represents the access dependency among nodes within these steps where each element V_{ij} in the j^{th} column of V represents whether the corresponding node i will be visited from node j.

To prevent the predicted access dependency matrix from collecting undesired blocks which have a tiny probability to be visited from the initial seed block, we use a threshold th and only collect the blocks which has the visiting probability larger than th. In other words, we assign $V_{ij} = 1$ only when $X_{ij} > th$. The thresholding reduces the resulting size of matrix V and only keeps the dependent blocks of higher confidence. The threshold value should be small in order to collect sufficient probable dependent blocks, but too small of a threshold value will require large storage space for the matrix V. The selection of the threshold parameter will be discussed in the experiment section. Algorithm 1 shows the pseudocode of the final prediction algorithm.

Algorithm 1 Discrete Markov-chain Path Prediction Algorithm

Require: Adjacent matrix A in size $n \times n$, threshold th.
1: Let X be an identity matrix in size $n \times n$.
2: Let V be an empty matrix in size $n \times n$.
3: **while** V not converge or exceed maximum iterations **do**
4: Propagate: Let $X = AX$
5: For each element X_{ij} in X, set $V_{ij} = 1$ if $X_{ij} > th$.
6: **end while**
7: **return** Matrix V as the prediction result.

4.3.2 Seed Grouping and Scheduling

After the prediction of block dependencies from each seed block is done, we derive a seed-scheduling algorithm to determine the grouping of seeds for each round and their computation order. The goal is to minimize the number of block accesses at run time. Below we first provide the cost function, which estimates the number of block accesses, and then present our optimization algorithm.

4.3.2.1 Grouping cost function

With the result of path prediction from the algorithm presented in the previous section, the cost function takes a grouping of seed blocks as the input and returns the total number of blocks to be accessed from the disk. We assume that the memory pool to store the data is large enough for the seeds in the same group to advect together for one time step without memory misses, i.e., no block will be released from memory and then reloaded due to a revisit in the same round. Since particles will never go back to the previous time steps, an easy choice of the data pool size to fulfill the above assumption is as large as the dataset size in one time step. A better choice of the data pool size will be discussed later in this section.

We also assume that the data pool is not big enough for any previously loaded data blocks to be reused for the later rounds. This is reasonable since the given integration step is usually long and thus a large number of blocks will be visited per round. Therefore the number of blocks to load in each group is the number of *distinct* blocks to be visited by the seed group, which can be obtained from the block visiting matrix V described in the previous section. Finally the total number of blocks to load for the entire process is the sum of the number of the counted distinct blocks from each group. We formulate the cost function as follows:

We use P to describe the input grouping as a list of k sets, where k is the number of groups. Let P_i contains a set of indices to the seeding blocks in group i. We denote each element in P_i as $P_{i1}...P_{im}$ where m is the number of elements in P_i. That is,

$$P_i = \{P_{i1}, P_{i2}, \ldots P_{im}\}, 1 \leq i \leq k. \tag{4.8}$$

Note that each index only appears once across the entire set of groups. Therefore the set elements in all groups are mutually exclusive, i.e.,

$$P_i \bigcap P_j = \emptyset \qquad 1 \leq i, j \leq k \wedge i \neq j. \tag{4.9}$$

The previous section (Section 4.3.1) has formulated an $N \times N$ matrix V which predicts the paths from every node. Recall that N corresponds to the number of seed blocks in the dataset. We denote V_j as the j^{th} column vector of V, where a non-zero element in the vector indicates that the corresponding block will be visited by the seeds from block j. Therefore the number of distinct blocks to be visited from a seed group P_i can be counted with the number of ones after merging the corresponding column vectors of V into one vector using the element-wise *logic-or* ($\vee$) operation, i.e.,

$$c(P_i) = ones(V_{P_{i1}} \vee V_{P_{i2}} \ldots \vee V_{P_{im}}). \tag{4.10}$$

Here *ones* counts the number of ones in the resulting vector. Finally given the grouping P, the cost representing the total number of blocks to load is

$$cost(P) = \sum_{P_i \in P} c(P_i). \tag{4.11}$$

Note that the visiting matrix V generated from the previous section records the dependent blocks where particles will visit from each seeding block at the maximum time steps. In FTLE computation, the actual integration time steps is constrained by the user-given parameter T. To adjust the cost specific for the user input T, after the general matrix V is generated, we can simply zero out the matrix elements whose corresponding data blocks are later than their initiating blocks by T time steps. Formally, given the matrix V and integration steps T, we assign $V_{ij} = 0$ when $Timestep(i) > Timestep(j) + T$, where $Timestep(\cdot)$ returns the corresponding time step for the input block index.

4.3.2.2 The Seed Scheduling Algorithm

As stated before, the group size, or the number of seed blocks per group, is limited by the memory space allocated for particle advection. Once the seed pool size is given, the group size will be equal to the seed pool size in bytes divided by the memory space to store the profiles of a block of seeds, including their positions. However in practice it is not trivial to determine the seed pool size because with limited memory space the size of the seed pool is competing with that of the data pool. We prefer the seed pool as large as possible so that fewer rounds are needed to process the entire seeds. Meanwhile, to fulfill the assumptions about the size of the data pool presented earlier, the data pool size should be large enough to prevent memory misses within each round. Fortunately, the minimum data pool size required for each round can be estimated through the predicted data block dependencies, supposed that the grouping of the seed blocks is determined. However, to optimize the grouping we need to provide the group size. As stated earlier, the group size is determined by the memory space available, which is affected by the unknown data pool size.

To solve this problem, instead of optimizing the grouping for a fixed group size, we generate a hierarchy of grouping where in each level the cost of the respective grouping is optimized. In this way the user can search for the setting of both data and seed pools in the memory space constraint with the minimum cost of runtime data accesses. In addition, by traversing the hierarchy we can determine the order to process the seed groups. This order serves as our seed scheduling order.

The grouping optimization problem is essentially a hierarchical clustering problem that groups seed blocks with similar paths together. As most clustering problems with optimal costs are NP-hard, we use an approximation algorithm to approach the minimum cost.

Since our cluster sizes are usually large, building the hierarchy using existing bottom-up hierarchical clustering methods with greedy clustering criteria is not an appropriate solution in our case. Instead, we find the optimal clustering by recursively dividing the seed blocks into two balanced-sized groups, as described below.

We approximate the optimal division by first converting the path prediction matrix back into a weighted undirected graph. Similarly, the graph has N nodes representing the N blocks. Given a pair of nodes, if by the predicted block dependencies there exists a path from one block to the other, we give an edge to the corresponding nodes and assign the weight as the count of intersected blocks between the paths starting from both blocks. Therefore, an edge with a lower weight means one node will less likely visit the other node, and thus can be separated during clustering. Formally, given the $N \times N$ path prediction matrix V, we create a converted undirected graph G' with N nodes. For each non-zero element V_{ij}, we add an edge (i, j) to G' with weight

$$w(i, j) = ones(V_i \wedge V_j) \tag{4.12}$$

where $\wedge$ represents element-wise *and* operation and *ones* counts the number of ones in the vector.

Given the converted graph, the top-down hierarchical clustering recursively finds the minimum balanced cut which results in two disjoint similar-sized subgraphs for each level. The minimum balanced cut essentially separates the seed blocks with minimum common paths between two groups. The METIS [82] library is used to efficiently approximate the minimum balanced cut. The minimum cut is performed recursively until less than two nodes are left in the subgraph. During the recursion, a binary tree representing the hierarchy is constructed. Each cut of the graph creates an internal node in the tree and at the bottom level the remaining one or two nodes are added as leaf nodes to the tree.

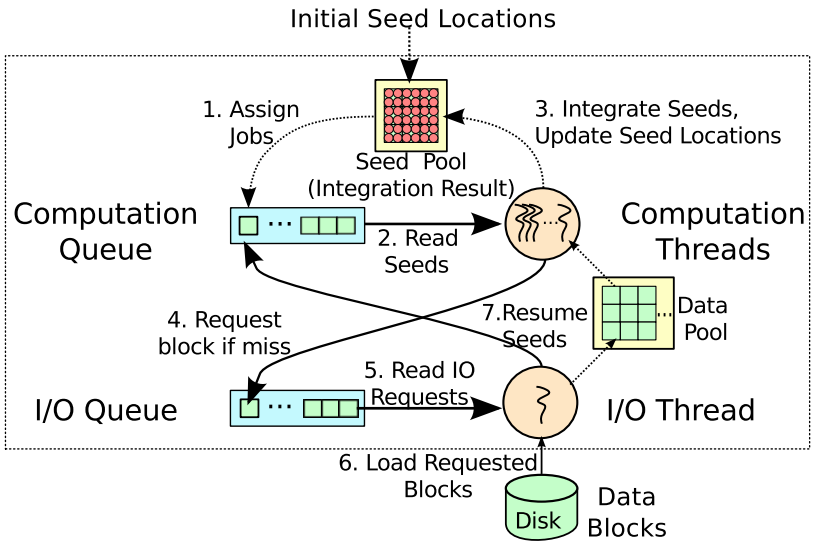


Figure 4.4: The out-of-core multithreaded particle advection system for flow map computation in rounds.

By traversing the tree in the depth-first search order and outputting the visited leaf nodes, we can obtain the execution order for the seed blocks. Then by separating this ordering into k groups, we obtain the seeding blocks to run for each round. Note that we still have the order among groups. Advecting seed groups in this order ensures that the data blocks visited from one group will have a higher chance to be reused by its sibling group, if they remain in the memory pool. Therefore, our scheduling method is optimized to achieve lowest number of disk accesses for different seed group sizes.

4.4 The Out-of-core Parallel Flow Map Computation System

After the grouping and ordering of seeds are determined, out-of-core pathline computation is performed accordingly, where a batch of seeds in each group are advected at a time to ensures that there are fewer seeds per round and their temporary positions during the advection can be stored in memory.

We extend the out-of-core system previously described in Section 3.5 to perform multiple rounds with different seed assignments, as illustrated in Figure 4.4. To further reduce the I/O costs between consecutive rounds, data in the memory pool from the previous round will remain for the next round in case they can be reused. In addition, in each round to ensure that seeds in the earlier time steps are processed first, for both I/O and computation we use a priority queue that releases jobs from the earliest time step in the queue. When the data pool is full, the I/O thread will start releasing an unused data block from the earlier time steps than the active time step for the seeds, since we know that those blocks will not be used anymore in the current round. When no data block earlier than the active time steps can be found, one unused block is released in the least recently used (LRU) manner.

4.5 Results

For performance evaluation, we used the same datasets presented in the previous chapter (*Isabel*, *Plume* and *Climate*). In the experiments, the datasets were first reorganized in time blocks, each of which contains two consecutive time steps of 32^3 spatial blocks, along with two layers ghost cells. This is the same setting as in the experiment of file layout method discussed in the previous chapter, but with a bigger block size for the spatial domain. The resulting file sizes for dataset *Isabel*, *Plume* and *Climate* in time blocks are 45.4 GB, 54.1 GB and 73.1 GB, respectively. All the timings of the experiments were measured on a Linux machine with a 7200 RPM disk, 4 GB of RAM and the Intel Core i7-2600 CPU supporting up to eight concurrent threads.

To show how better our proposed seed-scheduling algorithm performs, we used a four-dimensional Z-curve ordering of seed blocks as a competitive baseline, due to its high data locality. Given the number of seed blocks to advect per round, determined by the memory

Table 4.1: The graph statistics and preprocessing time for each dataset.

Dataset	# Blocks	# Edges	Graph Gen. Time	Path Gen. Time	Scheduling Time
Isabel	48,128	346 K	151 secs	23.2 secs	7.7 secs
Plume	57,344	327 K	676 secs	30.4 secs	19.3 secs
Climate	77,520	974 K	784 secs	136 secs	90.1 secs

space limitation, the Z-curve was uniformly divided into groups where seeds are activated and traced per group. This is the same strategy to schedule seeds from the ordering as described in Section 4.3.2. In addition, in the analysis we also tested a regular approach by applying a sequential ordering in *xyzt* order with the time dimension changing the slowest, which is the cases that the user simply groups and advects seeds per time step.

4.5.1 Path Prediction and Evaluation

Our path prediction method requires a one-pass process of the dataset to generate the access dependency graph. In our experiments, the graph was constructed by performing particle tracing within each time block, where the edge weights were computed by regularly placing one seed per 4^3 voxels at each time step. To generate the dependency graph, we used the parallel out-of-core system described in Section 4.4, but let all particles terminate at the border of each time block without propagating to the next block. Seven computation threads along with one I/O thread were used. The resulting graph properties and the computation time are listed in Table 4.1.

The Markov path prediction was implemented and run on Matlab using sparse matrices to save memory usage. To evaluate the quality of the prediction, we generated the actual access dependencies among all blocks by tracing regularly seeded particles from every seed

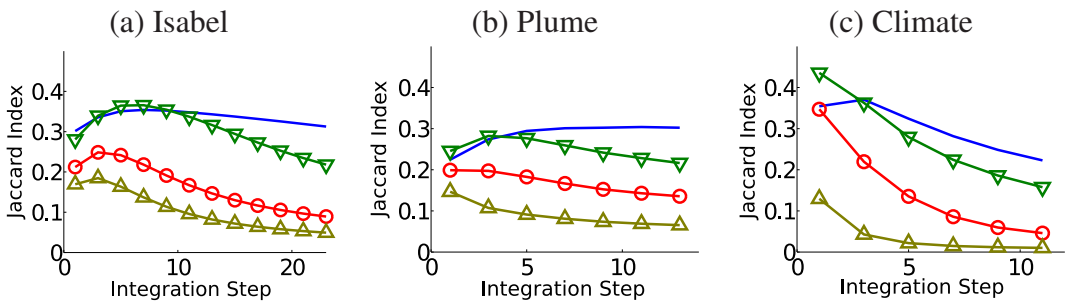


Figure 4.5: Evaluation of path prediction with threshold values $th = 10^{-4}$ (no mark), 10^{-3} ($\bigtriangledown$), 10^{-2} ($\bigcirc$) and 10^{-1} ($\bigtriangleup$). The Jaccard index shows the similarity between the predicted and the actual path.

block and gathered the dependent blocks along the particle advection paths. We considered the derived dependent blocks from each seed block as a set. To compare the similarity of the actual and predicted paths from each seed block, we used the Jaccard similarity coefficient, which is the fraction of the intersection over the union of two given sets , i.e,

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.13}$$

for two sets A and B. For each integration step, an average Jaccard index of the predicted and actual paths from all initiating blocks was computed.

Figure 4.5 shows the curves of prediction accuracy over increased integration steps with different confidence thresholds. As can be seen in the figure, by using a lower threshold, the predicted paths generally have higher average Jaccard index, meaning that they are more similar to the actual paths. We empirically chose the parameters with $th = 10^{-3}$ in our implementation, since the Jaccard similarity with this threshold is close to that of $th = 10^{-4}$, but requires less computation and storage to derive the predicted paths.

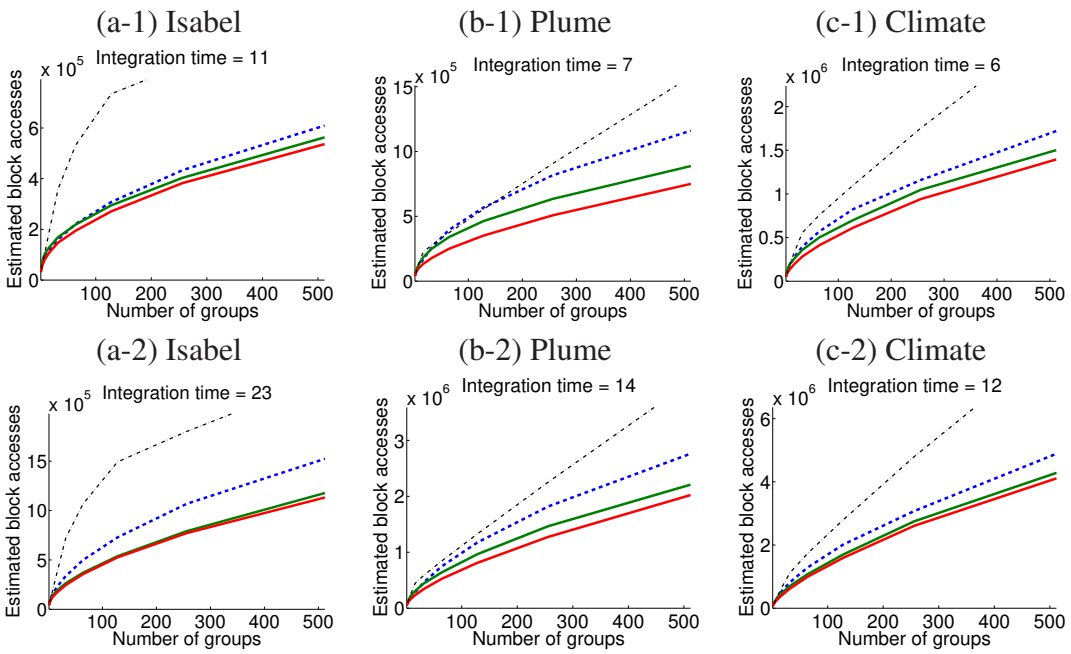


Figure 4.6: Evaluation of the scheduling by comparing the costs using the cost function in Section 4.3.2 with different group sizes. Two different integration steps were used: one forth (first row) and a half (second row) of the total time steps of each dataset. In each figure, the curves from the top to the bottom represent for sequential ordering (black dashed), Z-curve (blue dashed) and the optimized scheduling orders generated by our algorithm using predicted paths (green solid) and actual paths (red solid).

4.5.2 Scheduling Evaluation

After the dependent blocks for each seed block are computed, the grouping of the seed blocks and the order of pathline computation among them can be determined. To evaluate, we estimate the number of block accesses based on the cost function given in Section 4.3.2, where the path matrix is created using the actual paths collected as stated in the previous section. We compare the scheduling order generated from path prediction with threshold $th = 10^{-3}$ and that from the actual paths. This comparison shows how well the predicted

paths can be used to schedule the seed blocks to minimize disk accesses at run time. Four-dimensional Z-curve ordering of the blocks is also compared as a reference for a general but competitive scheduling order, due to its high data locality.

Figure 4.6 shows estimated numbers of block accesses with various numbers of seed groups and integration steps. The red and green solid curves represent the scheduling results generated by our algorithm using the actual and predicted paths. The blue dashed curve represents the Z-ordering result. As can be seen in the figure, the scheduling generated by our algorithm reduces more disk access times than the Z-curve ordering. Moreover, the result of scheduling by prediction is close to that of using the actual path. With more groups and longer integration steps, the discrepancy of disk accesses between our scheduling and Z-curve becomes larger. This means that the benefit of using our scheduling method is more significant when the dataset is much larger than the available memory and when the integration step is larger.

4.5.2.1 Performance Evaluation for Flow Map Computation

Now we compare the performance of flow map computation using our scheduling scheme with other methods described above. The integration steps in the experiments were assigned as half of the total time steps of each dataset, i.e., 23, 14 and 12 for *Isabel*, *Plume*, and *Climate* dataset, respectively. We tested different memory configurations of the seed pool with the size of 1, 0.5 and 0.25 GB, corresponding to 768, 384 and 192 seed blocks per group. Thus the *Isabel* dataset for example was tested with respective 32 ($= \frac{24{,}576}{768}$), 64 ($= \frac{24{,}576}{384}$) and 128 ($= \frac{24{,}576}{192}$) groups of the 24,576 seed blocks from the first 23 time steps. The data pool size was 2GB for all tests.

The disk access time was computed by summing up the time spent on all disk access function calls done by the I/O thread. As shown in Figure 4.7, the accumulated I/O time

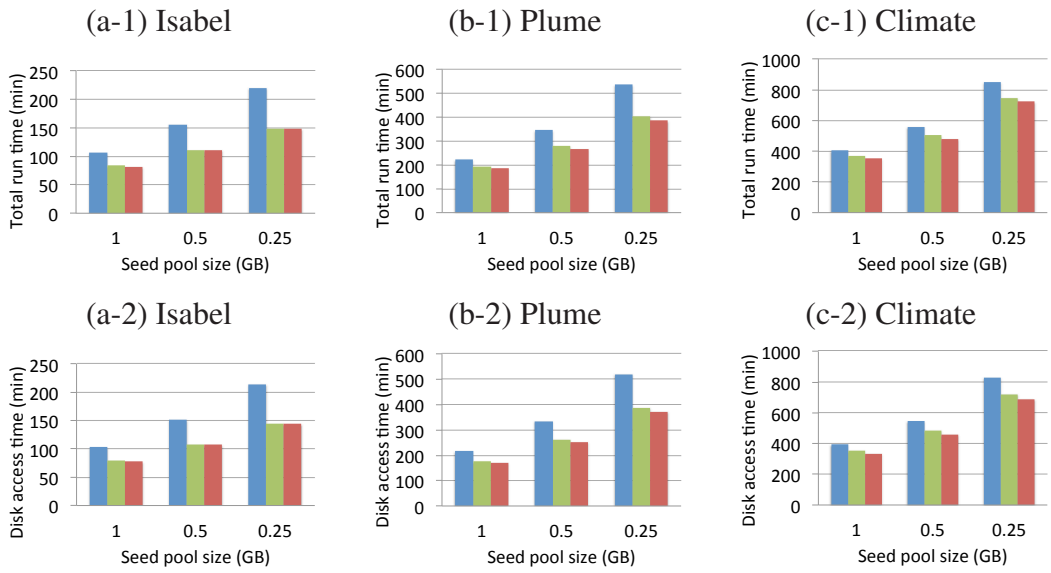


Figure 4.7: The comparison of total running time (first row) and disk access time (second row) with scheduling using Z-curve (blue bars), our algorithm from predicted paths (green bars), and our algorithm from actual paths (red bars). The integration steps were assigned as half of the total time steps of each dataset.

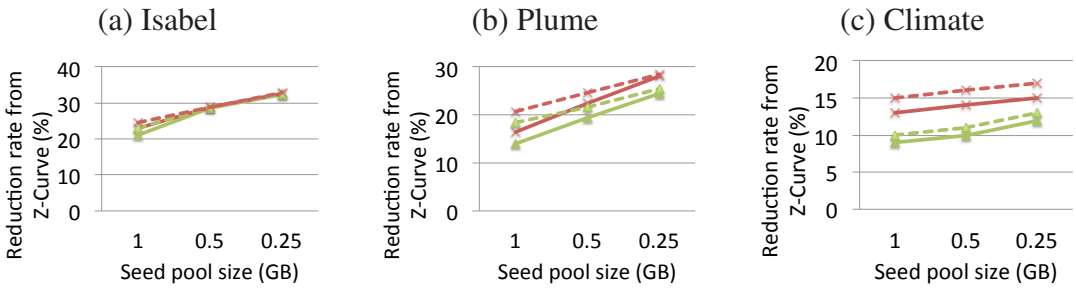
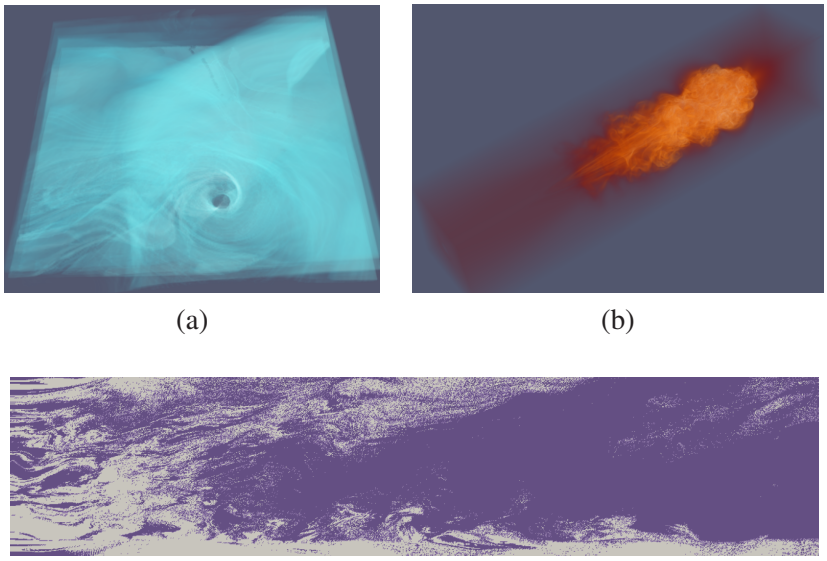


Figure 4.8: The comparison of total running time reduction (solid curves) and disk access time reduction (dashed curves) using our scheduling algorithm from predicted paths (green curves, marked with $\triangle$) and actual paths (red curves, marked with $\times$) over the Z-curve ordering.



(a) (b)

(c)

Figure 4.9: Visualizing the first frame of the FTLE fields for each dataset: (a) *Isabel*; (b) *Plume*; (c): *Climate*, showing one layer on the X-Y plane. The integration step was half of the respective total time steps. Brighter color means higher FTLE value.

of our seed scheduling from the predicted paths are lower than that of using Z-curve order, and this difference is also shown in the total running time. It can also be seen that when the number of groups becomes larger due to smaller memory pool sizes, the total disk access time becomes larger. Figure 4.8 shows the time reduction in percentage from the Z-curve ordering. By using our scheduling algorithm (shown as green curves in the figure), we achieved 10% - 33% reduction of disk access time and 8% - 32% reduction of total running time, over the method using Z-curve ordering. The final FTLE fields are visualized in Figure 4.9.

