2015

ANALYSIS AND VISUALIZATION OF FLOW FIELDS USING INFORMATION-THEORETIC TECHNIQUES AND GRAPH-BASED REPRESENTATIONS

Jun Ma
Michigan Technological University

ANALYSIS AND VISUALIZATION OF FLOW FIELDS USING INFORMATION-THEORETIC TECHNIQUES AND GRAPH-BASED REPRESENTATIONS

By

Jun Ma

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2015

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Science.

Department of Computer Science

Dissertation Co-advisor: *Ching-Kuang Shene*

Dissertation Co-advisor: *Chaoli Wang*

Committee Member: *Jingfeng Jiang*

Committee Member: *Scott A. Kuhl*

Department Chair: *Min Song*

Contents

List of Figures

List of Tables

Preface

Projects introduced in this dissertation are collaborated with several professors and graduate students. The work in Chapter 3 is a conjunctive work done by Jun Tao, Jun Ma, Professor Chaoli Wang and Professor Ching-Kuang Shene. The algorithm was designed by all of the coauthors and Jun Tao also wrote most of the paper. Jun Ma did the most of the implementation and result generation. For the project in Chapter 4, the algorithm design and the implementation were done by Jun Ma. Professor Chaoli Wang and Professor Ching-Kuang Shene supervised the project completion. For the work in Chapter 5, Jun Ma did the algorithm design and the implementation under the supervision of Professor Chaoli Wang and Professor Ching-Kuang Shene. Professor Jiangjing Feng also evaluated the project tool and contributed a section in the published paper. The project in Chapter 6 was designed and implemented by Jun Ma under the supervision of Professor Chaoli Wang, Professor Scott A. Khul and Professor Ching-Kuang Shene. James Walker and Jun Ma together designed the user study. The work in Chapter 7 was done by Jun Ma and Can Li under the supervision of Professor Chaoli Wang and Professor Ching-Kuang Shene. Jun Ma took the lead and Can Li implemented a few algorithms of the project. Professor Seung Hyun Kim evaluated the system and provided several valuable suggestions. Tools introduced in Chapter 8 were developed by Jun Ma under the supervision of Professor Ching-