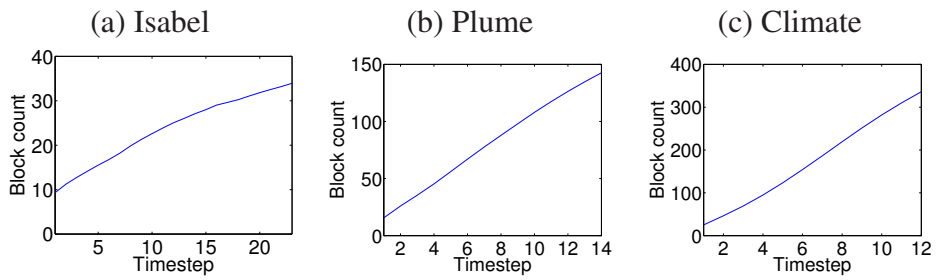


Figure 4.10: The number of dependent blocks in each time step in the actual path, averaged from each initial seeding block. Higher number of dependent blocks implies that the flow scatters more in the flow field.

4.6 Discussion

As listed in Table 4.1, the time spent on preprocessing to schedule the computation of pathlines finished within several minutes, far less than the pathline computation time to generate flow maps. As shown in Figure 4.6, our seed scheduling based on the predicted access dependencies can reduce the number of block accesses at a rate very close to that of using the actual pathlines to compute the access dependencies. Comparing the time reduction from using the Z-curve ordering, our scheduling method gained up to 25% speedup, and the time we can save is much larger than the required preprocessing time. This means with a little effort on the preprocessing we can reduce the flow map generation time more. Since the clustering problem is NP-hard, we currently do not have a way to show that our scheduling method is close to the optimal result, and we believe there is still room to improve. We leave the work of clustering optimization as the future work.

Among the three datasets, the *Climate* dataset does not have as much I/O and computation time reduction as the other datasets. We observe a higher divergence of the flow in the *Climate* dataset and hypothesize that the resulting high scattering of the dependent blocks

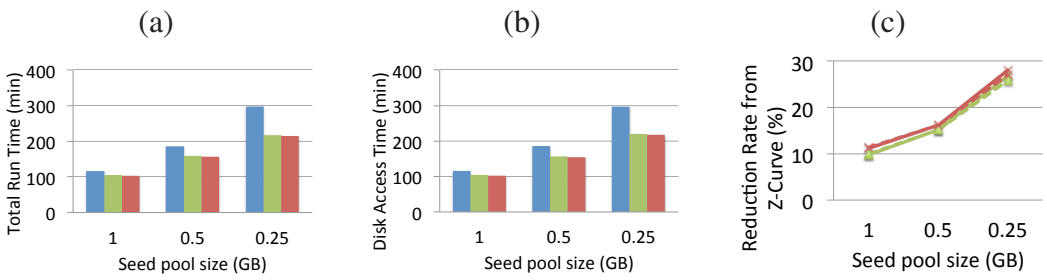


Figure 4.11: Run-time performance measurement of *Isabel* dataset stored in 64^3 blocks. Similar to Figure 4.6 and Figure 4.7, the colors in use represent for Z-curve (blueu), our algorithm using predicted paths (green), and our algorithm using actual paths (red).

makes clustering the seed blocks more difficult to optimize. Figure 4.10 shows the number of dependent blocks in each time step in the actual particle traces, averaged from each initial seed block. The *Climate* dataset has the highest block count. In addition, our method assumes that the seeds in the same seed block all move along a similar path. However this assumption may not be true when there is a critical point or highly divergent flow in the block or along the paths. A finer-grain clustering of the seed blocks can be applied in order to achieve more reduction of the number of disk accesses.

To see the effect of using a different block size, we further tested the performance of the *Isabel* dataset stored and scheduled in blocks of 64^3. As shown in Figure 4.11, our algorithm still out-performs the Z-curve based scheduling, but has less time reduction rate compared to the usage of 32^3 blocks in Figure 4.8a. This is because our algorithm has less flexibility to schedule the seeds in coarser grains. In addition, the overall running time generally takes longer than that of using 32^3 blocks. Therefore generally it is preferred that the selection of the block size is smaller. However, using a too tiny seed block leads to larger graphs in preprocessing. Note that although in our tests the seed blocks for scheduling have identical sizes as the data blocks, the user may want to use a smaller seed block size

when they cannot change the size of data blocks. This can be achieved by generating the flow graph and applying Markov chains to predict the block dependency with the smaller granularity of the seed blocks. We leave the improvement of the algorithm as future work.

4.7 Summary

This chapter has presented an efficient out-of-core seed scheduling algorithm for generating flow maps from large-scale time-varying flow fields, where the available memory is not large enough to hold the entire data set as well as the particle positions. We first model the local access dependencies as a graph and then compute the dependencies for longer integration steps using Markov chains. An algorithm is porvided to determine the processing order of the seed blocks using a divide-and-conquer method. We show that our scheduling method outperforms the simpler approach and the scheduling based on Z-curve ordering in both the estimated number of disk accesses and the total flow map generation time.

In this part, we have presented two approaches to improve the I/O performance of out-of-core pathline computation, both based on graph-based summary of flow behaviors. The first approach optimizes the file layout to facilitate disk prefetching, and the second approach optimizes scheduling for tracing excessive number of seeds. Both approaches attempt to minimize the total number of disk accesses and maximize the use of loaded data while in memory. While the scale of input data sizes in this part is still manageable by a single computer, the next part presents an I/O and storage efficient uncertainty modeling of flow fields due to unavoidable data reduction from large scale flow simulations.

Part III: Distribution-based Uncertainty Modeling

Chapter 5: Uncertainty Modeling and Error Reduction for Pathline Computation in Large-scale Time-varying Flow Fields

With the advancements in high-performance computer architecture, CFD (Computation Fluid Dynamics) simulations can now produce high-resolution flow fields with millions of spatial grid points and thousands of time steps or higher. A common problem is that machines for post-hoc analysis and visualization usually have much smaller storage space that can only handle a small fraction of the original simulation output. This gap will become even wider as exascale supercomputers are deployed [4]. As a workaround to reduce the data size while keeping the original spatial resolution, the common practice is to reduce the temporal sampling rate, e.g., once every tens or hundreds of time steps [105, 171]. Then the data values at time steps in between are approximated by linear interpolation. However, this inevitably introduces errors into the analysis and visualization results. Although the error at an individual point may be small, pathline computation, which integrates the interpolated vectors, will accumulate errors quickly, and thus the flow lines will become rather unreliable. While errors are inevitable, quantification and visualization of uncertainty is imperative. To visualize the uncertainty of flow fields, probabilistic pathline tracing has been proposed [102, 103, 113], where the flow field uncertainty is modeled in probability

distributions. While most of the uncertainty analysis algorithms assume that flow field uncertainty comes from measurements or ensemble runs, little focus has been on the analysis of interpolation errors caused by down-sampling time-varying flow fields.

In this chapter, we present a storage and I/O-efficient algorithm to model the interpolation errors for flow fields down-sampled in time. For each down-sampled value sequence we collect and store the distribution of the interpolation errors, which is later used to model the uncertainty of the interpolated values. To minimize the storage overhead, we observe that if a data sequence is fitted with least-squares regression, the error distribution can be approximated with a Gaussian distribution where its standard deviation is the standard error of the regression. Based on this idea, we propose a higher-order polynomial interpolation scheme and show that by storing only two extra parameters for each down-sampled value sequence, the interpolation error can be reduced while the error distribution needed for uncertain pathline visualization being approximated. Moreover, we show that our regression and error modeling method can be computed *incrementally*, by only one sequential scan of the flow data with memory requirement invariant to the number of time steps in the sampling interval. This incremental computation reduces the I/O cost and memory footprint of the error modeling procedure and is suitable for data streaming scenarios where data are available only for a short period of time and cannot be accessed afterwards. For example, it is suitable for *in situ* analysis of simulation output, where data are available in core only before being replaced by data generated in the next time step. With the error distribution, we further show that more accurate particle trajectories can be estimated from just the particle end positions at the sampled time steps. To achieve this, for each grid point $P1$ at the beginning of a sampling time interval, we perform pathline tracing but only store the particle end position $P2$ at the end of each time interval to save storage. To obtain a more

accurate pathline within this time interval without touching the data in between, we propose a forward-and-backward trace intersection technique that computes two uncertain pathlines from $P1$ and $P2$ respectively in the opposite directions toward each other, and then merge the two uncertain traces by the maximum likelihood principle. This refinement technique is shown to generate more accurate pathlines than conventional approaches, where only the down-sampled flow fields with our error model and the final particle positions per sampling interval are available.

The contributions of this research are thus threefold:

1. We present a storage-efficient error modeling method for interpolating down-sampled time-varying flow fields.

2. We show our error model can be computed incrementally to achieve efficient I/O and memory usage.

3. We provide a probability based method to refine the intermediate particle trajectories using our error model.

5.1 Related Works

Data reduction with uncertainty modeling has been discussed in Section 2.3. Here we focus on related works of uncertain flow visualization and polynomial fitting for data compression.

Uncertainty analysis and visualization have received considerable attention for more than a decade [77, 173]. Lately Brodlie et al. [18] provided a broad review including discussions about the sources of uncertainty and their visualization with different data types. For uncertain particle tracing, Luo et al. [103] and Love et al. [102] modeled the uncertain

velocities in PDFs and performed distribution-based particle integration based on Euler's equation. After each integration step, a representative particle location, e.g., the mean location, is extracted as the starting position for the next step. This approach approximates the actual pathline distribution with reduced computation complexity. In topology analysis of uncertain flow fields, Otto et al. [114, 113] constructed flow field topology using a Monte Carlo approach, which updates a particle density field describing the uncertain particle location. Schneider et al. [143] generated stochastic flow maps and measured the statistical flow separation analogous to FTLE in deterministic flow fields. The idea of our forward and backward trace intersection method can be related to estimation refinement methods incorporating information from the both ends of a time interval, such as optical flow computation in video analysis. In vector field analysis, recently Chen et al. [31] used forward and backward mappings to compute the approximate image for Morse decomposition.

Among the compression techniques that use polynomial fitting, Ameer [6] investigated polynomial fitting models for lossy image compression and showed that the linear polynomial model produces less error than JPEG when a high compression rate is desired. Fout and Ma [48] recently presented a lossless polynomial-based predict-coding method for floating-point value sequences.

5.2 System Overview

In this section we provide an overview of our incremental error analysis and pathline refinement method, shown in Figure 5.1. The first row of the figure shows a common procedure of pathline computation from down-sampled flow fields. The time-varying flow field is first sampled at a fixed time interval, referred to as the *sampling interval*, denoted as n. After data down-sampling, only the data at the two ends of the sampling intervals are

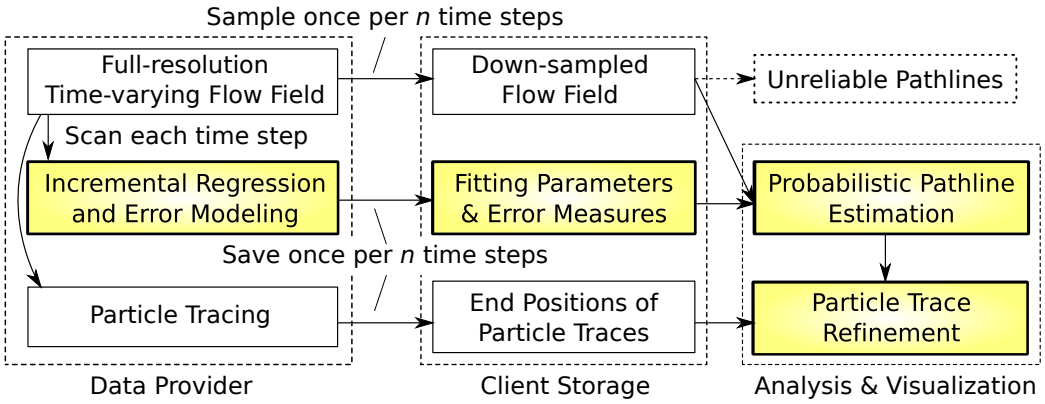


Figure 5.1: Overview of our system. White blocks: existing data down-sampling procedures. Color thick blocks: proposed error modeling method for uncertain pathline computation and refinement.

available for pathline computation, but the result will be unreliable, since the information in the unsaved time steps is lost and there is no indication of the error from linear interpolation.

To provide error measurement and control, we propose an error reduction and modeling scheme for down-sampled flow fields. As shown in the second row of Figure 5.1, we iteratively load each time step of the flow field and incrementally compute polynomial regression and its error measure over the time series of data. At the end of each sampling interval, i.e., every n time steps, we save the resulting polynomial parameters and the error measures to describe the uncertainty of the unsaved $(n-1)$ time steps of the data.

Since errors are accumulated quickly when computing pathlines from down-sampled flow fields, to improve the accuracy of the pathlines, we compute the end position of the particle originated from each grid point at the beginning of the sampling interval after traveling n time steps. This is similar to computing a flow map [62] for every n time steps. Storing flow maps in longer time steps has been used to improve the pathline accuracy and provide general information about the flow structure like FTLE [21, 64]. Given the flow

maps, we can utilize the error distribution computed above to refine the pathlines between the saved time steps. This component of our algorithm is illustrated in the last row of Figure 5.1, detailed in Section 5.5.

5.3 The Error Modeling and Reduction Method

In this section, we describe a storage-efficient method to model the uncertainty of data down-sampled in time, where the uncertainty is quantified by the distribution of temporal interpolation errors.

5.3.1 Terminology

We first define the terms that will be used in the rest of the chapter. Assume that the time-varying data at each step i, $i = 0, 1, 2...$ has a time stamp $t_i \in \mathbb{R}$. Given a grid point at a spatial coordinate $\mathbf{x}$, we form a sequence from a time-varying scalar field $\mathbf{F}$:

$$y_i = \mathbf{F}(\mathbf{x}, t_i) \in \mathbb{R}, i = 0, 1, 2, ... \tag{5.1}$$

Note that for a vector field in the flow field, we treat each vector component independently as separate scalar fields.

The time-varying flow field is sampled once every n steps, i.e., only data at steps $0, n, 2n, ...$, are stored. Thus interpolation is used to approximate y_i as $\hat{y}(t_i)$ at step i. The interpolation error is defined as the difference between the interpolated value and the true value, i.e., $y_i - \hat{y}(t_i)$. Below we explain our error modeling method for a time sequence at a single grid point within the first sampling time interval $[t_0, t_n]$. The same routine is applied to every grid point in every sampling interval.

5.3.2 Least Squares Estimation and Gaussian Errors

When modeling the distribution of a group of samples, Gaussian distribution is usually considered to provide a compact approximation. However, it is not always true that the samples are from a Gaussian distribution. Since our goal is to model the distribution of interpolation errors, one can alter the interpolation function, or fitting function, in a way that the error distribution is closer to a desired model. To do this, it is known that the least squares estimators optimize the function parameters by modeling the errors as a Gaussian distribution, and thus the function parameter estimation is equivalently a maximum likelihood estimation [15, 126]. With a regression function f, a data sequence $(x_i, y_i), i = 0, 1, ...n$, can be modeled as:

$$y_i \simeq f(x_i) + E, \tag{5.2}$$

where E denotes a random variable representing the error that can take positive or negative values. The least squares estimator approximates f by modeling E as a Gaussian distribution with a zero mean and an unknown variance, which is independent to the input x_i. Using the same model as in Equation 5.2, we fit the input data sequence with polynomial regression as the estimator and approximate the errors as a Gaussian distribution. In Section 5.6.2, we empirically validate this assumption with several fluid flow simulation datasets with statistical hypothesis tests.

With the errors of a least-squares regression modeled as a Gaussian distribution with zero mean, we only need to estimate and store its variance. Based on the least squares fitting, the unbiased estimator of the variance is derived [54] as

$$s^2 = \sum_{i=0}^{n} \frac{(y_i - f(t_i))^2}{(n+1) - K}, \tag{5.3}$$

where K is the number of parameters used in the regression. Note s is referred to as the *standard error* of the regression.

5.3.3 Quadratic Bezier Curve Fitting

To apply polynomial regression to our problem, it is necessary to determine an appropriate polynomial model to describe the data in the unsaved time steps. To deal with the challenge of large data sets, our goal is to find a polynomial model that requires the minimal storage space but with reasonable accuracy. Because in our framework the data in both ends of a sampling interval are stored, we enforce the polynomial function to always go through those end points. By doing so, the degrees of freedom are reduced by two for the polynomial and hence we have two less parameters to store. So, instead of using a general polynomial function, we choose the Bezier curve function, which is a form of polynomial with the end points fixed. Equation 5.4 defines a quadratic Bezier curve with two end points $\mathbf{P}_0$ and $\mathbf{P}_2$, and a control point $\mathbf{P}_1$:

$$\mathbf{B}(r) = (1-r)[(1-r)\mathbf{P}_0 + r\mathbf{P}_1] + r[(1-r)\mathbf{P}_1 + r\mathbf{P}_2], \tag{5.4}$$

where $r \in [0,1]$. The equation is also applicable to vector inputs. $\mathbf{B}(r)$ can be interpreted as an interpolation between the two end points $\mathbf{P}_0$ and $\mathbf{P}_2$. The equation also shows that $\mathbf{B}(r)$ is a quadratic polynomial of r.

The Bezier curve fitting also guarantees C^0-continuity for values on both ends of the sampling interval, which is not always true for a general polynomial since each sampling interval is fitted separately. Although Bezier curve fitting produces slightly higher error than general polynomial fitting of the same order, this drawback is relatively minor compared to the benefit stated above.

To select an appropriate polynomial order, we empirically choose the quadratic form, since for each input sequence it only requires one extra control point, hence less storage overhead, given that the two end points are always stored. We apply the quadratic Bezier curve fitting and model the Gaussian error for each value sequence within the sampling interval. As a result, for each sampling interval we store two extra values per grid point, the control point of the quadratic Bezier curve and the standard error.

Finally, based on Equation 5.2, for each grid point $\mathbf{x}$ at time $t \in [0,n]$ within a sampling time interval, we estimate the value V with Gaussian error by:

$$V_{\mathbf{x},t} \simeq \mathbf{B}_{\mathbf{x}}(t/n) + \mathrm{N}(0, s_{\mathbf{x}}^2), \tag{5.5}$$

where $\mathbf{B}_{\mathbf{x}}(t/n)$ denotes the Bezier curve function as in Equation 5.4 and $s_{\mathbf{x}}$ denotes the standard error corresponding to $\mathbf{x}$.

5.4 Incremental Error Modeling Algorithm

With the regression function defined by Bezier curves and the error modeled as Gaussian distributions, we show that the computation can be both I/O and memory-efficient for the data provider to analyze the original large data. We define our incremental algorithm with the following two constraints:

1. The algorithm reads the data in sequential time steps in only one scan. This reduces the I/O overhead of the error modeling procedure. It is also beneficial for analyzing streamed data where data are available in a sequential order.

2. The memory usage in the entire run should be invariant to the number of time steps in a sampling interval. Thus the algorithm is scalable to any arbitrary sampling rate. We assume that the memory can only cache a few time steps of the data.

Data regression and error distribution computation usually require two separate runs with the data, since the individual error, which is the difference between the estimated and the true value, cannot be computed before the regression function is determined. However, we show that the regression parameters and its standard error required in our error modeling can be computed together incrementally. As an illustration we first show that the RMSE of linear interpolation can be computed incrementally, which will then be used to derive the algorithm for our problem.

5.4.1 RMSE from Linear Interpolation

Given two temporal points (t_0, y_0) and (t_n, y_n), linear interpolation estimates the intermediate value $\hat{y}(t)$ at time $t \in \mathbb{R}, t_0 \leq t \leq t_n$ by

$$\hat{y}(t) = \frac{(t_n - t) \cdot y_0 + (t - t_0) \cdot y_n}{t_n - t_0}. \tag{5.6}$$

We will use $\hat{y}$ as the estimation to the input sequence. Thus the root-mean-square error (RMSE) of $\hat{y}$ is defined as

$$RMSE(\hat{y}) = \sqrt{\frac{1}{n+1} \sum_{i=0}^{n} (y_i - \hat{y}(t_i))^2}. \tag{5.7}$$

During the incremental computation, since the interpolated values $\hat{y}(t_i), 0 < i < n$ are unknown until the value y_n in the end point of the interpolation interval becomes available at step n, we cannot compute each error $(y_i - \hat{y}(t_i))$ when y_i is just produced. An intuitive approach is to store all values of $y_0 \cdots y_n$ before computing *RMSE*, but it violates our goal of using constant storage. However, with an arrangement of Equation 5.7, RMSE can be computed by incrementally accumulating some summations related to y_i at each time step. That is, by expanding Equation 5.7 and grouping each summation, we obtain:

$$RMSE(\hat{y}) = \sqrt{\frac{1}{n+1} \left(\sum_{i=0}^{n} y_i^2 - 2 \sum_{i=0}^{n} (y_i \cdot \hat{y}(t_i)) + \sum_{i=0}^{n} \hat{y}(t_i)^2 \right)}. \tag{5.8}$$

In Equation 5.8, the first term $\sum y_i^2$ can be incrementally accumulated at each time step; the last term can be summed at step n, since each $\hat{y}(t_i)^2$ can be computed using Equation 5.6, which only involves y_0 and y_n. The middle term needs further derivation using Equation 5.6, and by grouping each summation we get:

$$\sum_{i=0}^{n}(y_i \cdot \hat{y}(t_i)) = \frac{t_n \cdot y_0 - t_0 \cdot y_n}{t_n - t_0}\sum_{i=0}^{n} y_i + \frac{y_n - y_0}{t_n - t_0}\sum(y_i \cdot t_i). \tag{5.9}$$

As shown in Equation 5.8 and 5.9, by incrementally accumulating $\sum y_i$, $\sum(y_i \cdot t_i)$ and $\sum y_i^2$ at each time step, together with y_0 and y_n available at the last time step, the RMSE for linear interpolation can be sequentially computed. This is an incremental computation with one scan of the sequence, and it only requires constant memory for storing the above summations during the iterations, regardless of the length of the input sequence.

5.4.2 Incremental Quadratic Bezier Curve Fitting

In this section we provide the algorithm for incremental quadratic Bezier curve fitting and the standard error computation. In general, least squares regression optimizes the function parameters with minimum sum of squared errors to the data values:

$$\text{argmin}_f \sum_{i=0}^{n}(y_i - f(x_i))^2. \tag{5.10}$$

To optimize f is to solve the equation when the derivative of the above function equals to zero. For detailed derivation of solving polynomial regression the interested reader is referred to [126].

To optimize a quadratic Bezier curve, given an input sequence of time-value pairs $\{(t_i, y_i)|i = 0,1,2,...,n\}$, where each t_i is normalized into the range $[0,1]$, i.e., $t_i = \frac{i}{n}$, we define two end points $p_0 = y_0$ and $p_2 = y_n$, as the end points of the Bezier curve. One can

derive the equation for computing the control point p_1 as follows:

$$p_1 = \frac{\sum(y_i t_i) - \sum(y_i t_i^2) - p_0 \sum((1-t_i)^3 t_i) - p_2 \sum((1-t_i) t_i^3)}{2\sum((1-t_i)^2 \cdot t_i^2)}, \tag{5.11}$$

where $\sum(\cdot)$ is short for $\sum_{i=0}^{n}(\cdot)$. Equation 5.11 shows that we only have to preserve p_0 and accumulate the two partial sums $\sum y_i \cdot t_i$ and $\sum y_i \cdot t_i^2$ during the incremental computation. The rest summations can be pre-computed since they are not related to the data value y_i.

To compute the standard error for the results of quadratic Bezier curve fitting, we apply Equation 5.3 and assign $K = 1$ for one unknown p_1. We observe that the computation in Equation 5.3 resembles RMSE in Equation 5.7 with a different constant in the denominator (now it is n for $n+1$ data inputs). Therefore similar to the derived equation in Equation 5.9, we need to obtain the summations $\sum y_i^2$, $\sum(y_i \cdot \hat{y}(t_i))$ and $\sum \hat{y}(t_i)^2$ online, where $\hat{y}(t_i)$ is now estimated by the Bezier curve. Among these three summations, the first term has been summed up incrementally; the last term is summed by computing each $\hat{y}(t_i)$ from Equation 5.4, where p_0, p_1 and p_2 are known at the end of the interval. The second term is derived as:

$$\sum y_i \cdot \hat{y}(t_i) = p_0 \sum y_i + 2(p_1 - p_0) \sum (y_i \cdot t_i) + (p_0 - 2p_1 + p_2) \sum (y_i \cdot t_i^2). \tag{5.12}$$

Therefore, the standard error can also be computed incrementally by collecting an additional summation $\sum y_i$.

We summarize our incremental error modeling algorithm in Algorithm 2. Note that the same procedure is applied on each grid point at each iteration. In the end of each sampling interval the corresponding control points and the standard errors are generated and stored. As a result the extra storage required for each sampling interval is equal to twice of the data size of one time step, which is still relatively small compared to the original data size. What we gain is the error distribution with more accurate interpolation.

Algorithm 2 Incremental error modeling algorithm

Initialize $ysum$, $ytsum$, $yt2sum$ and $y2sum$ to 0, each representing $\sum y_i$, $\sum y_i \cdot t_i$, $\sum y_i \cdot t_i^2$ and $\sum y_i^2$, respectively.
for $i = 0$ **to** n **do** {For each time step in a sampling interval}
    Read data value y_i
    Let $t \leftarrow i/n$
    $ysum \leftarrow ysum + y_i$
    $ytsum \leftarrow ytsum + y_i \times t$
    $yt2sum \leftarrow yt2sum + y_i \times t^2$
    $y2sum \leftarrow y2sum + y_i^2$
end for
Compute the control point p_1 with Equation 5.11
Compute the standard error s with Equation 5.3, 5.4 and 5.12
return Bezier-curve control point p_1 and s.

5.5 Uncertain Pathline Estimation and Refinement

With the errors modeled as Gaussian distributions for the temporally down-sampled flow fields, in this section we present a probability based particle tracing algorithm to refine particle traces computed between the pre-stored time steps.

As we are processing data of the incoming time steps and computing the true particle traces incrementally using the full-resolution flow field, it is not possible to store the trace positions at every time step for every possible seed location. To reduce the storage overhead, without having to identify important pathlines through detailed analysis, one common solution is to store the particle traces only at certain sample time steps. The disadvantage of doing this, however, is that important details of the pathlines in the intermediate time steps are lost. In this section we show that with the error distributions computed from the temporally down-sampled flow field, as described previously, it is possible to identify the uncertainty and more importantly, to compute more accurate pathlines. In the following,

without loss of generality, we assume that pre-computed particle traces are sampled only at every n time steps, where n is the sampling interval used for the flow fields.

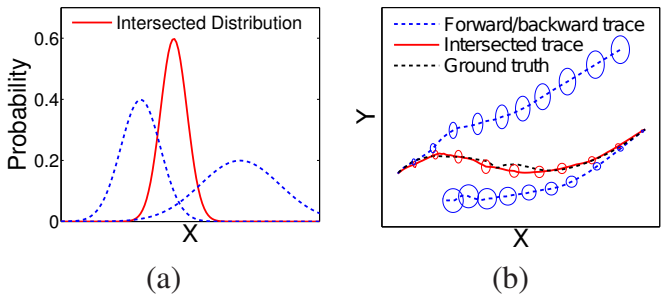
5.5.1 Probabilistic Particle Integration

We first review a general probabilistic particle integration method given an uncertain velocity field. To model a static 3D velocity field with uncertainty, we use a 3D random variable $\mathbf{V_x}$ to describe the probability distribution of the velocity at location $\mathbf{x}$ in the data domain D. That is, $\mathrm{P}[\mathbf{V_x} \subset [< u,v,w >, < u+du, v+dv, w+dw >]]$ denotes the probability of an uncertain vector at location $\mathbf{x}$, where its vector values are within the range $< u, u+du > \times < v, v+dv > \times < w, w+dw >$, where $(u,v,w) \in \mathbb{R}^3$. For a time-varying uncertain flow field at time t, we use the notation $\mathbf{V}_{\mathbf{x},t}$.

To model an uncertain particle position, we define a 3D random variable $\mathbf{X}_t$ as a particle density distribution at time t. To integrate the particle distribution, one can loop through all possible locations $(x,y,z) \in$ D and compute the probability of a particle moving to location (x,y,z) from the previous time step using Euler's integration method with $\mathbf{V}$. This convolution computation is involved with two nested loops of all possible locations. For more details the interested reader is referred to Otto et al.'s work [113].

5.5.2 Intermediate Trajectory Estimation

Given the particle positions on the same trace at the beginning and end time steps of a time interval, we show that with a temporally approximated vector field and the knowledge of errors, we can better estimate the particle trajectory in between. We observe that by utilizing a probabilistic particle tracing method, we can obtain two independently approximated particle traces within the same time interval–forward tracing from the starting position and backward tracing from the end position. These two estimations have a smaller



(a) (b)

Figure 5.2: (a): Illustration of intersection of two Gaussian distributions. (b): Illustration of our forward-backward trace intersection method. Each ellipse represents the uncertainty of the corresponding trace at a sampled time step modeled as a 2D Gaussian, where the radius is proportional to the standard deviation.

error close to the particle's initial position and a larger uncertainty in the end because errors are accumulated. To get a more accurate particle trace, we can combine these two approximated pathlines based on the co-occurrences of possible particle locations from both traces. Probabilistically, this is done by intersecting the two distributions:

$$\mathbf{X}_t^{\text{Intersect}} = \mathbf{X}_t^{\text{Fw}} \bigcap \mathbf{X}_t^{\text{Bw}}, \tag{5.13}$$

where $\mathbf{X}_t^{Fw}$ and $\mathbf{X}_t^{Bw}$ denote the respective forward and backward particle distributions at time t. Since both traces are independently estimated and thus the corresponding random variables are independent, the intersection can be computed by multiplying the two PDFs. Note that after the multiplication, normalization of the resulting probability distribution is required, so that the integration of the probabilities remains to be one.

Figure 5.2a illustrates the intersection of two Gaussian distributions. As we can see, the maximum likelihood of the intersection is closer to the distribution on the left, which has less standard deviation, or less uncertainty. In addition, the standard deviation of the intersected distribution is reduced and becomes less uncertain.

5.5.3 Pathline Integration Using the Gaussian Model

Since the flow field is modeled with errors as a Gaussian distribution, we present an efficient method to avoid the costly convolution computation for probabilistic particle tracing, by approximating the particle distribution at each time step with a Gaussian distribution.

A multi-dimensional Gaussian distribution X is defined by two parameters, the mean vector μ and the covariance matrix Σ. The PDF is typically denoted by $X \sim \mathrm{N}(\mu, \Sigma)$. Assuming $X \sim \mathrm{N}(\mu_X, \Sigma_X)$ and $Y \sim \mathrm{N}(\mu_Y, \Sigma_Y)$, the linear combination of these two Gaussian distributions is

$$aX + bY \sim \mathrm{N}(a\mu_X + b\mu_Y, a^2\Sigma_X + b^2\Sigma_Y), a, b \in \mathbb{R}, \tag{5.14}$$

which is another Gaussian distribution. We use this property to derive our Gaussian-based particle tracing method as the following.

Initially a deterministic seeding position $\mathbf{x}_0$ is given, i.e., $\mathbf{X}_{t_0} \sim \mathrm{N}(\mathbf{x}_0, \mathbf{0})$. To obtain the velocity at an arbitrary spatiotemporal position in a 4D domain, typically linear interpolation with the neighboring 16 grid points on a Cartesian space is performed. Since we use the quadratic Bezier curve as the interpolant for the time dimension, for each neighboring grid point we first perform temporal interpolation using Equation 5.4 and model the Gaussian error with the stored standard error s. Together we form a multivariate version of Equation 5.5 for the interpolated vector on a grid point $\mathbf{x}$ at time t:

$$\mathbf{V}_{\mathbf{x},t} \simeq \mathbf{B}_{\mathbf{x}}(t) + \mathrm{N}(\mathbf{0}, diag(\mathbf{s}_{\mathbf{x}}{}^2)). \tag{5.15}$$

Here $\mathbf{B}_{\mathbf{x}}(t)$ and $\mathbf{s}_{\mathbf{x}}$ are 3D vectors, and $diag(\mathbf{s}_{\mathbf{x}}{}^2)$ is a diagonal matrix with individually squared elements of $\mathbf{s}_{\mathbf{x}}$ on the diagonal. With the error modeled as a Gaussian distribution, the interpolated velocity at time step t on each grid point is also Gaussian. Therefore to

obtain $\mathbf{V}$ at an arbitrary position at time step t, we compute trilinear interpolation from the neighboring grid points using Equation 5.14.

To speed up the uncertain pathline estimation, we approximate the probabilistic particle integration by assuming that the probable particle locations are in a small local region, so that the velocities in this region are similar to that at the center of the particle distribution $\mu_{\mathbf{X}_t}$. This idea is similar to the uncertain pathline visualization method used by Luo *et al.* [103], where the velocity distribution at the representative particle location is applied at each integration step. With this idea we approximate the particle integration as the following equation, which resembles Euler's method:

$$\mathbf{X}_{t+\Delta t} \simeq \mathbf{X}_t + \mathbf{V}_{\mu_{\mathbf{X}_t},t} \cdot \Delta t, \tag{5.16}$$

where Δt is the integration step size. This computation involves another linear combination of the two Gaussian distributions, position $\mathbf{X}$ and the displacement as the velocity $\mathbf{V}$. By this integration method, $\mathbf{X}_i$ is always in a Gaussian distribution, which avoids the double-loop convolution computation and large memory requirement for storing the particle distribution described in Section 5.5.1. The computation of backward particle tracing is similar, by assigning a negative Δt in Equation 5.16.

Finally, to intersect two uncertain pathlines, we intersect the two random variables $\mathbf{X}_t^{Fw}$ and $\mathbf{X}_t^{Bw}$ at each corresponding time step. The following intersection equation for Gaussian distributions is derived [19]. Let $\mathrm{N}(\mu_c,\Sigma_c) = \alpha \mathrm{N}(\mu_1,\Sigma_1) \cdot \mathrm{N}(\mu_2,\Sigma_2)$, where α is the normalizing constant, we get:

$$\mu_c = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2) \tag{5.17}$$

$$\Sigma_c = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \tag{5.18}$$

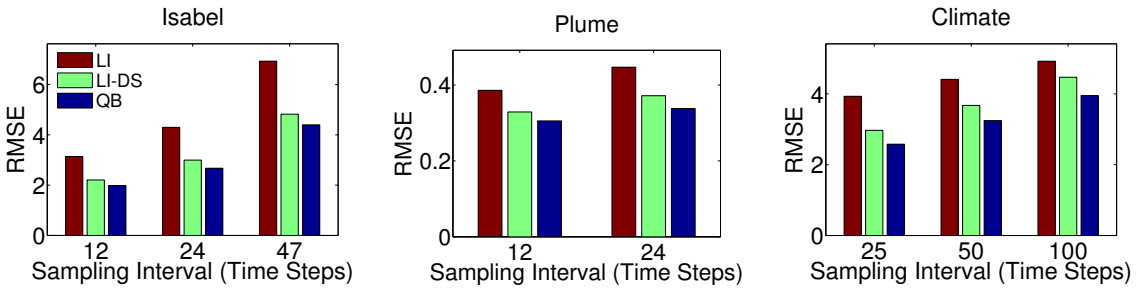


Figure 5.3: RMSE of different interpolation methods to the ground truth. Red: linear interpolation. Green: linear interpolation with a double sampling rate. Blue: quadratic Bezier curve interpolation.

As shown in Equation 5.17, the new mean μ_c is a linear combination of the two input mean values, weighted inversely by their respective variance. Therefore for each intermediate time step, the combined mean will be pulled to the one that is more certain, which is as desired for our particle path refinement.

Figure 5.2b illustrates our forward-backward intersection method applied on the test dataset *Isabel*. Given the start and end positions of a particle trace, we perform Gaussian-based particle tracing respectively and obtain two uncertain traces, as shown by the blue dashed lines. By intersecting these two compliment traces at each corresponding time step, we obtain the trace in red, which is very similar to the true pathline as the black dashed line.

5.6 Experimental Results

In this section, we show the experimental results of our error modeling of the flow field and the pathline refinement methods. Three time-varying datasets were used in our experiments: *Isabel*, *Plume* and *Climate*, as has been described in Chapter 3. We obtained 101 time steps for the *Climate* dataset in this experiment. Since different datasets have

Table 5.1: Percentage of samples where the errors of the estimation are within the 1.96 estimated standard deviation. (a): Test on the sequences of vector fields fitted by quadratic Bezier curves. (b): Test on the estimated trace locations using our forward-backward intersection method to the true locations.

Dataset	*Isabel*			*Plume*		*Climate*		
Sampling Interval	12	24	47	12	24	25	50	100
(a) Vector field	95.9	95.5	95.0	95.7	96.1	95.4	95.5	95.2
(b) Particle trace	94.3	86.9	83.0	72.9	78.8	89.6	75.7	64.8

different numbers of time steps, we chose 12, 24 and 47 time steps as the sampling intervals for *Isabel*, 12 and 24 for *Plume*, and 25, 50 and 100 for *Climate*.

5.6.1 Comparison of Interpolation Methods

Since our approach applies polynomial regression on the flow field data, to see how much errors were reduced, we compared the RMSE measures with our Bezier-curve interpolation and linear interpolation. Since in our method, each quadratic Bezier curve requires to store an extra control point, to perform a fair comparison we also compared the linear interpolation of data sampled with a double sampling rate; i.e., one extra time step of data in the middle of each sampling interval are stored and interpolated. Figure 5.3 shows the average RMSE measures with each sampling interval of different length. In each case, we can see that the quadratic Bezier curve interpolation (blue bars) can reduce the error the most given the same amount of storage space.

5.6.2 Validation of Gaussian-Distributed Errors

Now we validate our hypothesis that the errors from our quadratic Bezier curve fitting can be approximated as a Gaussian distribution. We first computed errors for each sampled data sequence, where an error sequence was formed by the differences between the true

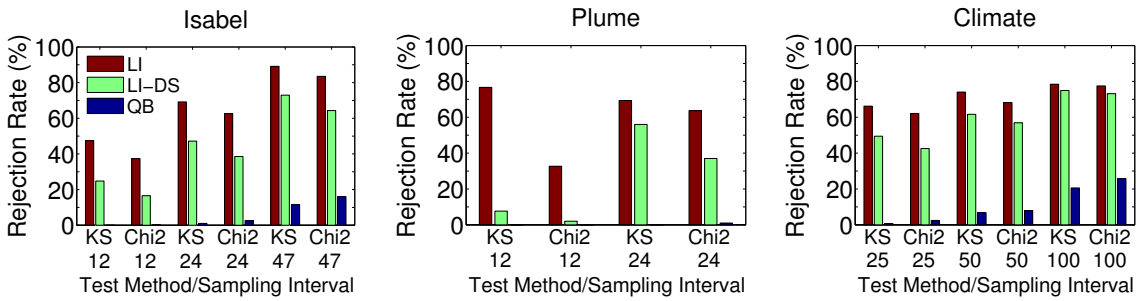


Figure 5.4: Validation of Gaussian errors using Kolmogorov-Smirnov test (KS) and Chi-square goodness-of-fit test (Chi2). The percentage of samples rejected by each test is shown. Red: linear interpolation. Green: linear interpolation with a double sampling rate. Blue: quadratic Bezier curve interpolation.

values and the approximated values. The sequence was then tested with two statistical tools, the Chi-square goodness-of-fit test and the Kolmogorov-Smirnov test against our hypothesis that the test sequence is a Gaussian with zero mean and the estimated standard error as the standard deviation. The former test is widely known while the latter is more robust and preferred nowadays. We used the threshold value (Alpha) 0.05 in both tests. That is, if the computed p-value is less than the threshold, our hypothesis is rejected. More than 10,000 randomly sampled data sequences were tested for each case and the rejection rate was recorded. For comparison, we also tested the normality of error distributions from using the original linear interpolation with and without the double sampling rate.

As shown in Figure 5.4, when the sampled sequences were approximated by the quadratic Bezier curve fitting (in blue bars), with 12 time steps as the sampling interval, less than 1% of the samples were rejected against our normality hypothesis by either of the test methods. With 24 or 25 time steps chosen as the sampling interval, less than 5% of the sequences were rejected. The percentage increased up to 17% with the sampling interval 47 or 50 time steps. The higher rejection percentage was 26% by the Chi-square goodness-of-fit test

Table 5.2: Comparison of the file sizes in each test case. The extra storage includes the error measures and the pathline end positions from each grid point and each sampling interval.

Dataset	*Isabel*			*Plume*		*Climate*		
Original File Size / Time steps	14.4 GB / 48			22.6 GB/29		98.0 GB / 101		
Sampling Time Interval	12	24	47	12	24	25	50	100
Down-sampled Time Steps	5	3	2	3	2	5	3	2
Down-sampled Flow Field Size (GB)	1.5	0.9	0.6	2.3	1.6	4.9	2.9	1.9
Total Extra Storage Size (GB)	3.6	1.8	0.9	4.7	2.3	11.6	5.8	2.9

with the sampling interval 100 steps in the Climate dataset. On the other hand, in all cases much more samples were rejected in the normality tests when the linear interpolation with or without a double sampling rate was used. The results show that compared to linear interpolation, the errors of Bezier curve fitting can be much better approximated by a Gaussian distribution with computable standard deviation.

Statistically, a Gaussian distribution has the property that 95% of the samples fall within the range approximately ± 1.96 standard deviations of the mean. We examine whether the estimated standard errors can provide this confidence interval. From the above sampled sequences approximated by the quadratic Bezier curve fitting, we counted the number of samples where the true error was within ± 1.96 standard deviation using the estimated standard error. As shown in row (a) of Table 5.1, the percentage of the sequences passing our test was larger than 95% in all the test cases. This test also shows the estimated standard errors can be used to provide the confidence interval.

5.6.3 Performance and Storage Requirement

We now evaluate the performance of our incremental error modeling algorithm. The test was performed on a Linux machine with Intel Core i7-2600 Quad-core CPU, 16 GB of

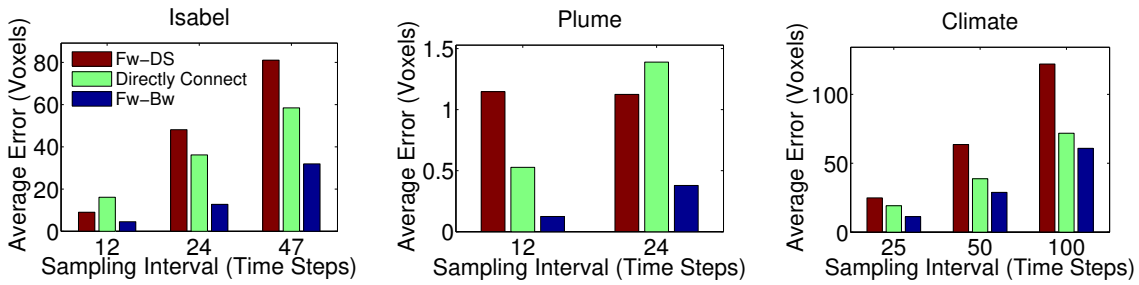


Figure 5.5: Average error of estimated trajectories to the true pathline. Red: Forward tracing on flow fields with a double sampling rate. Green: Directly connect. Blue: Our forward-backward intersection method.

RAM. Since we apply error modeling on the value sequence at each grid point individually, there is no dependency among different grid points and thus the computation can be easily parallelized. As presented in Algorithm 2, the incremental error modeling computation has two major tasks: first, it accumulates partial sums for each time step, denoted as ADD operation; Second, with the accumulated summations, it computes Bezier fitting and the standard error at the end of each sampling interval, denoted as FIT operation. We measured the operation time for each test case with four threads used. As expected, the computation time of each operation was invariant to the length of the sampling interval. The average ADD time was 168 ms, 420 ms and 282 ms respectively for *Isabel*, *Plume* and *climate* dataset; the average FIT time was 204 ms, 505 ms and 325 ms, respectively. The differences of above timings from various time steps and sampling intervals per dataset were within 2% of the mean.

The resulting storage for each test case is shown in Table 5.2. Given the sampling time interval n, the number of sampling intervals I in a total N time steps will be $I = \lfloor N/n \rfloor$. Thus $I+1$ samples will be stored including the first and the last time steps. Since at the end of each sampling interval our error model requires to store the Bezier control point and

the standard error for each grid point, if a time step of the flow field is in size of S, the total storage required is $(2I \cdot S)$. Furthermore, to obtain more accurate pathline, we store the flow map for each sampling interval and thus incurs additional $(I \cdot S)$ of storage. In total the extra storage in use is $(3I \cdot S)$.

5.6.4 Comparison of Intermediate Trajectory Estimation

With stored flow maps, i.e., end positions sampled from the true pathlines (computed from the full-resolution flow field), we estimate the particle traces in between, using three different methods:

1. **Regular forward tracing with a double sampling rate**: This is the conventional approach to compute pathlines using fourth-order Runge-Kutta integration method (RK4), where the sampled flow fields are linearly interpolated in the time dimension. Double sampling rate was used to further reduce the flow field error.

2. **Directly connect**: When particle end positions are given, we connect the beginning and end positions with linear interpolation.

3. **Forward-backward pathline intersection method**: This is our method incorporating the stored flow field with error modeling and the particle end positions.

We compared the accuracy of the above three estimated intermediate traces with true pathlines traced from the full resolution data using RK4 method as the ground truth. For each time step we collected the distance of each estimated particle location to the true location and measured the error by averaging the collected distances from each time step of the trajectory. As shown in Figure 5.5, our method produced better estimations than the other two methods in all cases with $15\% - 50\%$ less error in average.

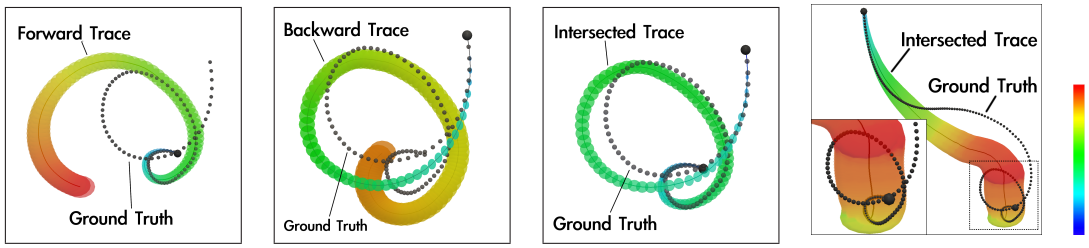


Figure 5.6: Uncertain forward (a1) and backward (a2) traces from two ends of a true pathline in the Isabel dataset within the sampling interval of 12 time steps. (a3): Our pathline refinement method by intersecting the previous forward and backward traces. (b): Our forward-backward intersected trace with the sampling interval of 24 time steps. Including for the rest of the figures, the color coding on all the estimated traces represents the estimated standard deviation (blue: lower error, and red: higher error). The 1.96 standard deviation is used as the error range.

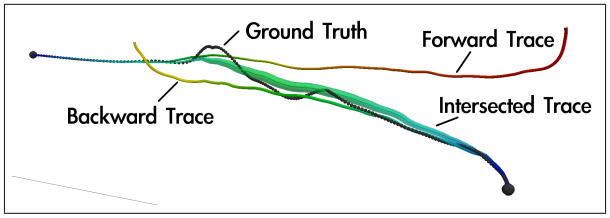


Figure 5.7: Test on the Plume dataset with the sampling interval of 24 time steps.

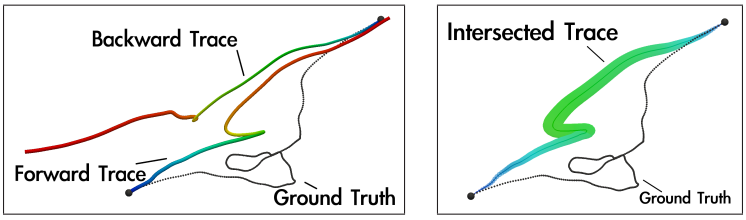


Figure 5.8: Forward and backward traces (a) and our intersected trace (b) in the Climate dataset with the sampling interval of 50 time steps.

5.6.5 Visual Analysis of Pathline Uncertainty

Finally, we visually inspect the trajectories computed from our forward-backward pathline intersection method with the estimated uncertainty measures. Since we model the uncertain particle locations in Gaussian distributions, at each integration step a pair of mean and covariance matrix is generated. To show the uncertainty of the trajectory, for each time step we plot an ellipsoid with the principal axes. To assign the magnitudes for the principal axes, we use 1.96 standard deviations in each corresponding dimension, by assuming that there is a 95% of probability for the true particle location to be enclosed by the ellipsoid. The estimated standard deviations are also used for color coding of both the traces and the ellipsoids.

Figure 5.6(a1)-(a2) show the respective particle trajectories from two end points of a flow trace going in the opposite directions. As can be seen in Figure 5.6(a1), where the dotted line is the ground truth, the forward trace initially follows the true trajectory well but then drifts off in the middle of the integration (the line becomes thicker, representing more uncertainty); in Figure 5.6(a2), the backward trace presents the vortex structure better, but also becomes less accurate as the integration proceeds. The intersection of the above two traces, as shown in Figure 5.6(a3), is closer to the ground truth than any of the two individual traces and can better preserve the vortex-shaped path. In Figure 5.6b, we show the trajectory estimation with the same initial position as Figure 5.6(a1)-(a3) in the bottom, but using the sampling interval of 24 time steps. In this case the vortex structure is barely seen in the intersected pathline, but the error range is larger, indicating higher uncertainty in the region.

Figure 5.7 shows an example of the Plume dataset, where a particle moves from left to right and becomes more turbulent in the middle. The forward trace drifts off a lot near the

end of the integration, while the backward trace stops in the middle of the true trace. The intersected trace in between is more similar to the true pathline.

Finally, Figure 5.8 shows a case in the Climate dataset using 50 time steps as the sampling interval. In this case with a larger sampling interval, both the forward and backward traces drift off quickly after the first few integration steps and thus the intersected trace in Figure 5.8b is also not closer to the true pathline. However, the uncertainty of the final trace is shown by its large width, which can help the user interpret the result without being totally misled.

5.7 Discussion

The experimental results in Section 5.6.1 show that our Bezier curve model can reduce the interpolation error effectively, compared to that using linear interpolation in the same storage. Moreover, the normality tests in Section 5.6.2 indicate the distribution of errors from Bezier curve fitting is close to be Gaussian, although the number of rejections in the normality tests increased as the sampling intervals became larger. In this case we may still provide a 95% confidence interval with the computed standard errors, as shown in Table 5.1(a). Whether the normality hypothesis and the confidence interval can be applied to larger sampling intervals for certain data sets requires more investigation and is part of our future work.

The performance of our incremental error modeling algorithm is efficient, as shown in Section 5.6.3. One limitation is that our incremental algorithm, as listed in Algorithm 1, requires caching four partial sums and the first value $y0$ for each grid point. This means the memory requirement, although remaining constant with respect to different sampling time interval n, is not negligible when the size of the flow field is large. We may alleviate this

problem by fitting a small spatial region of data values together to reduce the total number of sequences to fit, which is also part of the future work.

The extra storage cost in our framework is close to be three times of the down-sampled data, which consists of three different types of information: Bezier control point reduces the flow field interpolation error and allows the error to be modeled as Gaussian, described by the standard error of the fitting results. The particle end positions, or the flow maps, can avoid error accumulation during uncertain particle tracing. Although one can store flow maps in higher sampling rates or design an algorithm to adaptively sample pathline trajectories to preserve flow features, our error modeling scheme with forward-backward pathline intersection can further be used to refine the intermediate trajectories whenever the time-varying data are temporally down-sampled, a common practice in the field.

As shown in Section 5.6.4 and 5.6.5, our forward and backward pathline intersection algorithm can incorporate particle end positions and the error-modeled flow field to generate more accurate pathline trajectories, with point-wise error estimation indicating the uncertainty. Therefore, although the error of the reconstructed trajectories in the middle portion can still be large especially when passing through turbulent areas, our error estimation of reconstructed pathlines serves as an indication of data reliability, which provides a hint for the scientists to refine the sampling frequency for the interested temporal ranges whenever possible. Finally, although we model the uncertain pathline trajectories in Gaussian distributions, the estimated error range bounded by the 1.96 standard deviation does not always enclose the true path, especially when the integration step is too large. Table 5.1 row (b) lists the confidence level of the estimated error range, which represents the percentage of trajectory vertices whose errors to the true particle locations were within the corresponding ellipsoid range using the 1.96 standard deviation. We can see that the confidence levels

were less than 95% and would drop as the sampling interval increased. On the other hand, since our error modeling for vector fields still provides error estimation in 95% confidence intervals as shown in Table 5.1(a), we hypothesize that the smaller confidence level in the estimated particle distribution is due to the approximated Gaussian-based particle integration method. When the uncertain particle locations scatter more, the assumption of all the particles in the distribution moving together around the center of the distribution will not hold. In such a case Monte-Carlo based particle integration methods such as the work by Otto *et al.* [113] can be adopted, though with higher storage and computation costs. On the other hand, our Gaussian based pathline estimation still provides an effective qualitative measurement of errors for large sampling intervals.

5.8 Summary

In this chapter we have presented an error modeling and reduction method for large scale time-varying flow visualization, when it is not possible to process or store data from every time step due to the high computation and storage cost. We show that the errors can be modeled in Gaussian distributions when the data are fitted by Bezier curves with least squares regression, which can be incrementally computed with minimal memory footprint. We then show the applicability of the error model for pathline visualization using existing uncertain pathline computation techniques. With flow maps stored at the end of each sampling intervals, we propose a forward-backward pathline intersection method that can reduce the error of the intermediate pathlines more effectively. Finally we show that our approximated pathlines can reveal more details of the flow structure like vortices while

displaying the uncertainty within the sampling intervals. The target applications of our algorithm can be any vector field or pathline analysis tasks on down-sampled flow fields with the requirement of error indication.

There are many avenues for future work. As discussed in Section 5.7, while in this work we use a Gaussian-based particle integration method for uncertain pathline computation, we expect more accurate pathline estimation using more sophisticated algorithms. More incremental regression and error modeling methods can be explored. For example, whether we can model errors of all vector components in a joint distribution requires more investigation. While we empirically show that our datasets can be modeled by Bezier-curve functions with Gaussian errors, it is expect that some datasets may not fit our model. More suitable models could be explored with the aid of domain specific knowledge.

In the next part, a case study on large scale flow simulation is conducted, where spatial and temporal patterns of features are to be discovered. Efficient automatic feature detection and summary visualization over time steps are applied to facilitate hypothesis formulation and verification as well as data exploration in large simulation results.

Part IV: Feature Extraction and Summary Visualization for Flow Stability Analysis

Chapter 6: Visualization and Analysis of Rotating Stall for Transonic Jet Engine Simulation

To analyze large flow fields with hundreds or thousands of time steps, it is prohibitive for the scientists to load and visualize each time step of the data. Instead, it is more plausible to use visualization techniques that provides summarized overview of detected features of interest. From the temporal trends revealed from the visualization, the users are able to identify time steps and regions for further inspection. Since scientific visualization usually requires to deal with features not strictly defined, the visualization tool should allow interactive fine-tuning of the detection criteria with domain scientists involved. Thus it is crucial to devise a framework that processes the original data and extracts useful summary information to domain application for further analysis. Below we describe a framework specific for rotating stall analysis among large time-varying flow fields.

The study of turbine engine stability has been an ongoing research in aerospace engineering. In spite of the advances in technologies, the optimum performance of jet engines is still limited by the safe operation of the compressors. This limitation is mainly due to the emergence of the rotating stall when the operating range is pushed beyond the safety limit. The rotating stall is a local disturbance in the airflow through the turbine blades, which propagates among blade passages in the opposite direction of the rotor, thus increasing the loading of the blades. Although subtle initially, sustained rotating stall can eventually lead

to violent system instability causing destructive damage to the engine. Therefore, early detection of stall inception is essential to obtain sufficient response time in order to prevent engine failure.

Due to the recent advancements of parallel computing capabilities, application scientists can now perform high accuracy numerical simulations based on computational fluid dynamics (CFD) techniques in high resolutions. The results of the simulations contain rich information aiding domain scientists in understanding the complex nature of rotating stall. More specifically, researchers want to investigate: (1) the early signs of the rotating stall and (2) which local regions show such signs and at what time step? The identification of early signs of stall in the simulation data is important because the scientists can then study the simulation data around the detected time step in order to develop new stall precursors for practical use. However, since the size and complexity of the simulated unsteady data become significantly large, it poses unique challenges for the domain scientists to efficiently process or explore the simulation results in order to understand the stall phenomena. Moreover, stall analysis involves temporal data analysis, which requires appropriate visualization techniques that can effectively convey the extracted information from the time-varying data. The domain scientists also need to visually verify the occurrence of stall once it is detected through the hypothesized stall precursor analysis methods. Therefore, it is required to provide a visual analytics system that includes the domain scientists in the exploration loop, where they can synthesize and verify hypotheses through visualized interaction with the data.

In this chapter, we present a workflow for the analysis and visualization of transonic jet engine simulation data undergoing rotating stall. We first utilize existing stall precursor techniques to identify time steps and blade passages which are more susceptible to stall.

Based on a recent study in turbomachinery, stall can be identified by tracking the trajectories of vortices close to the blade tips over time [67]. We provide an efficient algorithm to extract and analyze the vortices from the simulation data. Furthermore, since rotating stall initiating in local regions of the rotor can be characterized as local disturbances, we introduce a novel method of instability analysis by detecting statistically anomalous regions among the blade passages. To effectively convey the analysis results over all blade passages and time steps, visual comparison tools using interactive juxtaposed plots are devised to reveal patterns allowing the scientists to quickly identify time periods and blade passages undergoing potential rotating stall. Once an interesting temporal and spatial region is located, a unified visualization system incorporating several spatial data visualization techniques allows the user to visually verify the suspected regions and further explore the data. Feedback from the domain scientists shows that the proposed integrated visual analytics system can substantially reduce the search space of the problem. Also, the patterns revealed by the proposed visualization methods for the stall analysis results demonstrate the potential for earlier identification of stall inception. Therefore, the contributions of this work to stall analysis and visualization are threefold:

1. Efficient algorithms for detecting stall precursors using an existing vortex analysis technique and a new statistics based method are proposed. The later is shown to be able to detect stall early in our study.

2. Comparative visualization techniques are applied to the above analysis results to facilitate visual identification of abnormalities that lead to stall.

3. A visual analytics framework for stall study is presented that integrates temporal and spatial visualization techniques for stall detection, verification, and further exploration in large datasets.

6.1 Related Work

To the best of our knowledge, this work is among the first visual analytics frameworks specific for the study of turbine engine stall. In a broader scope, visual analysis of different types of turbine flow has been studied in various purposes. Sadlo et al. [135] proposed a framework for design optimization of a hydraulic turbine, which focused on visualization of vorticies and analysis of vorticity distribution. In the analysis of power output in wind turbines, Shafii et al. [146] proposed a framework to analyze vortex-turbine interaction. Downstream vortices are visualized in vortex hulls and the degrees of intersection are color coded on the turbine blades, along with plots of related measures over time. Among works submitted to IEEE Visualization Contest 2011 on the study of a centrifugal pump turbine, Yee et al. [29] argued that detailed variations are easily missed in video presentation. To summarize vortices detected from all time series, they proposed a single-image summarization technique that visualizes vortices in ribbons.

Compared to previous research on related problems, our work focuses on the identification of abnormalities and patterns from large-scale time series data. As scientists are still exploring the criteria to detect stall inception, the current approach is through comparing flow phenomena in different blade passages and time steps. Therefore, in our problem comparative visualization of different time steps is more important than rendering the solution of a single time step. To analyze large unsteady data and provide effective summary

visualization of all passages and time steps, techniques including efficient vortex detection, anomaly analysis, as well as brushing and linking are used.

Vortex analysis in our work as well as all the above turbine-related works is an important means to understand the underlying flow phenomena. Jiang et al. [75] provided a review of different types of vortex detection methods. Depending on the choice of vortex definition, they categorized vortex detection methods into line based and region based approaches. Line based approaches detect and connect points of vortex cores to generate *vortex core lines*; region based approaches instead detect possible points associated in vortex regions. In general, line based approaches provide compact representation of vortices but region based approaches are computationally cheaper.

In our approach, λ_2-criterion [74] is used to efficiently extract vortex regions. λ_2-criterion for vortex detection has been applied in many previous studies for flow analysis purposes [88, 148, 142], as well as in the aforementioned works of turbine analysis. The method evaluates the Jocobian of flow velocities surrounding a given point, and label the point based on the eigenvalues of the matrix $[S^2 + R^2]$, where S is the rate-of-strain tensor and R is the rate-of-rotation tensor. The points that get negative second-largest eigenvalues (a.k.a. λ_2) are considered as part of a vortex region, and the more negative the value the more probable it is in a vortex. Alternative region based vortex measure Q-criterion [68] has also been widely used.

Anomaly or outlier detection has been studied in a variety of fields including statistics, machine learning and data mining with different data models and anomaly types [2, 30, 65]. In statistics, outliers are determined according to a presumed distribution, typically a normal distribution. Z-test detects outliers by examining each sample value and computing its deviation from the mean. If any of these deviations is larger than a scalar multiple,

typically 1.96x, of the standard deviation, the corresponding sample is considered as an outlier. Since the underlying standard deviation of the data is typically unknown, and using the standard deviation computed from the small number of test samples can be biased from outliers, Grubbs' test [57] is thus advantageous in determining the proper scalar multiple according to the number of samples. More advanced statistics approaches to obtain model parameters in time series include regression based methods [1] and EM algorithms [46]. In this work we use Grubbs' test for anomaly detection among the blade passages to detect the sign of instability.

Besides statistics approaches, when the data distribution is unknown, distance based methods like k-nearest neighbors are generally used, with time complexity typically higher than linear complexity [87, 128]. Alternatively, when a subset of data can be labeled to be normal or abnormal, supervised approaches are used to train the data model for anomaly detection [53].

Juxtaposition (or small multiples [14, 158]) is an effective visual design that encourages side-by-side visual comparison of multiple facets of a complex data set, without overplotting or occlusion that may occur in shared space techniques [52]. This comparative visualization has been used in many applications such as parameter space analysis, stock market trend analysis, census demographics, climatology and network analysis for various visualization plots such as bar charts, line charts, and adjacency matrices [84, 159, 99, 100].

Brushing and linking are interaction techniques commonly used to enhance scatterplot matrices [11, 22], parallel coordinates [7, 45] and other small multiple views [132]. The user brushes with an input device on one plot and the linked data points on other plots are highlighted concurrently. This effectively associates plots from different views of the data

together. In combining scientific and information visualization views, Gresh et al. [55] proposed WEAVE, which allows users to brush on 2D statistics and 3D spatial views of data. Doleisch [43] introduced SimVis, which integrates several visualization techniques including smooth brushing with fuzzy classification and time-dependent feature specification for unsteady CFD data. In our work, we extend the user interaction of brushing and linking to the search of the first occurrence of a qualified event, i.e. possible stall inception among the juxtaposed scatter plots of the temporal data.

6.2 Motivation, Background and Approach Overview

Stall inception has been actively researched for the past few decades. Although modern engine manufacturers can estimate the engine's safe operating range and impose a safety margin for unpredictable manufacturing variances, scientists and engineers still strive for narrowing the margin to obtain increased engine performance by suppressing any instability once it is detected. Therefore, to obtain sufficient response time before engine failure, early detection of stall is an important research to the related fields. Among instances of engine failure, rotating stalls have been identified as a cause of destructive flows. A rotating stall starts from localized flow separation from a turbine blade surface, which disrupts normal path of flow and forms small *stall cells* in the blade passages.Instead of following the bulk airflow direction, the stall cells propagate around the rotor at a slower rotating speed than the blades, thus inducing unbalanced forces on the turbine blades. The initial signs of stall cells can be subtle and intermittent, but sustained stall cells can quickly grow and become destructive to a compressor. To identify stall cells, it is required to observe the evolution of abnormal flow behaviors over time instead of at a single instance, thereby increasing the complexity of analyzing rotating stall. Traditionally, measures such as mass

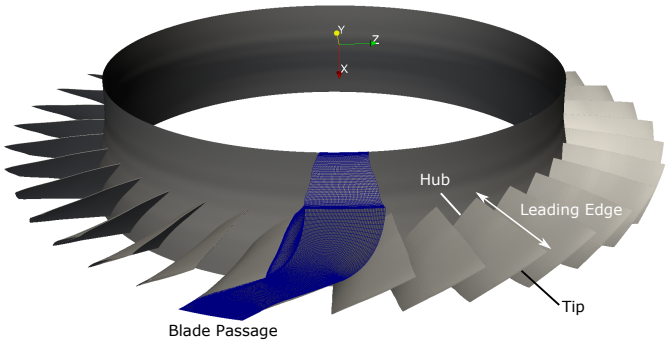
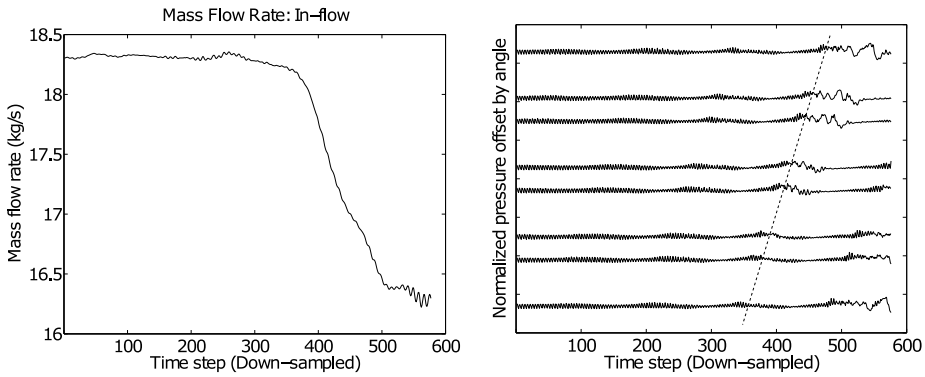


Figure 6.1: The compressor rotor in our dataset with highlighted terminologies.

flow rate and pressure probe-based analysis have been widely adopted to detect rotating stall. However, these methods still have their limitations at predicting when stall will occur, which will be described in Section 6.2.1.

With the advances of computation performance on supercomputers in the past decade, analyzing data generated from numerical simulations opens new opportunities for stall precursor research. A high-resolution CFD solver, TURBO [33], has recently been proven to be capable of capturing the major characteristics of stall [32]. However, the large size of the simulation results and the lack of visualization tools specific to the problem pose significant challenges for the scientists to study the data. Therefore, in this work, we present an analysis and visualization framework which extracts pertinent information for stall analysis from large scale simulation data.

The datasets used in our experiments are generated from simulations of a NASA single-stage compressor [130], which is a representative model of transonic axial compressor. The geometry of the rotor consists of 36 blade passages, shown in Figure 6.1. The solutions are computed by TURBO, which solves the Navier-Stokes equations in the full-annulus model. TURBO generates high-resolution unsteady flow fields for all passages, which are



(a) Mass flow rate plot. The rapid drop indicates the occurrence of stall.

(b) Pressure probe plot. The dotted line indicates the propagation of disturbance across different probes.

Figure 6.2: Traditional stall detection methods and the plots.

then analyzed in our framework to enhance the understanding of rotating stall. Several simulations were performed with different operating conditions that will lead to stable or stall conditions for the purpose of studying the transitions between them. In the following we first use a simulation dataset of a stall condition to explain and formulate our stall detection and analysis method. In the result section, other conditions are tested to verify and demonstrate the effectiveness of our method. Below we provide an overview of the existing stall analysis techniques and then describe the motivations behind the choices of precursors in this work.

6.2.1 Traditional Stall Detection Methods

In this section, we discuss two typical techniques for analyzing stall in engines along with their limitations.

Mass flow rate is a standard measure to verify the existence of stall in numerical experiments [63, 101, 162], which reflects the mass of air that flows through the compressor

per unit time. The mass flow rate, denoted by $\dot{m}$, is defined as:

$$\dot{m} = \rho \mathbf{v} \cdot \mathbf{A}, \tag{6.1}$$

where ρ represents the density and $\mathbf{v}$ is the flow velocity. $\mathbf{A}$ is the area vector of a surface, which is the cross section of the inlet or exit of the compressor stage for stall analysis. It has been shown that when a system undergoes a rotating stall, the mass flow rate drops significantly. A plot of mass flow rate against time in Figure 6.2a shows a characteristic curve of the mass flow rate drop under stall conditions. It can be observed that from around time step 250 onwards the mass flow rate starts to decrease, and it drops rapidly after around time step 350. By observing the drop of the mass flow rate, scientists can explicitly identify the occurrence of stall. Although the mass flow rate plot is commonly used in stall analysis, it does not serve as a stall precursor because mass flow rate drops after stall occurs. Moreover, since mass flow rate is a global measure, it does not provide detailed phenomena or specific regions of stall in data for further analysis.

Use of **pressure probes** in stall detection has been extensively studied and adopted widely in practice [107, 42, 41]. The probes are located circumferentially within the engine casing to record the pressure oscillation over time. Since the pressure in stall cells is distinct from that of their surrounding areas, pressure probes can capture these spikes in value once a stall cell passes a probe location. Figure 6.2b shows the pressure probe plot used by the scientists to analyze stall. In the figure, eight curves represent eight pressure probes in different angular locations, where each curve is a plot of the pressure values over time. It can be seen that the oscillation of each curve changes drastically when stall occurs after time step 350 and the abnormality shifts to the next probe as the stall cells rotate. By correlating the occurrence of stall cells detected by pressure probes located at different circumferential locations, the scientists can estimate their rotation speed. When this speed

drops close to half of the rotor speed, stall will occur imminently. In spite of numerous successful stall analysis approaches based on pressure probing, it is nontrivial to find the proper probe locations and the number of probes to use, which can be different for different compressor models. Moreover, the current pressure probe plotting can only convey a few probes per chart. When the number of curves representing the pressure probes increases, the chart becomes cluttered and difficult to visually identify oscillation changes.

6.2.2 Problem Statement

High resolution data generated from CFD simulations provide rich information that allows scientists to conduct detailed studies of engine stall. However, it also poses significant challenges to analyze the large amount of data that are produced. Since stall analysis requires observing the temporal evolution of flow behaviors, our goal is to help the domain scientists visually detect early stall inception from its temporal patterns and verify the detected regions with their domain knowledge. With a well-designed visual analytics framework to help identify time steps and regions of possible stall inception, the exploration time and effort for stall analysis can be substantially reduced.

The visual analytics system will meet the following requirements set by the scientists:

1. Analyze the dataset based on selected stall detection methods.

2. Visualize the stall analysis results with sufficient information for the scientists to identify salient time steps and regions for possible stall inception.

3. Once an important spatiotemporal region is selected, render the corresponding data for visual verification of the phenomena and allow a more detailed exploration of the region.

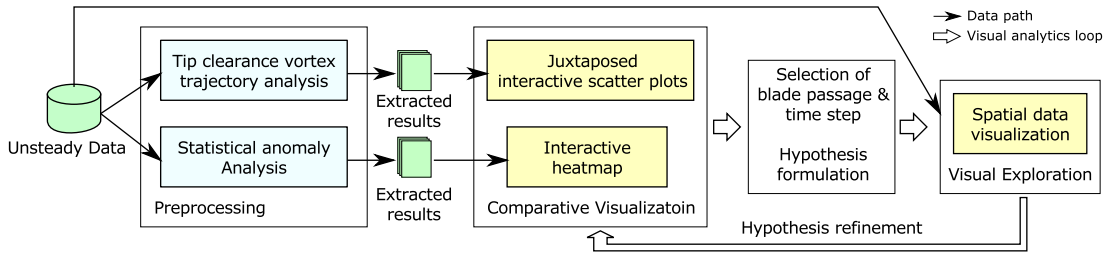


Figure 6.3: Overview of the visual analytics framework.

Next we provide an overview of our visual-analytic framework.

6.2.3 Overview of The Visual Analytics Workflow

Figure 6.3 presents a schematic view of the stall analysis and exploration framework. The proposed framework processes large scale unsteady data and extracts information using two stall analysis methods based on domain knowledge. The first method uses a stall precursor proposed by Hoying et al. [67], which detects stall based on the orientation of the tip clearance vortex in each blade passage. We automate the detection using vortex extraction techniques on the velocity fields and calculate the orientation of each vortex structure. In the second method, inspired by the use of pressure probes to detect abnormal value changes, we adopt a statistical method to detect anomalies among the blade passages. These analysis methods measure the tendency of stall inception for all time steps of data in the preprocessing stage, which will be described in Section 6.3.

Once the related information to stall analysis is extracted, the results are presented in interactive visualizations revealing the temporal trends. By observing the temporal and spatial patterns in the visualization, the scientists are able to determine time steps and blade passages of possible stall inception for further analysis. After a salient spatiotemporal region is identified, the system then allows the scientists to explore the raw data at the

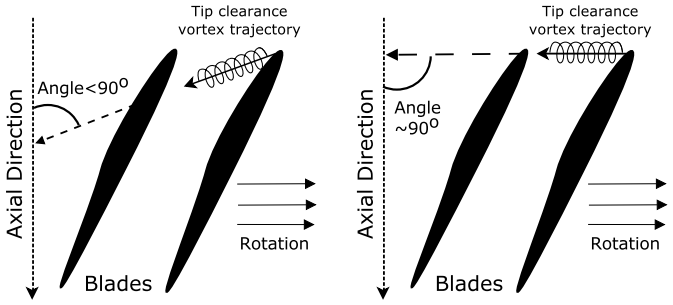
specific time step in customized visualization, where they can verify the hypothesis of stall inception and explore the dataset to design new precursors. The visual comparison and exploration framework will be described in Section 6.4.

6.3 Detailed Descriptions of Stall Analysis Methods

Although the detection of stall has been under research for several decades, the actual phenomena that cause the inception of stall are still not completely understood. There is a growing need of novel stall precursors which can detect early signs of rotating stall with sufficient accuracy. In this section, we describe our techniques to compute the tip clearance vortex angles for stall detection and provide a statistical analysis method for finding anomalies in the data. Since the dataset is large in size, the analysis procedure should be efficient and only scan the data once. Accordingly, the stall analysis methods are designed with a minimum number of parameters, which extract sufficient but compact information to allow the user to adjust parameters in the interactive visualization. The extracted information will help the scientists efficiently identify time steps and blade passages of stall inception.

6.3.1 Stall Analysis Using Tip Clearance Vortex Trajectory

According to previous research, a well structured vortex, called a tip clearance vortex, usually forms on a blade tip under stable operating conditions. Normally these vortices do not inhibit flow through the blade passages. However, as the compressor nears a rotating stall, the tip clearance vortices start to move in a specific direction and eventually breaking down. Hoying et al. [67] hypothesized that when the angle of the tip clearance vortex core becomes perpendicular to the axial direction, the condition is approaching stall. A conceptual diagram depicting this phenomenon is presented in Figure 6.4. Figure 6.4a shows the state of tip clearance vortex in a healthy condition; Figure 6.4b presents the idea



(a) Tip vortex in a healthy condition.

(b) Tip vortex in a stall condition.

Figure 6.4: A stall precursor based on the trajectory of the tip clearance vortex. Here the side view of the rotor is shown. The major flow direction follows the axial direction.

of a near stall condition when the angle between tip clearance vortex and axial direction is close to 90°. Hence, by observing the trend of the change of these angles, potential blade passages which show early signals of a rotating stall can be identified.

Although stall can be detected through analyzing the tip clearance vortices, automatic extraction and tracking of these vortices is not trivial. Since the location and orientation of the tip clearance vortex can vary from blade to blade, it is difficult to identify the tip clearance vortex from the other vortices in a blade passage automatically. We use prior knowledge from the scientists that the tip clearance vortex can be seen close to the blade tip in a passage and is usually the largest. Therefore the search area of vortices can be reduced to one third of the entire passage region toward the tip as the *target search region*. For each detected vortex we measure and store the length and the angle made with the axial direction, which is described as follows.

In order to efficiently detect vortices from all passages and time steps, region based vortex detection method λ_2-criterion [74] is used. As introduced in Section 6.1, negative λ_2 measures indicate the corresponding point is in a vortex region, with the more negative

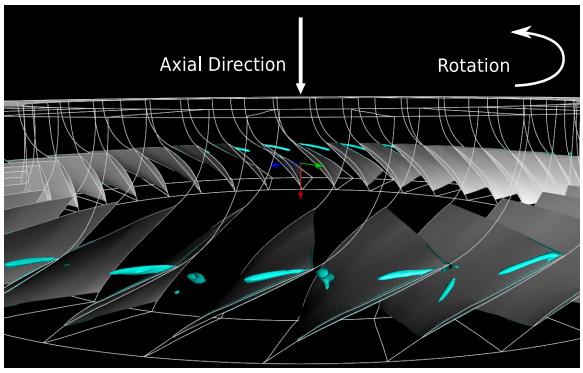


Figure 6.5: Vortices found by the λ_2 criterion. The largest vortex close to the blade tip in each passage is the tip clearance vortex.

the value the higher possibility it is in a vortex. We allow the domain expert to examine and tune the threshold by rendering the extracted vortices from selected time steps in the user interface. This chosen threshold value is then fixed for vortex extraction processes during the remainder of the experiments. Since the tip clearance vortex is generally larger than other vortices in the same passage in normal conditions and even in near-stall conditions, we use a higher (more negative) threshold to filter out small or ill-structured vortices. This allows us to compute its angle to the axial direction with less noise, before it breaks down at stall. Figure 6.5 shows a sample isosurface rendering on λ_2 criterion where the structures of the tip clearance vortices are visible.

Since there may still exist several vortices in the search region, before measuring the orientation of each vortex region, we employ a connected component labeling algorithm [165] to label the detected points based on their spatial connectivity, where two neighboring points in the grids are considered connected. The algorithm scans through each point and labels the points by a group ID, where connected points are given the same ID. By doing

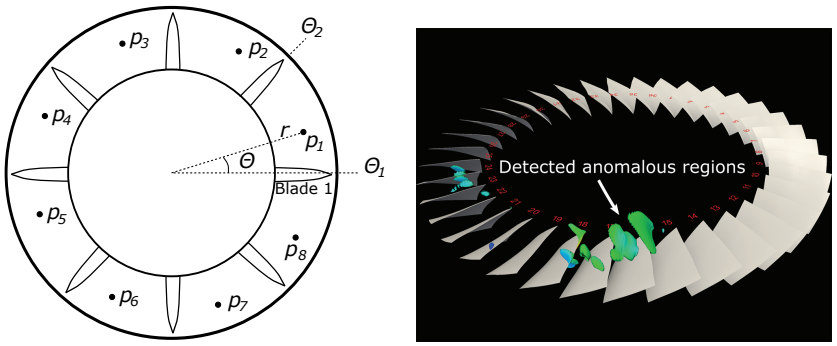
so, points in different vortices are grouped separately, from which the respective lengths of the vortex cores and their angles to the axial direction can be extracted.

Since the tip clearance vortices are usually straight as shown in Figure 6.5, the principal component analysis (PCA) is used to extract the principal direction of each connected point group. From the result of PCA, we select the direction of the principal eigenvector $\vec{v}$ as the direction of the current vortex core and measure its angle Θ with axial direction $\vec{A_{dir}}$. The vortex length is approximated by two times the principal eigenvalue.

The presented algorithm is efficient because both the λ_2 criterion and the connected component labeling only require point-wise scans through the data points. In order to identify stall inception with higher confidence, the scientists would like to see the increasing trend of the tip clearance vortices to be perpendicular to the axial direction. Therefore we store the computed lengths and angles of vortices detected in each blade passage. Combined with our overview visualization showing the temporal changes of vortex angles in all the blade passages, the expert can more easily detect early signs of rotating stall, which will be described in Section 6.4.1.1.

6.3.2 Statistical Anomaly Detection for Rotating Stall Analysis

While the tip clearance vortex analysis provides one aspect of stall detection, we devise a statistics-based anomaly detection method to reveal more information from the simulation results. This method is inspired by how stall is commonly analyzed and detected from pressure probe readings, through observing irregularity of the pressure values. In a healthy condition, it is expected that the recorded pressure values by each probe exhibit periodicity every time a blade passage passes a probe location. This is because the blades are axisymmetrically similar, so ideally the flow behaviors among the blade passages should also be



(a) (b)

Figure 6.6: (a): Illustration of points extracted by the anomaly detection algorithm. All the points $p_1, ...p_8$ in the eight passages of this simplified example have the same radius r and relative angle θ. (b): Anomalous regions of pressure detected in a time step.

axisymmetric. When stall cells appear within the passages, this symmetry breaks down and thus pressure probes are able to detect the stall cells since abnormal pressure oscillations are observed in the readings. Therefore, we extract sets of axisymmetric points from the passages and detect point-wise anomalies for each point set.

In order to efficiently detect point-wise anomalies from large unlabeled datasets, we adopt Grubbs' outlier test [57] for each point set in symmetry, assuming these points form a normal distribution in a normal condition. Grubbs' test is widely used for its computational simplicity [152]. It is more robust than the simpler Z-test when the standard deviation of the data distribution is unknown, as mentioned in Section 6.1. Note that Grubbs' test is not applicable if too many outliers exist in the sampled data. However, since our goal is to detect early signs of stall inception, it can be expected that outliers are less observed in relatively normal conditions.

To detect anomalies in the flow field, for each time step we first group points from each passage that are at the same relative position into a point set. The number of point sets

is equivalent to the number of grid points in a passage. Figure 6.6a illustrates a simplified rotor to facilitate understanding, in which one point from each passage has the same relative position and will therefore be grouped together.

Formally, in a compressor of m blade passages, a point in the k-th blade passage, $1 \leq k \leq m$, can be defined as

$$p_k(x, r, \theta) = \langle x, r, \theta + \theta_k \rangle \tag{6.2}$$

in the cylindrical coordinate, where r is the radius from the center and θ is the relative angular offset from the referencing angle θ_k of the k-th passage. x represents the axial position toward turbine downflow, which is the direction pointing into the paper.

For each set of points $p_1 \cdots p_m$ at the same relative position in each passage, we form a value set $\mathbf{V}$:

$$\mathbf{V} = \{f(p_k(x, r, \theta)) | k = \{1, 2, ..., m\}\} \tag{6.3}$$

where $f(p)$ is the field value (pressure) at point p. Then for each point set we apply Grubbs' test by first computing the mean $\mu(\mathbf{V})$ and standard deviation $\sigma(\mathbf{V})$ of these values. Based on Grubbs' test, a value $\mathbf{V}_k$ is an outlier of set $\mathbf{V}$ if

$$\frac{|\mathbf{V}_k - \mu(\mathbf{V})|}{\sigma(\mathbf{V})} > \frac{m-1}{\sqrt{m}} \sqrt{\frac{t^2_{\alpha/(2m),m-2}}{m-2+t^2_{\alpha/(2m),m-2}}}. \tag{6.4}$$

Here $t_{a,\nu}$ is the critical value of the t-distribution with the alpha value $a = \alpha/(2m)$ and degrees of freedom $\nu = m-2$. For our case of anomaly detection, the significance level $\alpha = 0.01$ is used. Thus with the sample size $m = 36$ from 36 blade passages, the right hand side of the above inequality is around 3.3296.

We repeat Grubbs' test for all point sets in the dataset and extract anomalous points in terms of pressure values. Figure 6.6b presents one instance of the rotor where detected

anomalous regions based on the pressure values are highlighted. Although the anomaly detection is performed individually for each point set, in the experiments we observe contiguous detected abnormal points in local neighborhood, which form larger anomalous regions in some passages.

Before further experiments were conducted to confirm the validity of our hypotheses, we showed the detected anomalous regions in selected time steps and the expert agreed that our approach has potential to capture the development of stall cells. However, since a detected anomalous region alone does not provide much evidence, the expert asked to see the temporal behavior of the anomalous regions for further verification. Therefore, we devised a compact visualization of detected anomalies from all time steps. From patterns revealed in the visualization, the expert confirmed that the detected regions are related to stall cells, as will be discussed in Section 6.4.1.2.

6.4 Visual Analytics Interface Design for Stall Analysis

In this section we present how the visual analytics system integrates appropriate visualization techniques for the scientists to easily identify spatial and temporal patterns from the stall analysis results. Since both of the above stall analysis methods are hypothesized to identify stall, in order to verify the hypotheses the scientists rely on (1) inspecting trends of the detected results to see how they evolve over time and how long the detected events persist, and (2) studying the flow data surrounding the detected regions to observe the temporal trend of the related variables. To achieve this systematically, in Section 6.4.1 we first introduce our comparative visualization interface that presents the trends of the precursor results through multiple juxtaposed charts. Once an important region is identified through

the visualization, our visual exploration system allows verification and in-depth analysis of the detected results through spatial visualization techniques, as described in Section 6.4.2.

6.4.1 Comparative Visualization of Stall Analysis Results

In order to effectively help scientists identify potential stall inception in a large simulation dataset which stores thousands of time steps, we devise a comparative visualization tool to convey the results of stall precursor analyses over all time steps. Our comparative visualization tool satisfies the following design goals:

1. Can identify time steps and blade passages that are more susceptible to stall. This is done by comparing the measures of the stall analysis results.

2. Can reveal the temporal pattern of the analysis results. This is important because a single detected instance may not be significant enough to determine stall.

3. Can facilitate the detection of early signs of stall inception from the analysis results.

4. We describe the comparative visualization system for each of the stall analysis method separately in the following sections.

6.4.1.1 Comparative Visualization of Tip Vortex Angles

According to Hoying et al. [67], stall can be detected when tip clearance vortices become perpendicular to the axial direction. Although one can search for vortices around 90 degrees by a single scan of the vortex analysis result, visualization is necessary for the following two reasons: (1) Since a single instance may not be significant enough to determine stall, it is required to observe the increasing trend of the vortex angle toward 90 degrees over a longer time span. (2) With more information presented, the scientists are able to formulate new stall precursors with different criterion. Therefore, to effectively convey

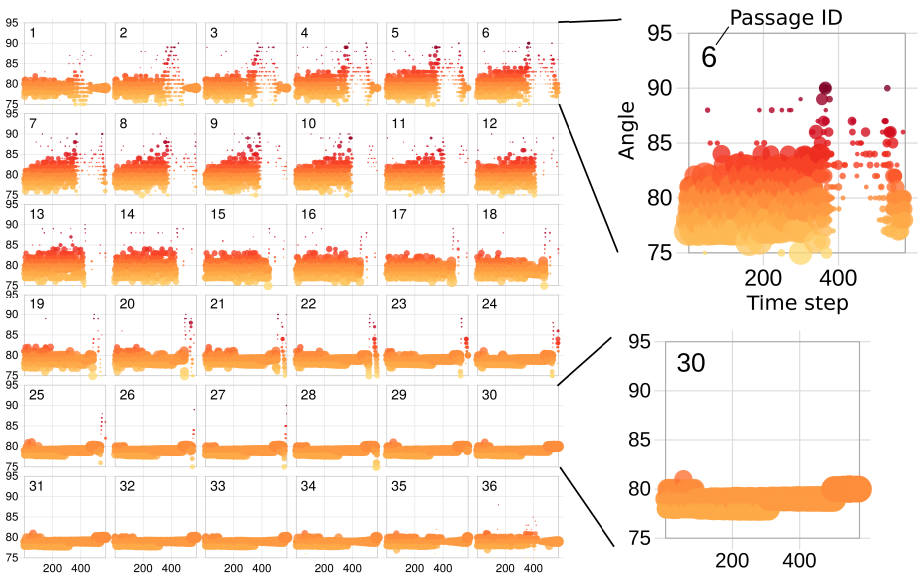
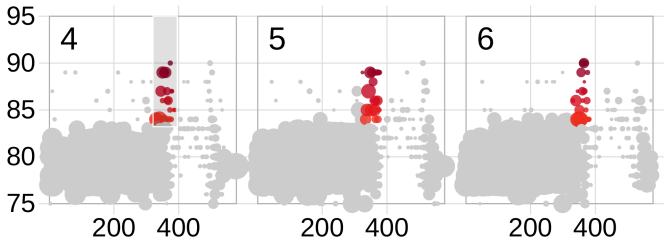


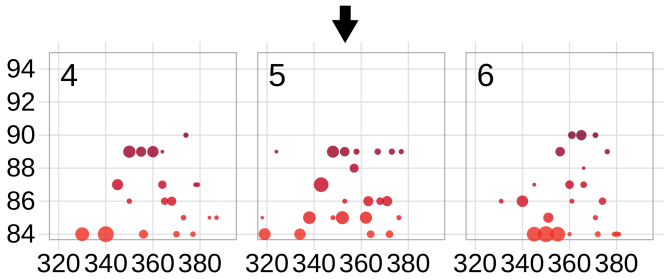
Figure 6.7: The juxtaposed visualization of tip clearance vortex angles.

and compare the vortex analysis results for all time steps and all passages, we create an *interactive juxtaposed visualization* where the vortex angles for each passage are plotted against time and put side-by-side for contrast.

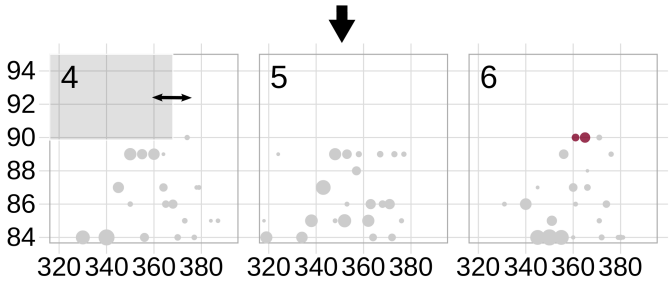
To provide an overview of the tip clearance angles over all time steps and passages, it is desired that the visualization simultaneously presents attributes of each vortex including the angle, length and the ID of the passage it belongs to. Figure 6.7 shows the visualization of the vortex analysis result for the dataset in the stall condition previously presented in Section 6.2.1. In the juxtaposed visualization, each small chart corresponds to a blade passage where the horizontal axis represents time steps and the vertical axis represents vortex angle. Inside each chart, each extracted vortex is represented as a dot whose size is proportional to the corresponding vortex length and the color is modulated by the closeness of the angle to 90 degrees. If the color is close to dark red, the vortex is approaching 90 degrees.



(a) Brushing time steps of interest where the vortex angles approach 90°.



(b) Zooming in from the brushed area.



(c) Highlighting angles around 90° in the earliest time steps. The size of the highlighted region can be easily adjusted to find the first qualifying instance among all passages.

Figure 6.8: Illustration of the interface to find the first time step and passage where the tip clearance vortex first approaches 90°. Selected passages are shown for demonstration purpose.

Figure 6.7 shows an increasing trend of vortex angles toward 90 degrees, which is more easily seen in passages 2 to 13. Around time step 400, these passages show small and sparse dots, indicating the vortices break down. Concurrently, in passages 25 to 35 more stable and longer vortices (larger circles) are displayed with angles around 80 degrees.

To further investigate the time steps and passages where the vortex angles first approach 90 degrees, the interaction tool allows the user to brush (Figure 6.8a) and zoom into the angles and time steps of interest (Figure 6.8b). To look for the vortex that first becomes 90 degrees, the user can first select an area including degree 90 and then adjust the width of this area to change the selected time interval. As the user expands the area width, dots falling within the selected time and angle interval will be highlighted in all passages (Figure 6.8c). In this way, it is easy to contrast vortex angles and time steps across all passages and find the first occurrence of vortex within the desired angle range, which can be chosen other than 90 degrees to further explore the characteristics of the tip clearance vortices.

Figure 6.8c shows the highlighted dots representing the vortex angles first near 90 degrees, which starts from time step 361 in passage 6. This time step coincides with the rapid decrease of the mass flow rate shown in Figure 6.2a, but the tip clearance vortex analysis method provides a more specific time and location for investigating stall phenomena. The user can then select this passage and time step in the visual exploration tool for verification and further investigation, which will be described in Section 6.4.2.

6.4.1.2 Comparative Visualization of Statistical Anomaly

As discussed in Section 6.3.2, the anomaly detection algorithm attempts to identify abnormal regions, which can be stall cells that eventually cause a rotating stall. To verify this, it is required to see how these regions evolve over time. In general, the behavior of the anomalous regions can be classified into three broad types: (1) sporadically appearing

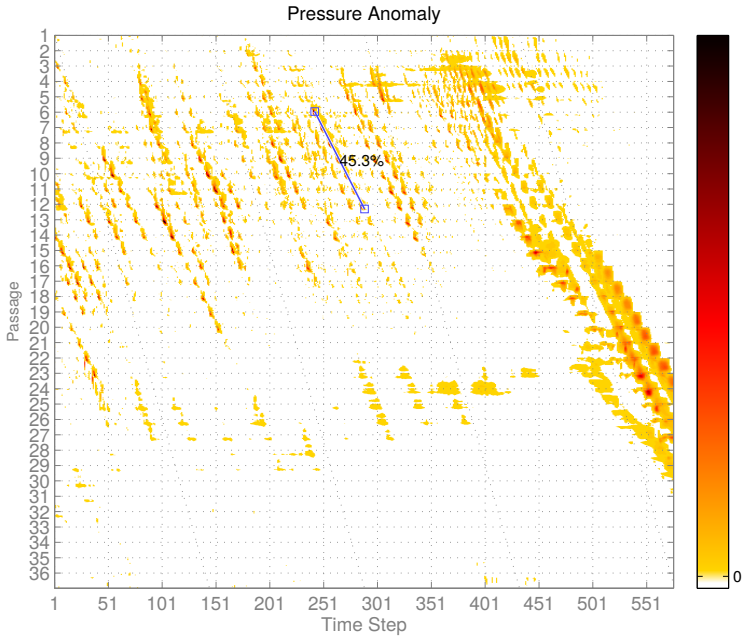


Figure 6.9: Anomaly analysis chart of pressure values.

and dissipating over a short time period, (2) remaining at a similar physical position over certain time steps, (3) moving along with the blades over certain time steps. In the study of stall inception, the scientists are more interested in detecting types (2) and (3) where the anomalous regions persist over a longer time span. Once they are detected, it is necessary to know the rotation speed of these hypothesized stall cells among passages.

To meet the above requirements, we utilize a 2D heatmap that encapsulates both the spatial and temporal patterns of the detected anomalous regions, as shown in Figure 6.9. Since we are more interested in how the anomalous regions move across passages over time and less in their movement within a passage, in the plot, the horizontal axis represents the time step and the vertical axis represents the angular position labeled by the passage

ID. We use a colormap for the count of anomalous points detected in different radii(r) and axial positions(x) but in the same angular position, with darker color for a larger count.

Through the inspection of Figure 6.9, two major trends in the propagation of anomalous regions can be observed: (1) Horizontal direction, meaning the detected anomalies stay in the same blade passages, i.e., rotate at the same speed as the rotor over time; (2) slant direction, meaning these regions move across the blade passages, or rotate at a different speed than the rotor speed. It can also be seen that before time step 350, the chart shows fragmented but aligned slant patterns, as highlighted by the blue line. This means the anomalous region intermittently moves across passages, which pattern cannot be easily captured if we only visualize a single time step of the anomalous regions at a time. After time step 350, persistently growing anomalous regions crossing almost all passages are formed. The time step 350 found for this dataset is consistent with the drop-off of the mass flow rate shown in Figure 6.2a. This result suggests that anomaly analysis can be used to identify stall. Therefore, the heatmap representation achieves the goal to reveal patterns for the scientists to track the behaviors of anomalous regions for signs of possible stall cells. With this representation, the scientists can also easily find interesting time steps and passages for further stall study.

While the chart of pressure anomaly provides the expert with clues of potential stall cells and where they are located, it is desired to calculate the rotational speed of the anomalous region to further confirm a stall cell has been detected. A line widget is used to find the velocity by adjusting the end points of an event, as shown by the blue line in Figure 6.9. The number besides the line shows the fractional rotation speed of a stall cell to the rotor speed, where a horizontal line indicates 100% of the rotor speed. The measurement of

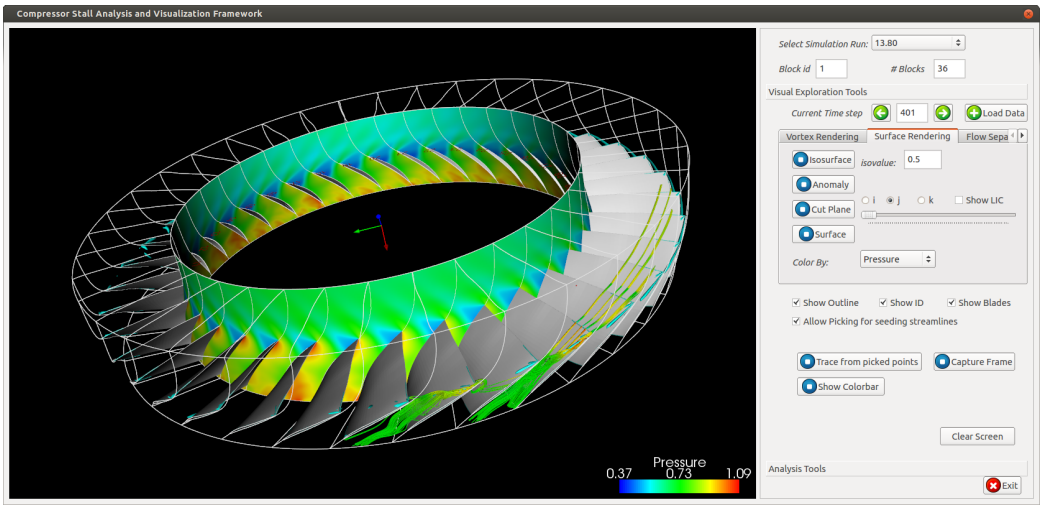


Figure 6.10: The visual exploration system. The interface includes stall analysis tools and an example rendering of streamlines, isosurfaces and cut planes.

the rotation speed reads less than 50% of the rotor speed, which, according to the expert, indicates the near-stall condition.

6.4.2 Integrated Visual Exploration System

After the important time steps and blade passages are identified from the visual comparison tools presented above, our visual exploration system is designed in a way that the scientists can verify the detection results and further explore the original dataset. Figure 6.10 shows the interface that integrates the above analysis and rendering techniques for stall analysis. The proposed system is built using VTK, which is a widely used open-source visualization library [144]. The visualization system serves two purposes required by the expert: (1) to evaluate and verify the stall analysis results, and (2) to further explore the dataset around the selected regions for enhanced understanding.

To verify that the detected passage has a tip clearance vortex angle perpendicular to the axial direction, isosurfaces of λ_2 criterion in the selected time step are used to show the orientation and size of the vortices. Since streamlines and pressure values are commonly used to affirm the existence of a vortex structure, the exploration system can display streamline traces surrounding the tip clearance vortices and cut planes of pressure values. To explore anomalous regions, the scientists look for abnormal changes of variable values including pressure, density, and entropy etc. Regions of the statistical anomalies based on the method described in Section 6.3.2 are rendered to provide immediate indication of abnormal regions in the selected time step. The visualization system allows the scientists to explore the data based on hypothesis formation and visual verification. In order to reach to a definitive conclusion, the system allows expert to go back and forth between the abstract plot-based visualization and data domain exploration so that they can adjust/refine their hypotheses based on visual feedbacks from the data. The proposed framework involves scientists in the exploration loop which leverages free exploration of data and knowledge discovery.

6.5 Results and Expert Feedback

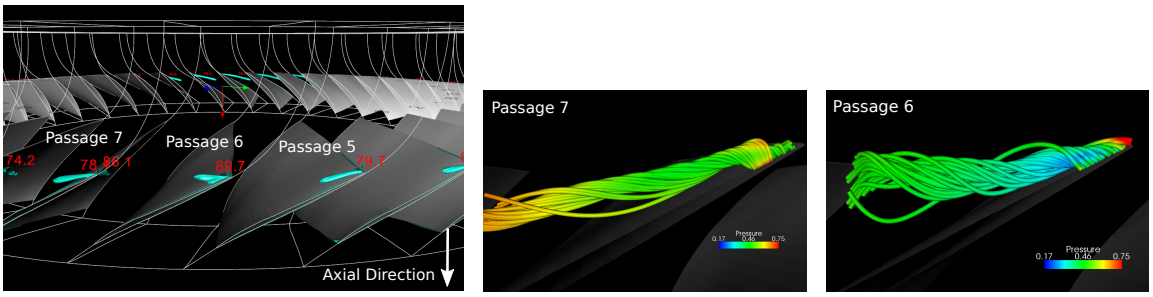
In this section we demonstrate the effectiveness of our visual analytics framework in a rotating stall analysis. In order to verify the two proposed stall analysis methods are able to detect stall inception, experiments in different operating conditions were performed. The analyses were then shown by our visualization interface and examined by the domain scientists. The collaboration involved an expert with more than 28 years of simulation and analysis experience on transonic turbine stages and two aerospace engineering PhD students. The expert feedback was collected from regular bi-weekly meetings in the past one

and a half years, which allowed us to gradually improve the framework and the experimental design.

The compressor rotor used in experiment consists of 36 blade passages, each of which is stored in curvilinear grids of dimensions $151 \times 71 \times 56$, forming a multiblock dataset. Thus each time step requires storage of 690 MB in PLOT3D format consisting of 21.6 million grid points. The simulation divides a full turbine revolution into 3600 iterations, where the flow data is output and stored every 25 simulation iterations. In our experiments, each simulation runs at least 4 revolutions to observe whether stall occurs, which generates at least 400 GB of data with 576 time steps per dataset. Note that the time steps presented here are in the unit of the stored time steps at the sampling rate of 25 simulation iterations. In the following, we first show that our visual analytics system can help detect the inception of stall, using the previously presented dataset in the stall condition. Then we demonstrate the applicability of the stall analysis system in different operating conditions to show its potential use for stall analysis and exploration. Finally, the performance measures are listed to show the efficiency of the stall analysis algorithms.

6.5.1 Verification of the Proposed Stall Detection Methods

In this section we will verify the proposed method for stall analysis by showing the consistency of analysis results to what the domain expert expects. The mass flow rate plot is a standard measure to stall analysis and thus the basis to verify our analysis results. For the dataset in the stall condition presented earlier, as shown in Figure 6.2a, the mass flow rate drops rapidly starting from around time step 350, indicating the occurrence of stall. This time step is consistent to our analysis results. Through brushing the juxtaposed charts of the tip clearance angles (Figure 6.8), as discussed in Section 6.4.1.1, we find passage



(a) Vortex regions detected by λ_2 criterion, labeled with angles to the axial direction.

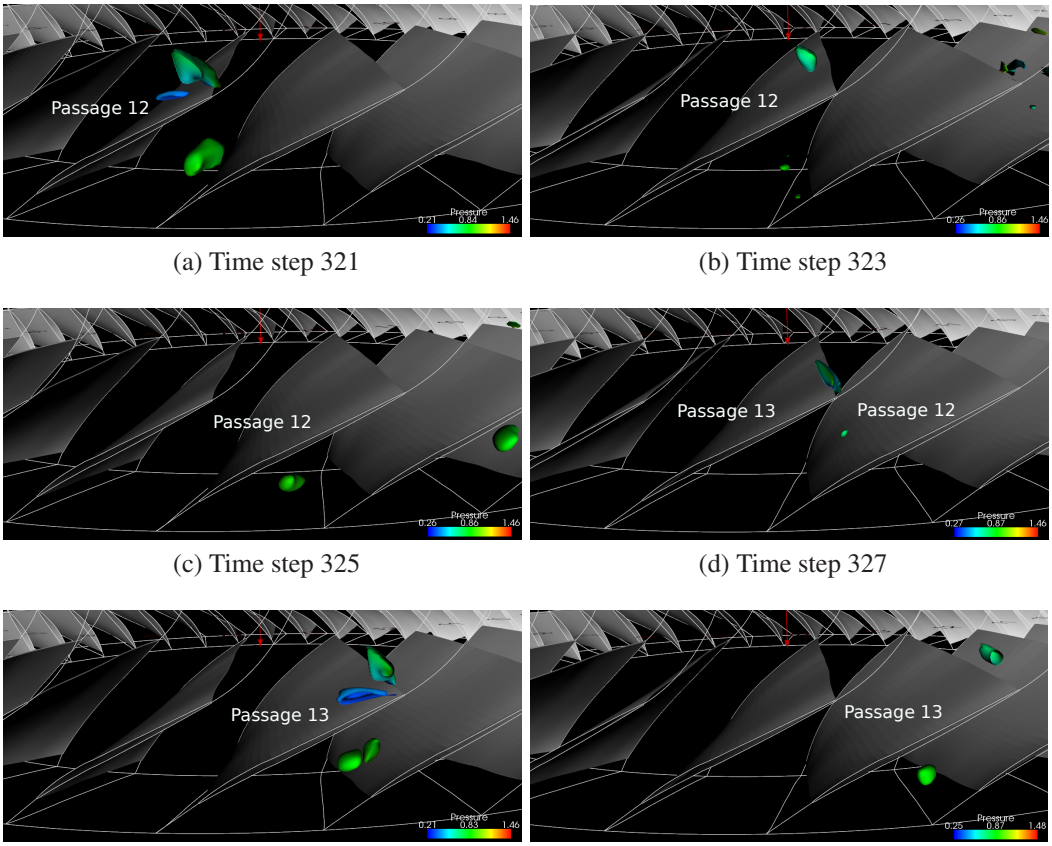
(b) Streamlines showing the tip clearance vortex of passage 7, colored in pressure.

(c) Streamlines showing the tip clearance vortex of passage 6, colored in pressure.

Figure 6.11: Tip clearance vortices detected at time step 361. Passage 6 shows the vortex structure more perpendicular to the axial direction.

6 at time step 361 showing the vortex angle first approaching 90 degrees. Similarly, Section 6.4.1.2 describes that in the visualization of anomaly analysis (Figure 6.9), a persistent large anomalous pattern can be seen starting from around time step 350 in passages 3-6. Therefore both visual analysis tools allow scientists to detect and verify the occurrence of stall as well as pinpoint the time steps and passages where it is occurring. In addition to the comparison to the mass flow rate, it is necessary for the expert to inspect the raw data by visualizing the stall phenomena in the detected regions. The visual exploration system was used to examine and verify the detected regions with the expert's domain knowledge, as described in the following paragraphs.

Verification of the tip clearance vortex analysis. To inspect the structure of the tip clearance vortex found at time step 361 and verify its perpendicularity to the axial direction, our visual exploration system was used to extract isosurfaces of λ_2 measures. In Figure 6.11a, the isosurface representing the tip clearance vortex region in passage 6 shows a distinct vortex structure than those in other passages. The orientation also appears perpendicular to the



(a) Time step 321 (b) Time step 323

(c) Time step 325 (d) Time step 327

(e) Time step 329 (f) Time step 331

Figure 6.12: Anomalous regions of pressure detected within time steps 321-331. These regions grow, shrink, disappear and show up again in the adjacent passage.

axial direction. To observe the flow surrounding the detected vortex, Figure 6.11c depicts the streamlines colored by pressure value. From the vortical structure of the streamlines, it is evident that the selected region has a strong vortical flow. This demonstration shows the efficacy of the proposed juxtaposed plots to visually detect the time steps and passages having tip clearance vortex angles close to 90 degrees as a precursor of stall.

Verification of the anomaly analysis. Next, the expert was presented the heatmap visualization revealing the temporal patterns of detected anomalous regions, as previously

shown in Figure 6.9. As discussed in Section 6.4.1.2, it is hypothesized that the fragmented but aligned slant patterns in the early time steps are stall cells. Since according to the expert, stall cells generally appear in areas close to blade tips, in order to verify this the system was used to render anomalous regions in selected time steps. Figure 6.12 shows a fixed window wherein compressor blades move to the right. The surfaces in color are the detected anomalous regions. This visualization confirms that the detected regions are close to the tip region.

Expert Feedback. By inspecting the evolution of the anomalous regions over time, an interesting phenomenon caught the attention of the domain expert. Figure 6.12a-b shows the anomalous regions detected in passage 12, but disappear in a later time step (Figure 6.12c). Later, anomalous regions are detected again in passage 13 (Figure 6.12d-e). This corresponds to the fragmented slant pattern seen in the anomaly analysis chart, where the fragmented pattern indicates the intermittency of the anomalies and the slant direction indicates their propagation across passages. By seeing this the expert realized that the detected anomalous regions conform to the behavior of stall cells. A brief explanation is as the following.

When stall cells occur within a passage, it forms a blockage and changes the angle of attack (AOA) of the flow. While AOA decreases in the current passage and stabilizes the flow, it increases in the neighboring passages, which in turn causes a stall there. In this way the stall cell transports from one passage to another. Therefore it is evident that the detected anomalous regions are related to stall cells.

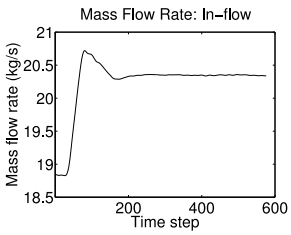
With the evidence that the fragmented slant patterns shown in the anomaly analysis chart are stall cells, the expert hypothesized that the anomalous regions detected in the

earliest time steps of this dataset are in fact stall cells. This hypothesis can be justified because the initial condition of the given dataset was already close to stall. In order to verify whether the anomaly analysis method can capture the transition from a stable to unstable condition and detect early stall inception, we ran more simulations with different throttle settings as below.

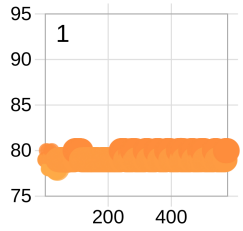
6.5.2 Stall Analysis of Simulations from Different Throttle Settings

In order to see the applicability of the proposed stall analysis and visualization techniques in different stability conditions, we analyzed results of simulations in different operating conditions suggested by the expert. The operating condition can be adjusted by a parameter modeling the rates of air passing through the exit throttle of the compressor in *corrected mass flow rates*. It is known that setting the corrected mass flow rate low will cause stall, while setting it high will stabilize the flow. Based on previous studies conducted on the compressor currently under investigation, stall will occur when the corrected mass flow rate is set to 13.80 kg/s, corresponding to the condition used to produce the data discussed above. To examine the capability of our stall analysis system for data generated in different conditions, we present two different throttle settings leading to different conditions: One is a known stable condition with the throttle setting 16.00 kg/s ; the other is an initially unknown condition with the setting 14.20 kg/s, which turned out to be a stall condition after a long simulation run.

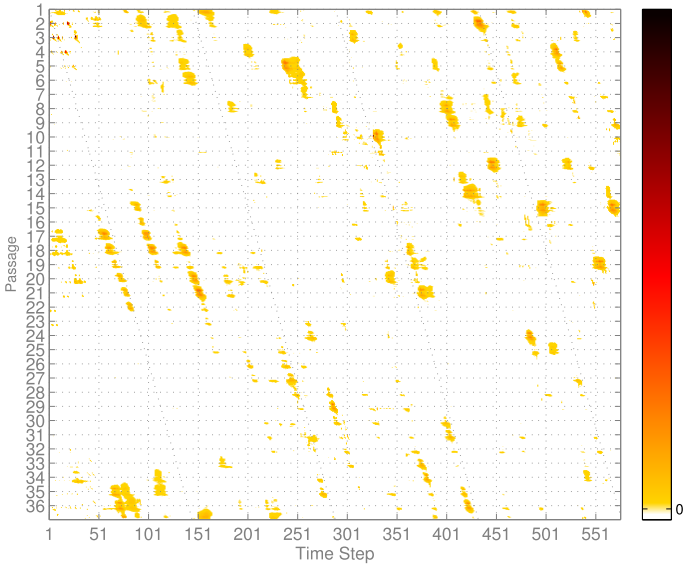
Stable Condition (Corrected Mass Flow Rate 16.00 kg/s) In this case the dataset was generated with a throttle setting of 16.00 kg/s from an initially less stable condition. In Figure 6.13a, the mass flow rate increases and stabilizes after time step 200. The visualization of the tip clearance vortices also shows no sign of stall, where the vortex angles maintain



(a) Mass flow rate plot.



(b) Tip clearance vortex analysis chart.



(c) Anomaly analysis chart.

Figure 6.13: Results of a stable condition using the corrected mass flow rate of 16.00 kg/s. Figure (b) shows the plot of a passage representing for all other passages in the similar behavior.

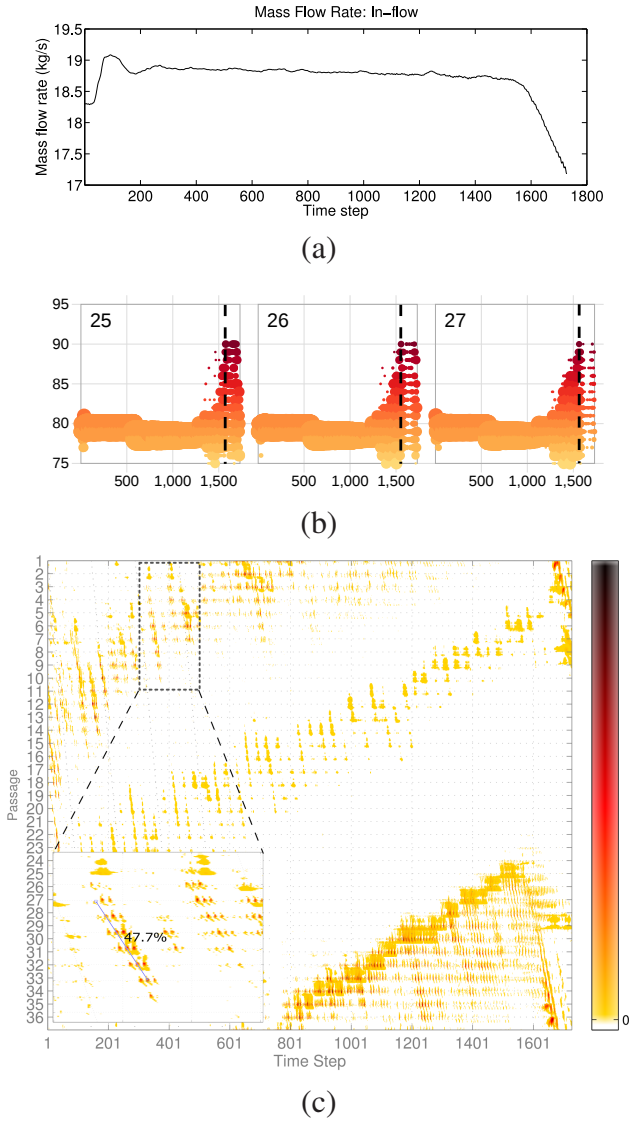


Figure 6.14: Results of a stall condition using corrected mass flow rate 14.20 kg/s. (a): Mass flow rate plot. (b): Tip clearance angle analysis charts (Showing passages 25-27). (c): Anomaly analysis chart. The lower left figure shows the dashed rectangular region within passages 1-10 and time steps 300-500.

around 80 degrees. Figure 6.13b plots the vortex angle degrees of a passage over time, which represents all other passages in the similar stable condition.

The anomaly analysis chart in Figure 6.13c shows anomalous regions more scattered after time step 200. This scattered pattern is significantly different than that in the previous stall condition (Figure 6.9), which provides further evidence that our anomaly analysis and visualization can distinguish stable and unstable conditions.

Stall Condition (Corrected Mass Flow Rate 14.20 kg/s) For this throttle setting, a corrected mass flow rate of 14.20 kg/s is used, which is in between the known stall condition (13.80 kg/s) and stable condition (16.00 kg/s). Before it turned out to be a stall condition, it was difficult to tell whether this setting would lead to stall. As shown in Figure 6.14a, the mass flow rate first increases from a less stable condition and becomes stable with small oscillations appearing after time step 300. A stable mass flow rate is maintained until approximately time step 600, where the mass flow rate begins a slow decline. Finally stall occurs at around time step 1500, or almost 10.5 rotor revolutions. The data size of the entire simulation output is 1TB.

In the tip clearance vortex analysis shown in Figure 6.14b, the time step when the vortex angles first become 90 degrees is also around 1500, occurring in passages 25-27. The similar time steps and passages of stall occurrence can also be observed in the anomaly analysis chart in Figure 6.14c. Additionally, the fragmented slant patterns continuously appear and propagate from the beginning of the simulation. These patterns are similar to that previously shown in the stall condition with the throttle setting at 13.80 kg/s (Figure 6.9), which correspond to the occurrence of stall cells. This experiment shows that instabilities,

though not clearly picked up by mass flow rate and tip vortex analysis, are already present in earlier time steps and captured by our anomaly analysis method.

6.5.3 Algorithm Performance and Storage Size

To measure the performance of our stall analysis algorithms, a Linux desktop machine was used with a 7200 RPM disk, 16GB RAM and the Intel Core i7-2600 CPU supporting up to 8 concurrent threads. The algorithms were implemented with OpenMP parallelization, where the vortex angle computation was parallelized over passages and the anomaly analysis computation was parallelized over point sets. It took around two hours to complete processing a dataset of 576 time steps (four rotor revolutions). Within each time step, the computation of the λ_2 criterion on the curvilinear grids, the vortex angles and the anomaly analysis took 7.6, 0.63 and 0.42 seconds in average, respectively. The average I/O time was 1.9 seconds to load the PLOT3D files. The extracted information used for the comparative visualization, output in text formats, was less than 0.1 MB per time step. The small file size was a result of vortex analysis only storing the lengths and angles per detected vortex and the anomaly analysis only storing the count of anomalous points per angular position, as described in Section 6.4.1.2.

6.6 Discussion of the experimental results

The above results on various throttle settings demonstrate the capability of our visual analytics system to detect stall inception. In all tests, the identified time steps of stall are consistent with the standard mass flow rate analysis results. More importantly, the anomaly analysis and visualization, as shown in Figure 6.9, displays pronounced trends at a much earlier time than the other two methods (vortex angle and mass flow rate analysis). According to the expert this shows that the proposed statistics-based anomaly analysis

method is able to capture instability in the flow field, and can be a useful tool to predict the ultimate occurrence of stall. Early stall detection can also avoid wasting unnecessary simulation cycles in order to determine whether a throttle setting leads to stall. This is especially useful in fine-tuning the throttle setting to find the critical stall point.

This study also demonstrates the successful application of visualization to feature detection, when the characteristics of the feature is to be discovered and the detection criteria are to be searched for. The intermittent behavior of the stall cells makes automatic detection and tracking of these stall cells difficult, since most object tracking algorithms require a certain type of spatial coherence in adjacent time frames [23, 119]. Instead, by aligning the detected regions of all time steps, our visualization of anomaly can reveal implicit patterns easy to pick up by human eyes. Since the proposed anomaly analysis approach uses a general statistics method, we believe it can be applied to stability analysis in other rotating devices as well as other data types where symmetry is present.

In terms of limitation, our current visual design does not distinguish the different locations of anomalous regions in a given passage (e.g., tip or hub area) and requires the user to visualize the original data. We plan to add more visual design elements into the heatmap, such as more colors and interaction. In addition, in order to extract tip clearance vortices, the current method requires the user input of λ_2 threshold when a new model is given. Visualization tools to aid the threshold search like predicates [138, 139] and more sophisticated extraction criteria can be applied in the future work. To obtain more precise tip clearance vortex angle, line-based vortex detection methods can be used to explicitly obtain vortex core lines. Finally, the visual exploration tool can be further improved by integrating brushing and linking techniques on 3D views and the selected tip clearance vorticies and anomalous regions from the corresponding visualizations.

6.7 Summary

We have presented a visual analytics system with integrated visualization techniques to help scientists study the complex phenomena of rotating stall. The system tightly couples domain knowledge into stall analysis algorithms, which efficiently extract essential information from large simulation data. The analysis results are depicted through compact and comprehensible visualization techniques using interactive plots. Our integrated visualization interface allows the scientists to verify the hypothesized stall inception regions and further explore the detected regions in detail. Experimental results and positive feedback from domain scientists provide a strong indication of the efficacy and potential of our system to help future stall analysis and precursor development.

Chapter 7: Visual Comparison of Anomaly Detection Methods for Flow Stability Analysis

The previous chapter shows the potential capability of the proposed statistical anomaly analysis to detect stall cells in early formation of rotating stall. In order to further validate and enhance the detection results for stall discovery, it is required to design alternative stall detection methods from different observations and hypotheses. In this chapter, we adopt two general data analysis techniques, a temporal signal processing and a distribution-based method, to detect possible stall cell locations for fluid flow simulation data.

For the first method, according to earlier research on physical experiments, stall cells can be identified by observing abnormal oscillation of pressure signals over time, where Fourier frequency analysis can be used. However, most existing stall detection methods only apply on physical experiments with few number of pressure probe signals, where the pressure probe locations need to be well-selected by the domain scientists. For the simulation data, we devise an efficient flow instability detection and visualization method on millions of temporal signals, by a temporal anomaly detection method with Fourier analysis.

Second, according to our domain expert, since stall cells are regional phenomenon where abnormal flow behaviors present, detecting anomalies in regions to find stall cells is more suitable than that per point. Therefore, we extend the anomaly detection method

for axisymmetric points presented in the previous chapter to a region based method which finds anomalous regions among axisymmetric regions. A region-based anomaly detection method using local distributions is developed.

Finally, to facilitate validation and enhance inference among the results from above methods based on different hypotheses, we present an interactive interface that allows the user to easily compare and analyze the detection results. Our experimental results on simulation data in different stability conditions show the coherence and additional insights among the proposed methods.

The rest of this chapter is organized as follows: Section 7.1 discusses related work of temporal analysis. Section 7.2 and Section 7.3 presents temporal and region-based spatial anomaly detection methods, respectively. In Section 7.4, the comparative visualization interface is provided, and the experimental results are discussed in Section 7.5. The chapter is summarized in Section 7.6.

7.1 Related Work

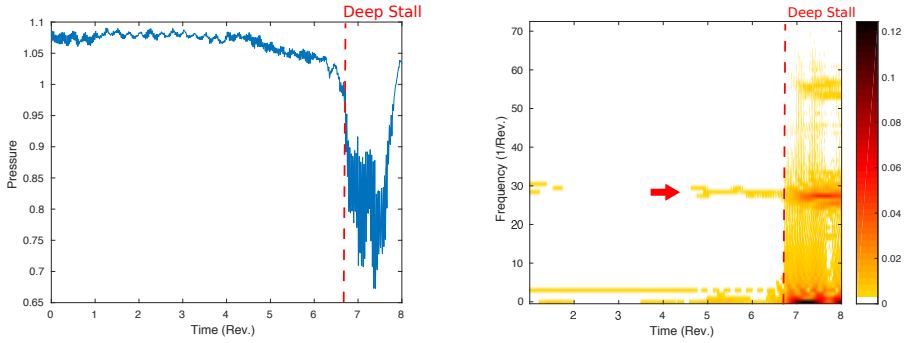
Temporal analysis for stall detection has been extensively studied for the past decades. In physical experiments, pressure probes are positioned circumferentially within the engine casing to capture pressure readings over time. The collected time series are typically analyzed in the frequency domain using Fourier transform to identify periodical occurrence of flow instability that may indicate the passing of stall cells through the probes [40, 107, 115, 157]. Recently, Pan et al. [115] used windowed FFT plots to visually identify stall cell frequencies and correlate these frequencies with different readings from different probing locations.

Instead of Fourier analysis, it is also assumed that the pressure probe signals generally present periodicity over time, where the period is one rotor revolution. In other words, the curve of the signal in one revolution should be similar to that in the next revolution. Tahara et al. [151] computed the correlation of signals between consecutive revolutions and triggered stall warning when the correlation drops to some extent. Li et al. [94] used a similar idea of signal periodicity but counted the frequencies of uncorrelated signals within a statistical window, where stall warning occurs if higher frequency counts are detected.

Works on general anomaly detection has been discussed in Section 6.1. For **temporal anomaly analysis and visualization,** many problems have been researched including traffic analysis [27, 161], social media [28, 154] and medicine [91]. Lakhina et al. [89] used PCA (Principle Component Analysis) to analyze network traffics over time and detected high responses in residual vectors as anomalies. Based on frequency analysis, Janicke et al. [72] visualized wavelet transformed data to study the El Nao phenomenon. Valdivia et al. [161] detected traffic anomaly by a variation of Fourier transform over graphs.

7.2 Temporal Anomaly Detection

Previous researches including the windowed FFT and correlation approaches, as described in Section 7.1, have been shown effective in analyzing pressure probe signals. However, it is not trivial to apply these methods to simulation data. This is due to that the amount of data collected from simulation is much larger than that from a few probes in physical experiments. A solution to efficiently analyze and effectively visualize the stall detection results from all possible probing locations is desired. More importantly, since most full-annulus turbine flow simulators including TURBO [33] compute field values per passage in the relative frame following the rotor rotation, it is computationally cheaper to



(a) Signal (b) Windowed FFT

Figure 7.1: Example of windowed FFT plot for a signal from a stalling simulation. (a): A drastic drop in the 7^{th} revolution is seen. (b): Windowed FFT plot. The color represents the magnitude of the corresponding frequency (Y-coordinate) for one revolution time interval before the corresponding time step (X-coordinate). The drastic change of the frequency response can be easily observed after the 7^{th} revolution. In addition, a moderate response around frequency 30 can be observed within revolutions 4-7.

also analyze signals over time in the relative frame. However, most of the rotating stall analysis methods are designed where the pressure values are collected from fixed probe locations (i.e., absolute frame) in physical experiments. Therefore, the known frequencies expected in the absolute frame, like the blade-passing frequency, may not be observed in the relative frame.

We observed that since stall cells are local instabilities that can propagate through turbine passages, the oscillation frequencies of pressure readings can change when a stall cell enters a probe location. By applying the windowed FFT analysis, this change of frequencies can be captured, even in the relative frame. Figure 7.1 demonstrates a time series of pressure values collected in the same location of a rotor passage (i.e., relative frame) over time, under a simulation towards deep stall. As can be seen, the frequency response in Figure 7.1b changes drastically when deep stall happens after revolution 7. Moreover, a

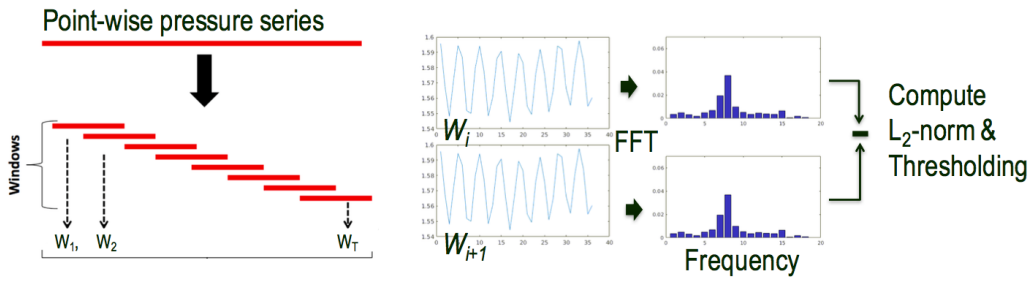


Figure 7.2: Illustration of temporal anomaly detection based on windowed FFT.

unique frequency response is seen before deep stall happens, which is due to the presence of a stall cell. Since the domain scientists are more interested in identifying stall cells as the precursor of deep stall, this change of frequency response before deep stall is important to detect, which we regard as temporal anomaly.

Figure 7.1 shows frequency responses for only a single signal of a grid point. To extract temporal anomalies for time series from all possible grid points, we regard the Fourier frequency space as the feature space and compare consecutive feature vectors to detect drastic changes. Specifically, for each time series of pressure values collected from the same grid point, the feature vector computed from windowed FFT of a time window is compared with the previous time window. If these two feature vectors are sufficiently different, we regard the current time step and the corresponding grid point as a temporal anomaly, where a stall cell is expected to be present at the corresponding time and locations.

Figure 7.2 illustrates the process of our temporal anomaly detection. A pressure series is collected from a grid point of the pressure field over time and the feature vectors computed by FFT are compared for consecutive windows. To compare two feature vectors, the L_2-norm distance is applied. Note that the advantage of comparing consecutive time series in the frequency domain is that any non-harmonic signal input can be easily identified since

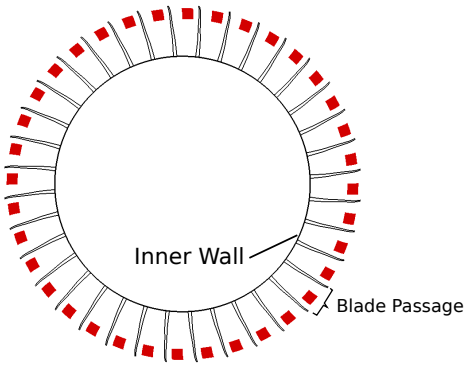


Figure 7.3: A 2D illustration of axisymmetric regions (red boxes) to be grouped for region-based spatial anomaly analysis. This figure shows a cut plane of the turbine, where the major flow passes vertically into the paper. Note that the red region size is exaggerated for visibility.

they present distinct frequency responses. Although a single anomalous event may not be sufficient to indicate stall cell passing, we rely on a statistical summary approach as will be described in Section 7.4.

7.3 Region-based Spatial Anomaly Detection

In this section, we present the region-based spatial anomaly detection technique, which extends the anomaly detection approach on axisymmetric points presented in Chapter 6 for stall analysis. Since this method only identifies anomalies within a single time step, the term *spatial anomaly* is to be distinguished from temporal anomaly described in the previous section.

As discussed in Chapter 6, the flow behavior in a turbine which consists of many blade passages presents axisymmetry among the passages. It is therefore hypothesized that in a normal condition, the regional flow behavior in the same location relative to each blade passage will be similar. More specifically, the field value distribution in these axisymmetric

regions are similar. As illustrated in Figure 7.3, we divide the passages into small spatial blocks and group axisymmetric blocks for anomaly analysis. The goal is to identify blocks which have distinct statistical behavior than the other blocks in the corresponding group.

The problem to tackle here is essentially to identify one or more distributions that are different from the other distributions, assuming most distributions in the group are similar. This assumption is valid in stall analysis since we focus on finding the inception of flow instability, when most of the passages still behave similarly in a relatively stable condition.

Similar to the previous per-point method, our approach first estimates the *expected* distribution as a basis for all the distributions in the group to compare with. In other words, if any of the distribution in the group is sufficiently different from the *expected* distribution, that distribution is regarded as an outlier, and the corresponding block in the physical space is reported anomalous. We define the *expected* distribution by the per-bin average of histograms of the distributions in the group, i.e.,

$$H^{expected}(bin_k) = \frac{1}{N}\sum_{i=1}^{N} H^i(bin_k) \tag{7.1}$$

where $H^{expected}(bin_k)$ represents the frequency of the *expected* distribution at bin k, for any bin in the histogram representing the distribution. $H^i(bin_k)$ is the frequency at bin k in the i^{th} distribution of the group of N distributions. Equivalently, this is to compute the distribution of sample values from all distributions in the group. The benefit is that the major value distribution is attained and the effect of outlier values to the *expected* distribution, if present, is reduced.

After the *expected* distribution is formed for a group, we compare it with each distribution in the group using the Earth Mover's Distance (EMD). The EMD is a distance measure defined by the minimal ground transport effort to match two distribution shapes. EMD has

been widely used in pattern matching and image analysis [80, 140], as well as to compare probability distributions in uncertain data [134]. Besides its robustness against noise [97], EMD's measuring of ground transportation is able to capture an outlier if its value deviates from the majority of the distribution, which is particularly desired in our anomaly detection study. To compute the EMD for 1D distributions, we use the *match distance* as the ground distance [133] since the EMD can thus be efficiently computed by the absolute difference between the CDFs of the distributions [164]:

$$EMD(X,Y) = \int_{-\infty}^{\infty} |F_X(x) - F_Y(x)|\, dx \tag{7.2}$$

Here $F_X(x)$ is the cumulative distribution function (CDF) of the distribution X at x.

After the EMD is computed for all distributions, a user specified fixed threshold is used to extract blocks that have high EMD measures. We apply the above method for all groups of blocks per time step and mark the blocks identified as spatial anomalies. As will be shown in the Result section, using EMD as the distance measure for anomaly detection generates results with less noise than other bin-to-bin comparison distance measures.

In the previous two section, we present two anomaly detection methods that provide quantitative assessments of possible flow instability. With a fixed thresholding for each detection method, the anomalies are extracted. In the next section, we propose an interactive interface that allows visual comparison and pattern discovery between these two detection results.

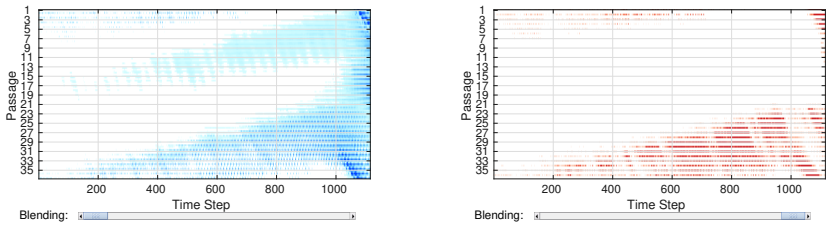
7.4 Comparative Visualization of Anomaly Detection Results

To reveal temporal patterns of the detected anomalous regions among more than a thousand time steps and allow comparison between different detection results, we extend the time-passage heatmap chart as presented in the previous chapter. Briefly, we overlay two

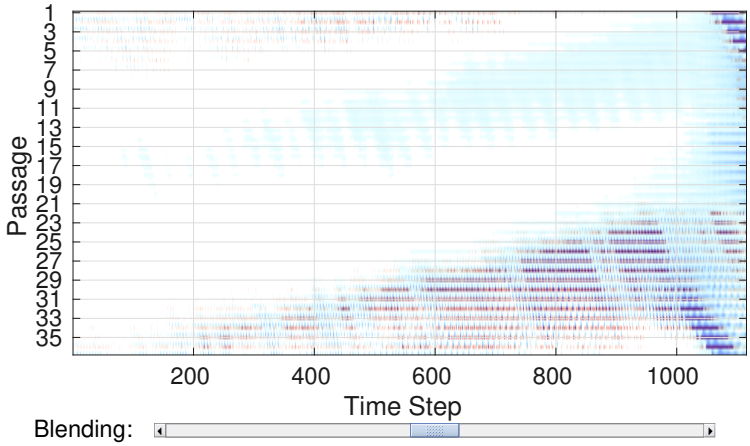
heatmap charts from different detection methods with a well-chosen color scheme to provide effective pattern comparison. The detail of each step is described as follows.

The anomaly chart. According to the domain expert, as also discussed in the previous chapter, stall cells are organized flow anomalies that present the following properties: (1) They can exist and propagate across passages for a longer time, and (2) they can grow in size to hinder the normal airflow through the compressor. In other words, in the overview visualization the expert is interested in how the detected flow anomalies propagate among passages instead of how it moves inside a passage. Therefore, a 2D heat map is used to visualize the anomaly detection results, where the Y axis on the chart represents the passage number and the X axis represents the time step, as shown in Figure 7.4a and 7.4b. Each point on the chart is color coded by the size of the detected anomalous region in the corresponding passage and time. As a result, points with higher counts and connected with other points across several time steps are more salient. This design is similar to the time-passage heatmap described in the previous chapter.

Superimposition. In addition to visualizing the anomaly pattern of a single variable, the expert is interested in how the anomalies detected from different variables are correlated and whether the combination of them generates a more confident indication of stall. Therefore, comparing and contrasting anomalies from different variables are required. We provide superimposition views [73] which composite two anomaly charts into one chart with transparency. Compared to juxtaposition views which place visualizations side-by-side, superimposition does not split the user's attention into different parts of the screen, and it also makes the comparison intuitive [73]. Since superimposition may cause visual clutter in complex co-occurrence regions, we provide an interaction tool to overcome this problem as discussed below.



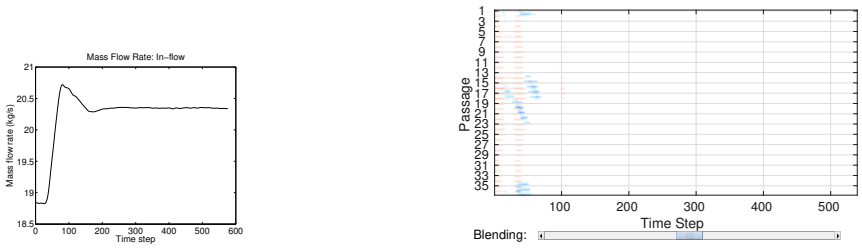
(a) Plot of regional spatial anomaly. (b) Plot of temporal anomaly.



(c) Comparative plot of spatial and temporal anomaly. The co-occurrence regions are highlighted in blended purple color.

Figure 7.4: Spatial and temporal anomaly visualization for a simulation of a stall condition (with the setting of the corrected mass flow rate = 14.20 kg/s).

Alpha blending and interaction. To superimpose two selected charts, we overlay them with alpha blending, where the blending coefficient α is adjusted by the user. As shown in Figure 7.4, a slider in the bottom is added to adjust α, where moving the slider to an end shows the anomalies of a single variable in the chart. Therefore, the user can move the slider to contrast different anomaly detection results in position with both contents. Since it is hypothesized that the co-occurrence of anomaly from different detection criteria can indicate the existence of stall cells with more confidence, it is required to highlight these regions on the chart. Therefore, we increase the saturation of co-occurrence regions on the chart when the slider is closer to the middle position, as shown in Figure 7.4c. This



(a) Mass flow rate plot. (b) Comparative plot of temporal anomaly (in red) and regional anomaly using EMD (in blue).

Figure 7.5: The mass flow rate and comparative anomaly plot for the stable condition with the corrected mass flow rate = 16.0 Kg/s.

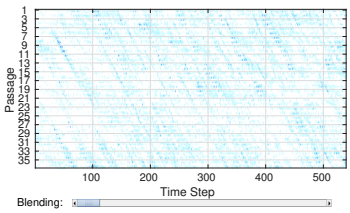
enhances the focus on co-occurrence regions and keeps the regions of individual occurrence in context.

7.5 Results

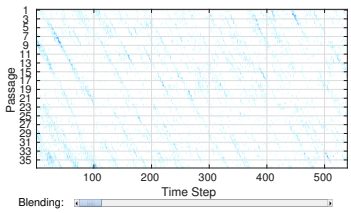
To show the efficacy of the proposed anomaly detection methods for stall analysis, we compare our methods to alternative approaches and with expected results under different simulation conditions. Two known flow conditions, a stable condition and a stall condition, with respective throttle parameter settings (the corrected mass flow rate) were tested. Spatial and temporal anomaly analysis methods were conducted for both conditions, with fixed threshold settings. The FFT window size in temporal anomaly analysis was empirically set to 36 time steps, equivalently $\frac{1}{4}$ of a rotor revolution.

7.5.1 Stall Detection Methods Comparison for a Stable Condition (the Corrected Mass Flow Rate Setting = 16.0 Kg/s)

To demonstrate the robustness of our stall detection methods that they do not falsely classify stable flow as anomaly, we perform our anomaly analysis on a simulation of a



(a) Regional anomaly plot using distribution mean.

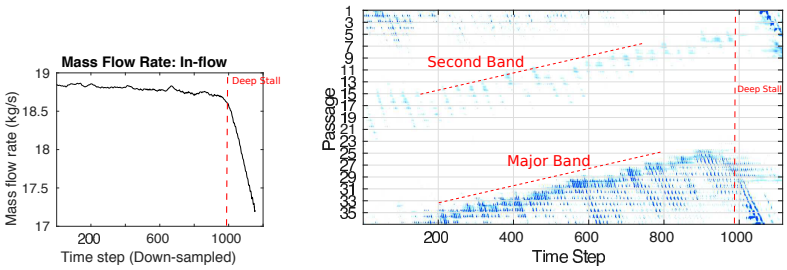


(b) Regional anomaly plot using KL divergence.

Figure 7.6: Anomaly plots using alternative distribution distance measures (corrected mass flow rate = 16.0 Kg/s).

stable condition. The mass flow rate plot in Figure 7.5a shows an increase of the mass flow in the first 100 time steps and then the curve becomes stable. This indicates a transition from a less-stable condition to a stable condition due to a change of the throttle parameter setting. As shown in Figure 7.5b, the plots of both regional and temporal anomaly analysis reveal this stability condition change, where anomaly is only seen within the first 100 time steps.

To demonstrate that the choice of the Earth Mover's distance (as shown in Figure 7.5b) performs better than other measures for identifying spatial anomalies, two alternative approaches were tested. The first is the *distribution-mean* approach, which reduces the detection of distribution-based anomaly to point-based anomaly, by computing the distribution average for each region. As a result, the Grubbs' method as described in the previous chapter can be applied. However, this simple approach drops much information per region and introduces noise as shown in Figure 7.6a. Alternatively, we could apply a different distance measure when comparing each distribution with the *expected* distribution described in Section 7.3. The KL (Kullback–Leibler) divergence, a bin-to-bin distribution distance measure was tested. As shown in Figure 7.6b, the noise present in the later time steps is reduced but



(a) Mass flow rate plot (b) Point-based spatial anomaly plot

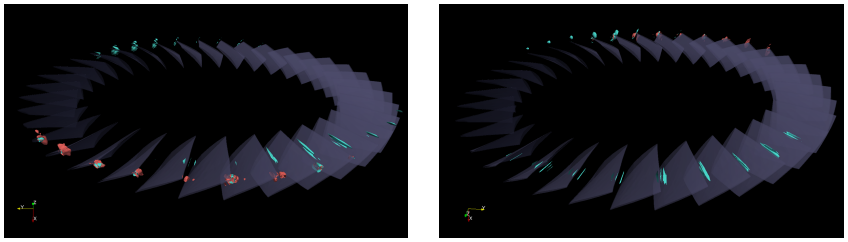
Figure 7.7: Plots using existing stall detection methods for the stall condition with the corrected mass flow rate = 14.2Kg/s.

overall it still detects undesired anomalies. These tests show that our region-based spatial anomaly detection method with the use of EMD is a well-chosen approach.

7.5.2 Stall Detection Methods Comparison for a Stall Condition (the Corrected Mass Flow Rate Setting = 14.2 Kg/s)

To further verify that our proposed stall cell detection methods are able to detect early inception of stall cells, we analyze a simulation of a stall condition. The comparative plot of the proposed region-based spatial anomaly and temporal anomaly for this stall condition has been shown previously in Figure 7.4c. For comparison, Figure 7.7a shows the plots of the mass flow curve over time and Figure 7.7b shows point-based spatial anomaly, both described in the previous chapter.

Figure 7.7a shows a sharp drop on the curve, indicating that the deep stall occurs around time step 1000. Figure 7.7b as a previous method also shows a distinct anomaly pattern after time step 1000. Moreover, a major band of anomaly after passage 24 is seen, meaning that the flow instability occurs in these passages, which also indicates possible stall cell inception. In addition, there is a second band of anomaly before passage 21 indicating possible anomaly stall cells.



(a) View from the first band. (b) View from the second band.

Figure 7.8: Spatial rendering of region-based spatial anomaly and temporal anomaly from different view. Blue region: Spatial anomaly. Red region: Temporal anomaly.

The comparative plot as shown in Figure 7.4c, exhibits similar anomaly trends but provides additional information. In the figure, the dark purple color indicating more confident stall cell locations is only seen in the passages of the first band (beyond passage 24). Meanwhile, in the passages of the second band (before passage 21), only spatial anomaly (in blue) is detected. Therefore, the comparative plot infers that stall cells are more likely located in the passages of the major band. The domain scientists can next focus on anomalous regions in spatial rendering from the corresponding time steps and passages in the major anomaly band.

7.5.3 Spatial Rendering of Stall Regions and Expert Feedback

To see the spatial location of the detected anomalies from different methods, Figure 7.8 renders the detected regions from a near-stall time step 800 in the stall condition as used in Section 7.5.2. Figure 7.8a shows the view from passages corresponding the first band, where both spatial and temporal anomalies occur. It is of great interest to see that in areas close to blade tips, both spatial and temporal anomaly regions overlap each other. This

indicates that these regions are highly unstable because they are identified by two independent methods. On the other hand, the additional regions detected by the spatial anomaly detection method are all in similar regions close to the corresponding blade leading edge.

According to the domain expert, the detected regions close to blade tip are expected regions where stall cells should occur. Therefore this verifies the applicability of the proposed stall detection methods based on anomaly analysis. The other regions close to leading edges are unexpected but interesting too, because these regions could also be related to the inception of stall. Overall, the proposed anomaly analysis methods with the comparative plot provide a guidance to locate possible stall cells for detailed study.

7.6 Summary

In this chapter we have presented different anomaly detection methods and a comparative visualization tool for the analysis of large-scale time-varying flow simulation. The temporal anomaly analysis makes use of windowed FFT as a temporal feature descriptor for comparison; the regional anomaly analysis identifies anomaly among axisymmetric regions by using EMD among local distributions. Both methods are shown effective in experiments of stable and stall conditions. The overlaid anomaly plots allows intuitive comparison between the patterns from different detection methods. More importantly, all presented approaches can clearly distinguish unstable and stable conditions using constant thresholds, which is useful as an indicator for predicting the occurrence of future deep stall during simulation.

The anomaly analysis methods presented in this chapter may be extended to more stall analysis applications. The regional method is more desired than the point-based method

because data represented in distributions typically take less storage space. This is advantageous in *in-situ* stall analysis, where each processor having a passage of data can summarize the data in local distributions before they are transferred to other processors for *expected* distribution computation and comparison. On the other hand, the temporal method is a local approach which does not require information from other passages. Therefore, it can be applied to analyze single-passage simulations or simulations with very few number of passages in the model.

In addition, the presented anomaly detection methods use general techniques for large data analysis when symmetry or periodicity presents in the data. They can locate possible features for further study especially when the feature definition is still under exploration.

Part V: Conclusion

Chapter 8: Conclusion

This dissertation has presented data summarization approaches for large-scale flow visualization and analysis under different levels of I/O constraints.

1. A weighted directed graph is introduced to model runtime data dependencies in time-varying flow fields. When the flow fields are too large to fit into a desktop's main memory, the proposed graph-based file layout and seed scheduling methods are shown to reduce the overall I/O latency of out-of-core pathline computation.

2. Due to limitations on storage and I/O bandwidth, a time-varying flow field is usually down-sampled before it is sent to the visualization machine. An I/O-efficient incremental algorithm is proposed to model the uncertainty of the lost data, which is demonstrated to enable probabilistic particle tracing for uncertainty visualization.

3. To understand the development of rotating stall from large turbine flow simulations, a visual analytics system is developed that efficiently extracts spatiotemporal features to reduce manual efforts in data exploration. Spatial and temporal anomalies as well as vortex orientations are summarized into interactive time-passage visualizations for easier temporal pattern discovery.

Due to the ever growing computing power available for running more complex scientific simulations, efficient analysis for large data has become an immediate need in recent scientific research. Meanwhile, visualization has played an important role to aid human understanding and facilitate pattern discovery from large complex data. The presented data summarization techniques for various scenarios may pave future research on other use cases for flow visualization and analysis. For example, the graph-based summarization of flow dependencies with the use of Markov chains may be employed for unstable flow detection using graph-based anomaly analysis [58, 141]. The proposed distribution-based error modeling for flow fields has also been applied for uncertain FTLE analysis [59]. In addition to modeling errors for down-sampled vector fields, our method is naturally applicable to time-varying scalar fields like pressure fields.

As the data size generated continues to grow rapidly, the amount of information to digest will soon exceed the limit of a human's capability. Automatic data analysis like machine learning and data mining have become indispensable in large data analysis. Since in most times the criteria of features cannot be programmatically defined, the last mile of data analysis is usually completed through visualization with human adjustable detection parameters. Our visual analytics system for rotating stall study is a successful example that reveals unforeseen patterns from automatically detected flow anomalies. In light of this, future research will continue to enhance the coupling between visualization and automatic data analysis techniques to enable scientific data discovery.

Bibliography

[1] Bovas Abraham and Alice Chuang. Outlier detection and time series modeling. *Technometrics*, 31(2):241–248, 1989.

[2] Charu C. Aggarwal. *Outlier analysis*. Springer, 2013.

[3] A. Agranovsky, D. Camp, C. Garth, E. W. Bethel, K. I. Joy, and H. Childs. Improved post hoc flow analysis via lagrangian representations. In *2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 67–75, Nov 2014.

[4] S. Ahern, A. Shoshani, K.-L. Ma, and A. Choudhary. Scientific discovery at the exascale. *Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*, (February), 2011.

[5] James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press.

[6] Salah Ameer. *Investigating polynomial fitting schemes for image compression*. PhD thesis, University of Waterloo, 2009.

[7] N. Andrienko and G. Andrienko. Coordinated views for informed spatial decision making. In *Coordinated and Multiple Views in Exploratory Visualization, 2003. Proceedings. International Conference on*, pages 44–54, 2003.

[8] F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, 1973.

[9] T. Athawale, E. Sakhaee, and A. Entezari. Isosurface visualization of data with nonparametric models for uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):777–786, 2016.

[10] Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 4.3 edition, 2015. ISBN 978-1-930934-30-6.

[11] Richard A Becker and William S Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[12] Michael A. Bender, Bradley C. Kuszmaul, Shang Hua Teng, and Kebin Wang. Optimal Cache-Oblivious Mesh Layouts. *Theory of Computing Systems*, 48(2):269–296, 2011.

[13] J. C. Bennett, H. Abbasi, P. T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9, 2012.

[14] Jacques Bertin. *Semiology of graphics: diagrams, networks, maps*. University of Wisconsin press, 1983.

[15] Wolfgang Bischoff, Heinz Cremers, and Werner Fieger. Normal distribution assumption and least squares estimation function in the model of polynomial regression. *Journal of Multivariate Analysis*, 36(1):1–17, 1991.

[16] A Bogomjakov and C. Gotsman. Universal rendering sequences for transparent vertex caching of progressive meshes. *Computer Graphics Forum*, 21(2):137–149, 2002.

[17] Georges-Pierre Bonneau, Hans-Christian Hege, Chris R. Johnson, Manuel M. Oliveira, Kristin Potter, Penny Rheingans, and Thomas Schultz. *Scientific Visualization: Uncertainty, Multifield, Biomedical, and Scalable Visualization*, chapter Overview and State-of-the-Art of Uncertainty Visualization, pages 3–27. Springer London, 2014.

[18] Ken Brodlie, Rodolfo Allendes Osorio, and Adriano Lopes. A review of uncertainty in data visualization. In *Expanding the Frontiers of Visual Analytics and Visualization*, pages 81–109. Springer London, 2012.

[19] PA Bromiley. Products and convolutions of gaussian probability density functions. Technical report, Tina-Vision, 2014.

[20] Ralph Bruckschen, Falko Kuester, Bernd Hamann, and Kenneth I. Joy. Real-time out-of-core visualization of particle traces. In *PVG '01: Proceedings of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics*, pages 45–50, 2001.

[21] Steven L Brunton and Clarence W Rowley. Fast computation of finite-time Lyapunov exponent fields for unsteady flows. *Chaos*, 20(1):017503, 2010.

[22] Andreas Buja, John Alan McDonald, John Michalak, and Werner Stuetzle. Interactive data visualization using focusing and linking. In *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on*, pages 156–163. IEEE, 1991.

[23] J.J. Caban, A. Joshi, and P. Rheingans. Texture-based feature tracking for effective time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1472–1479, Nov 2007.

[24] D. Camp, H. Childs, A. Chourasia, C. Garth, and K.I. Joy. Evaluating the benefits of an extended memory hierarchy for parallel streamline algorithms. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 57–64, 2011.

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

[26] Howard Castrup. Distributions for uncertainty analysis. *Proceedings of the International Dimensional Workshop*, 12(May 2004), 2001.

[27] Junghoon Chae, Yuchen Cui, Yun Jang, Guizhen Wang, Abish Malik, and David S. Ebert. Trajectory-based Visual Analytics for Anomalous Human Movement Analysis using Social Media. In E. Bertini and J. C. Roberts, editors, *EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association, 2015.

[28] Junghoon Chae, Dennis Thom, Yun Jang, Sung Ye Kim, Thomas Ertl, and David S. Ebert. Visual Analytics of Microblog Data for Public Behavior Analysis in Disaster Events. In M. Pohl and H. Schumann, editors, *EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association, 2013.

[29] Alexis Yee Lyn Chan, Joohwi Lee, and Russell M. Taylor II. Visualization of vortex core differences between ensemble simulations. *IEEE Visualization Contest 2011*, 2011.

[30] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[31] Guoning Chen, Konstantin Mischaikow, Robert S. Laramee, and Eugene Zhang. Efficient morse decompositions of vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):848–862, 2008.

[32] Jen-Ping Chen, Michael D. Hathaway, and Gregory P. Herrick. Prestall behavior of a transonic axial compressor stage via time-accurate numerical simulation. *Journal of Turbomachinery*, 130(4):041014, 2008.

[33] Jenping Chen, Robert Webster, Michael Hathaway, Gregory Herrick, and Gary Skoch. Numerical simulation of stall and stall control in axial and radial compressors. In *44th AIAA Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, 2006.

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

[35] Yi-Jen Chiang. Out-of-core isosurface extraction of time-varying fields over irregular grids. In *Vis '03: IEEE Visualization*, pages 217–224, 2003.

[36] Hank Childs. Data Exploration at the Exascale. *Supercomputing Frontiers and Innovations*, 2(3):5–13, 2015.

[37] Christian Conti, Diego Rossinelli, and Petros Koumoutsakos. GPU and APU computations of Finite Time Lyapunov Exponent fields. *Journal of Computational Physics*, 231(5):2229–2244, 2012.

[38] Carlos D Correa and Peter Lindstrom. The Mutual Information Diagram for Uncertainty Visualization. *International Journal for Uncertainty Quantification*, 3(3):187–201, 2013.

[39] Michael Cox and David Ellsworth. Application-Controlled Demand Paging for Out-of-Core Visualization. In *Vis '97: Proceedings of the IEEE Visualization 1997*, page 235, 1997.

[40] Gustave C. Dahl. *Stall precursor determination of an LM-2500 gas turbine*. PhD thesis, Naval Postgraduate School, 2007.

[41] I. J. Day, T. Breuer, J. Escuret, M. AU - Cherrett, and A. AU - Wilson. Stall inception and the prospects for active control in four high-speed compressors. *Journal of Turbomachinery*, 121(1):18–27, 1999.

[42] IJ Day. Stall inception in axial flow compressors. *Journal of Turbomachinery*, 115(1):1–9, 1993.

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

[44] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro. Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pages 67–75, 2013.

[45] D. Ericson, J. Johansson, and M. Cooper. Visual data analysis using tracked statistical measures within parallel coordinate representations. In *Coordinated and Multiple Views in Exploratory Visualization, 2005. (CMV 2005). Proceedings. Third International Conference on*, pages 42–53, 2005.

[46] Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. In *the Seventeenth International Conference on Machine Learning*, pages 255–262, 2000.

[47] N. Fabian, K. Moreland, D. Thompson, A.C. Bauer, P. Marion, B. Geveci, M. Rasquin, and K.E. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 89–96, 2011.

[48] Nathaniel Fout and Kwan-Liu Ma. An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Transactions on Visualization and Computer Graphics*, 18:2295–2304, 2012.

[49] C. Garth, F. Gerhardt, X. Tricoche, and H. Hagen. Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1464–1471, 2007.

[50] C. Garth, G.-S. Li, X. Tricoche, C. D. Hansen, and H. Hagen. Visualization of coherent structures in 2d transient flow. *Topology-Based Methods in Visualization, Proceedings of the 2007 Workshop*, pages 1–14, 2007.

[51] Christoph Garth, Xavier Tricoche, Tobias Salzbrunn, Tom Bobach, and Gerik Scheuermann. Surface Techniques for Vortex Visualization. *Proc. of VisSym*, pages 155–164, 2004.

[52] Michael Gleicher, Danielle Albers, Rick Walker, Ilir Jusufi, Charles D Hansen, and Jonathan C Roberts. Visual Comparison for Information Visualization. *Information Visualization*, 10(4):289–309, 2011.

[53] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *J. Artif. Int. Res.*, 46(1):235–262, 2013.

[54] W Greene. The least squares estimator. In *Econometric Analysis*. Prentice Hall, seventh edition, 2011.

[55] D.L. Gresh, B.E. Rogowitz, R.L. Winslow, D.F. Scollan, and C.K. Yung. Weave: a system for visually linking 3-d and statistical visualizations applied to cardiac simulation and measurement data. In *Visualization 2000. Proceedings*, pages 489–492, 2000.

[56] M Griebel. Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves. *Parallel Computing*, 25(7):827–843, July 1999.

[57] Frank E Grubbs. Sample criteria for testing outlying observations. *The Annals of Mathematical Statistics*, pages 27–58, 1950.

[58] Yi Gu, Chaoli Wang, Tom Peterka, Robert Jacob, and Seung Kim. Mining Graphs for Understanding Time-Varying Volumetric Data. *IEEE Transactions on Visualization and Computer Graphics*, 2626(c):1–1, 2015.

[59] H. Guo, W. He, T. Peterka, H. W. Shen, S. M. Collis, and J. J. Helmus. Finite-time lyapunov exponents and lagrangian coherent structures in uncertain unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1672–1682, June 2016.

[60] Hanqi Guo, Jiang Zhang, Richen Liu, Lu Liu, Xiaoru Yuan, Jian Huang, Xiangfei Meng, and Jingshan Pan. Advection-Based Sparse Data Management for Visualizing Unsteady Flow. *IEEE Transactions on Visualization and Computer Graphics*, 2626(c):1–1, 2014.

[61] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248–277, 2001.

[62] George Haller and Themistoklis Sapsis. Lagrangian coherent structures and the smallest finite-time Lyapunov exponent. *Chaos*, 21(2):023115, 2011.

[63] M.D. Hathaway, G. Herrick, J. Chen, and R. Webster. Time accurate unsteady simulation of the stall inception process in the compression system of a US army helicopter gas turbine engine. In *31st Annual International Symposium on Computer Architecture*, pages 166–177, 2004.

[64] Marcel Hlawatsch, Filip Sadlo, and Daniel Weiskopf. Hierarchical line integration. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1148–63, 2011.

[65] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[66] T. Höllt, A. Magdy, P. Zhan, G. Chen, G. Gopalakrishnan, I. Hoteit, C. D. Hansen, and M. Hadwiger. Ovis: A framework for visual analysisof ocean forecast ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1114–1126, Aug 2014.

[67] D. A. Hoying, C. S. Tan, Huu Duc Vo, and E. M. Greitzer. Role of blade passage flow structures in axial compressor rotating stall inception. *Journal of Turbomachinery*, 121(October 1):735–742, 1999.

[68] J.C.R. Hunt. Vorticity and vortex dynamics in complex turbulent flows. *Canadian Soc. for Mechanical Eng., Trans.*, 11(1):21–35, 1987.

[69] M. Isenburg and P. Lindstrom. Streaming Meshes. In *VIS '05: Proceedings of the IEEE Visualization 2005*, pages 231–238, 2005.

[70] ISO/TAG4/WG3. *Guide to the Expression of Uncertainty in Measurement (GUM)*. International Organization for Standardization (ISO), Geneva, 1995.

[71] Bruce Jacob, Spencer Ng, and David Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2007.

[72] Heike Janicke, Michael Bottinger, Uwe Mikolajewicz, and Gerik Scheuermann. Visual exploration of climate variability changes using wavelet analysis. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1375–1382, 2009.

[73] W. Javed and N. Elmqvist. Exploring the design space of composite visualization. In *2012 IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–8, 2012.

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

[75] M. Jiang, Raghu Machiraju, and David S. Thompson. Detection and visualization of vortices. In *Visualization Handbook*, pages 287–301. Academic Press, 2004.

[76] Christopher Johnson and Charles Hansen. *Visualization Handbook*. Academic Press, Inc., Orlando, FL, USA, 2004.

[77] C.R. Johnson and A.R. Sanderson. A next step: visualizing errors and uncertainty. *IEEE Computer Graphics and Applications*, 23(5):6–10, 2003.

[78] Doug Joseph and Dirk Grunwald. Prefetching using Markov Predictors. In *ACM SIGARCH Computer Architecture News*, pages 121—-132, 1999.

[79] David Kao, Alison Luo, Jennifer L. Dungan, and Alex Pang. Visualizing spatially varying distribution data. In *Proceedings of the Sixth International Conference on Information Visualisation, 2002*, pages 219–225, 2002.

[80] Vasileios Karavasilis, Christophoros Nikou, and Aristidis Likas. Visual tracking using the earth mover's distance between gaussian mixtures and kalman filtering. *Image and Vision Computing*, 29(5):295–305, 2011.

[81] Z. Karni, A. Bogomjakov, and C. Gotsman. Efficient compression and rendering of multi-resolution meshes. In *Vis '02: Proceedings of the IEEE Visualization 2002*, pages 347–354, 2002.

[82] G. Karypis and V. Kumar. *MeTiS - a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices.*, 1998.

[83] Jens Kasten, Christoph Petz, Ingrid Hotz, BR Noack, and HC Hege. Localized Finite-time Lyapunov Exponent for Unsteady Flow Analysis. *Proceedings of the Vision, Modeling, and Visualization Workshop*, 1, 2009.

[84] Johannes Kehrer, Harald Piringer, Wolfgang Berger, and M Eduard Groller. A model for structure-based comparison of many categories in small-multiple displays. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2287–2296, 2013.

[85] W. Kendall, J. Wang, M. Allen, T. Peterka, J. Huang, and D. Erickson. Simplified parallel domain traversal. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, Nov 2011.

[86] Jinoh Kim, Hasan Abbasi, Luis Chacon, Ciprian Docan, Scott Klasky, Qing Liu, Norbert Podhorszki, Arie Shoshani, and Kesheng Wu. Parallel in situ indexing for data-intensive computing. *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 65–72, 2011.

[87] Edwin M Knox and Raymond T Ng. Algorithms for Mining Datasets Outliers in Large Datasets. *24th International Conference on Very Large Data Bases*, pages 392–403, 1998.

[88] B Kohler, R Gasteiger, U Preim, H Theisel, M Gutberlet, and B Preim. Semi-Automatic Vortex Extraction in 4D PC-MRI Cardiac Blood Flow Data using Line Predicates. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2773–2782, December 2013.

[89] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM Computer Communication Review*, 34(4):219, 2004.

[90] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Euro-Par 2011 Parallel Processing*, number 6852, pages 366–379. Springer Berlin Heidelberg, 2011.

[91] T. Y. Lee, A. Chaudhuri, F. Porikli, and H. W. Shen. Cyclestack: Inferring periodic behavior via temporal sequence visualization in ultrasound video. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pages 89–96, March 2010.

[92] Henry Lehmann and Bernhard Jung. In-situ multi-resolution and temporal data compression for visual exploration of large-scale sientific simulations. *Journal of Chemical Information and Modeling*, 53(9):1689–1699, 2013.

[93] Tom Leighton and Satish Rao. Multicommodity Max-Flow Min-Cut Theorems and Their Use in Designing Approximation Algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[94] Fanyu Li, Jun Li, Xu Dong, Dakun Sun, and Xiaofeng Sun. Stall warning approach with application to stall precursor-suppressed casing treatment. In *Proceedings of ASME Turbo Expo 2016: Turbomachinery Technical Conference and Exposition*, pages 1–9, 2016.

[95] Hongwei Li, Chi-Wing Fu, Yinggang Li, and Andrew Hanson. Visualizing large-scale uncertainty in astrophysical data. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1640–7, 2007.

[96] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 2626(c):1–1, 2014.

[97] H. Ling and K. Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(5):840–853, 2007.

[98] Shusen Liu, J.A. Levine, P. Bremer, and V. Pascucci. Gaussian mixture model based volume visualization. In *2012 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 73–77, 2012.

[99] Xiaotong Liu, Yifan Hu, Stephen North, and Han-Wei Shen. Correlatedmultiples: Spatially coherent small multiples with constrained multi-dimensional scaling. *Computer Graphics Forum*, 2015.

[100] Xiaotong Liu and Han-Wei Shen. The effects of representation and juxtaposition on graphical perception of matrix visualization. In *Proceedings of the 33rd annual ACM conference on Human factors in computing systems*. ACM, 2015.

[101] S. Ljevar, H. C. de Lange, and a. a. van Steenhoven. Two-dimensional rotating stall analysis in a wide vaneless diffuser. *International Journal of Rotating Machinery*, 2006:1–11, 2006.

[102] A.L. Love, A. Pang, and D.L. Kao. Visualizing spatial multivalue data. *IEEE Computer Graphics and Applications*, 25(3):69–79, 2005.

[103] Alison Luo, David Kao, and Alex Pang. Visualizing spatial distribution data sets. In *Proceedings of the Symposium on Data Visualisation 2003*, VISSYM '03, pages 29–38. Eurographics Association, 2003.

[104] Jun Ma, Chaoli Wang, and Ching-kuang Shene. FlowGraph : A Compound Hierarchical Graph for Flow Field Exploration. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pages 233–240, 2013.

[105] Kwan-Liu Ma. In situ visualization at extreme scale: challenges and opportunities. *IEEE Computer Graphics and Applications*, 29(6):14–19, 2009.

[106] Alan M. MacEachren, Anthony Robinson, Susan Hopper, Steven Gardner, Robert Murray, Mark Gahegan, and Elisabeth Hetzler. Visualizing Geospatial Information Uncertainty: What We Know and What We Need to Know. *Cartography and Geographic Information Science*, 32(3):139–160, jan 2005.

[107] N. M. McDougall, N. A. Cumpsty, and T. P. Hynes. Stall inception in axial compressors. *Journal of Turbomachinery*, 112(1):116–123, 1990. McDougallCumpsty-Hynes1990.

[108] Kenneth Moreland, Matthew Larsen, and Hank Childs. Visualization for Exascale: Portable Performance is Critical. *Supercomputing Frontiers and Innovations*, 2(3):67–75, 2015.

[109] National Research Council. *Frontiers in Massive Data Analysis.* Washington, DC: The National Academies Press, 2013.

[110] Rolf Niedermeier and Peter Sanders. *On the Manhattan-Distance Between Points on Space-Filling Mesh-Indexings*. Univ., Fak. für Informatik, 1996.

[111] B. Nouanesengsy, Teng-Yok Lee, Kewei Lu, Han-Wei Shen, and T. Peterka. Parallel particle advection and ftle computation for time-varying flow fields. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, 2012.

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

[113] M. Otto, T. Germer, and H. Theisel. Uncertain topology of 3d vector fields. In *2011 IEEE Pacific Visualization Symposium (PacificVis)*, pages 67–74, 2011.

[114] Mathias Otto, Tobias Germer, Hans-Christian Hege, and Holger Theisel. Uncertain 2d vector field topology. *Computer Graphics Forum*, 29(2):347–356, 2010.

[115] Tianyu Pan, Qiushi Li, Tailu Sun, Zhiping Li, and Yifang Gong. Effects of Radial Loading Distribution on Partial Surge Initiated Instability in a Transonic Axial Flow Compressor. In *Volume 2C: Turbomachinery*, page V02CT44A012. ASME, 2015.

[116] Valerio Pascucci and R.J. Frank. Global Static Indexing for Real-Time Exploration of Very Large Regular Grids. In *SC '01: Proceedings of the ACM/IEEE 2001 Conference on Supercomputing*, pages 45–45, 2001.

[117] T. Peterka, R. Ross, B. Nouanesengsy, Teng-Yok Lee, Han-Wei Shen, W. Kendall, and Jian Huang. A study of parallel particle tracing for steady-state and time-varying flow fields. In *IPDPS '11: IEEE International Parallel Distributed Processing Symposium*, pages 580–591, 2011.

[118] Armin Pobitzer, Ronald Peikert, Raphael Fuchs, Benjamin Schindler, Alexander Kuhn, Holger Theisel, Krešimir Matković, and Helwig Hauser. The State of the Art in Topology-Based Visualization of Unsteady Flow. *Computer Graphics Forum*, 30(6):1789–1811, 2011.

[119] Frits H. Post, Benjamin Vrolijk, Helwig Hauser, Robert S. Laramee, and Helmut Doleisch. The State of the Art in Flow Visualisation: Feature Extraction and Tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[120] Frits H. Post and Theo Walsum. *Focus on Scientific Visualization*, chapter Fluid Flow Visualization, pages 1–40. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[121] Kai Pöthkow and Hans-Christian Hege. Positional uncertainty of isocontours: Condition analysis and probabilistic measures. *IEEE Transactions on Visualization and Computer Graphics*, 17:1393–1406, 2011.

[122] Kai Pöthkow and Hans-Christian Hege. Nonparametric models for uncertainty visualization. In *Proceedings of the 15th Eurographics Conference on Visualization*, EuroVis '13, pages 131–140, 2013.

[123] Kai Pöthkow, Britta Weber, and Hans-Christian Hege. Probabilistic marching cubes. In *Proceedings of the 13th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'11, pages 931–940, 2011.

[124] Kristin Potter, Joe Kniss, Richard Riesenfeld, and Chris R. Johnson. Visualizing summary statistics and uncertainty. *Computer Graphics Forum (Proceedings of Eurovis 2010)*, 29(3):823–831, 2010.

[125] Kristin Potter, Jens Krüger, and Christopher Johnson. Towards the visualization of multi-dimensional stochastic distribution data. In *Proceedings of The International Conference on Computer Graphics and Visualization (IADIS) 2008*, 2008.

[126] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. Least squares as a maximum likelihood estimator. In *Numerical Recipes In C: The Art Of Scientific Computing*, number i, chapter 15.1, pages 657–661. Cambridge University Press, 2nd edition, 1992.

[127] Dave Pugmire, Hank Childs, Christoph Garth, Sean Ahern, and Gunther H. Weber. Scalable computation of streamlines on very large datasets. In *SC '09: ACM/IEEE Supercomputing*, pages 16:1–16:12, 2009.

[128] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data - SIGMOD '00*, 29:427–438, 2000.

[129] W. Reich and G. Scheuermann. Analysis of streamline separation at infinity using time-discrete markov chains. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2140–2148, 2012.

[130] L Reid and RD Moore. Performance of single-stage axial-flow transonic compressor with rotor and stator aspect ratios of 1.19 and 1.26, respectively, and with design pressure ratio of 1.82. *NASA Technical Paper*, 1978.

[131] Marzia Rivi, Luigi Calori, Giuseppa Muscianisi, and Vladimir Slavnic. In-situ visualization: State-of-the-art and some use cases. *PRACE White Paper*, pages 1–18, 2011.

[132] J.C. Roberts. State of the art: Coordinated multiple views in exploratory visualization. In *Coordinated and Multiple Views in Exploratory Visualization, 2007. CMV '07. Fifth International Conference on*, pages 61–71, 2007.

[133] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[134] Brian E. Ruttenberg and Ambuj K. Singh. Indexing the earth mover's distance using normal distributions. *Proceedings of the VLDB Endowment*, 5(3):205–216, 2011.

[135] F. Sadlo, R. Peikert, and E. Parkinson. Vorticity based flow analysis and visualization for pelton turbine design optimization. In *IEEE Visualization, 2004.*, pages 179–186, 2004.

[136] Filip Sadlo and Ronald Peikert. Efficient visualization of lagrangian coherent structures by filtered AMR ridge extraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1456–63, 2007.

[137] Filip Sadlo, Alessandro Rigazzi, and Ronald Peikert. Time-dependent visualization of lagrangian coherent structures by grid advection. In Valerio Pascucci, Xavier Tricoche, Hans Hagen, and Julien Tierny, editors, *Topological Methods in Data Analysis and Visualization*, Mathematics and Visualization, pages 151–165. Springer Berlin Heidelberg, 2011.

[138] T. Salzbrunn and G. Scheuermann. Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1601–1612, 2006.

[139] Tobias Salzbrunn, Christoph Garth, Gerik Scheuermann, and Joerg Meyer. Pathline predicates and unsteady flow structures. *The Visual Computer*, 24(12):1039–1051, 2008.

[140] Roman Sandler and Michael Lindenbaum. Nonnegative matrix factorization with earth mover's distance metric for image analysis. *IEEE trans. on pattern analysis and machine intelligence*, 33(8):1590–1602, 2011.

[141] Aliaksei Sandryhaila and José M.F. Moura. Big Data Analysis with Signal Processing on Graphs. *IEEE Signal Processing Magazine*, 31(5):80–90, 2014.

[142] T Schafhitzel, J E Vollrath, J P Gois, D Weiskopf, A Castelo, and T Ertl. Topology-Preserving $\lambda 2$-based Vortex Core Line Detection for Flow Visualization. *Computer Graphics Forum*, 27(3):1023–1030, 2008.

[143] Dominic Schneider, Jan Fuhrmann, Wieland Reich, and Gerik Scheuermann. A variance based ftle-like method for unsteady uncertain vector fields. In Ronald Peikert, Helwig Hauser, Hamish Carr, and Raphael Fuchs, editors, *Topological Methods in Data Analysis and Visualization II*, Mathematics and Visualization, pages 255–268. Springer Berlin Heidelberg, 2012.

[144] W. Schroeder, K. Martin, and B. Lorensen. The Visualization Toolkit. *Kitware*, 2006.

[145] Shawn C. Shadden, Francois Lekien, and Jerrold E. Marsden. Definition and properties of lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3-4):271–304, 2005.

[146] Sohail Shafii, Herald Obermaier, Rodman Linn, Eunmo Koo, Mario Hlawitschka, Christoph Garth, Bernd Hamann, and Kenneth I Joy. Visualization and analysis of vortex-turbine intersections in wind farms. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1579–91, September 2013.

[147] Cláudio T Silva, Yi-Jen Chiang, Jihad El-Sana, and Peter Lindstrom. Out-Of-Core Algorithms for Scientific Visualization and Computer Graphics. *Visualization 2002 Course Notes*, 2002. IEEE Visualization '02 Course Notes.

[148] Simon Stegmaier, Ulrich Rist, and Thomas Ertl. Opening the Can of Worms: An Exploration Tool for Vortical Flows. In *IEEE Visualization 2005*, pages 463–470. IEEE Computer Society, 2005.

[149] Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. Synergistic Challenges in Data-Intensive Science and Exascale Computing. Technical report, DOE Office of Science, 2013.

[150] P.D. Sulatycke and K. Ghose. A fast multithreaded out-of-core visualization technique. In *IPPS '99/SPDP '99: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, pages 569–575, 1999.

[151] Nobuyuki Tahara, Masahiro Kurosaki, Yutaka Ohta, Eisuke Outa, Takurou Nakajima, and Tomofumi Nakakita. Early Stall Warning Technique for Axial-Flow Compressors. *Journal of Turbomachinery*, 129(3):448–456, 2007.

[152] John K. Taylor and Cheryl Cihon. *Statistical Techniques for Data Analysis*. Chapman and Hall/CRC, 2nd edition, 2004.

[153] Marc Tchiboukdjian, Vincent Danjean, and Bruno Raffin. Binary mesh partitioning for cache-efficient visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):815–28, 2010.

[154] D. Thom, H. Bosch, S. Koch, M. Worner, and T. Ertl. Spatiotemporal anomaly detection through visual analysis of geolocated twitter messages. In *Visualization Symposium (PacificVis), 2012 IEEE Pacific*, pages 41–48, Feb 2012.

[155] D. Thompson, J. A. Levine, J. C. Bennett, P. T. Bremer, A. Gyulassy, V. Pascucci, and P. P. Pébay. Analysis of large-scale scalar data using hixels. In *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 23–30, 2011. ThompsonLevineBennettEtAl2011.

[156] Anna Tikhonova, Carlos D. Correa, and Ma Kwan-Liu. Explorable images for visualizing volume data. *2010 IEEE Pacific Visualization Symposium (PacificVis)*, D(VIDi):177–184, 2010.

[157] M. Tryfonidis, O. Etchevers, J. D. Paduano, A. H. Epstein, and G. J. Hendricks. Prestall Behavior of Several High-Speed Compressors. *Journal of Turbomachinery*, 117(1):62–80, 1995.

[158] Edward R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[159] C. Turkay, A. Slingsby, H. Hauser, J. Wood, and J. Dykes. Attribute signatures: Dynamic visual summaries for analyzing multivariate geographical data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2033–2042, Dec 2014.

[160] Shyh-Kuang Ueng, C. Sikorski, and Kwan-Liu Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, 1997.

[161] P. Valdivia, F. Dias, F. Petronetto, C. T. Silva, and L. G. Nonato. Wavelet-based visualization of time-varying data on graphs. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, pages 1–8, Oct 2015.

[162] Huu Duc Vo, Choon S. Tan, and Edward M. Greitzer. Criteria for spike initiated rotating stall. *Journal of Turbomachinery*, 130(1):011023, 2008.

[163] C. Wang, H. Yu, and K. L. Ma. Application-driven compression for visualizing large-scale time-varying data. *IEEE Computer Graphics and Applications*, 30(1):59–69, 2010.

[164] M. Werman, S. Peleg, and A. Rosenfeld. A distance metric for multidimensional histogram. *CVGIP: Graphical Models and Image Processing*, 32(3):328–336, 1983.

[165] Joseph N. Wilson and Gerhard X. Ritter. Connected Component Algorithms. In *Handbook of Computer Vision Algorithms in Image Algebra*, chapter 6, pages 173–185. CRC Press, 2nd edition, 2000.

[166] Pak Chung Wong and J. Thomas. Visual analytics. *IEEE Computer Graphics and Applications*, 24(5):20–21, 2004.

[167] J. Woodring, M. Petersen, A. Schmeißer, J. Patchett, J. Ahrens, and H. Hagen. In situ eddy analysis in a high-resolution ocean climate model. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):857–866, 2016.

[168] Lijie Xu and H.-W. Shen. Flow Web: a graph based user interface for 3D flow field exploration. In *Proceedings of IS&T/SPIE Visualization and Data 2010*, volume 7530, page 13, 2010.

[169] Sung-Eui Yoon and Peter Lindstrom. Mesh layouts for block-based caches. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1213–20, 2006.

[170] Sung-Eui Yoon, Peter Lindstrom, Valerio Pascucci, and Dinesh Manocha. Cache-oblivious mesh layouts. *ACM Transactions on Graphics*, 24(3):886, 2005.

[171] Hongfeng Yu, Chaoli Wang, Ray W Grout, Jacqueline H Chen, and Kwan-Liu Ma. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications*, 30(3):45–57, 2010.

[172] Yusman Azimi Yusoff, Farhan Mohamed, Mohd Shahrizal Sunar, and Sanjiv Joshi Hari Chand. *Medical Imaging Technology: Reviews and Computational Applications*, chapter State of the Art in the 3D Cardiovascular Visualization, pages 143–168. Springer Singapore, 2015.

[173] Torre Zuk and Sheelagh Carpendale. Theoretical analysis of uncertainty visualizations. *Proceedings of SPIE*, pages 66–79, 2006.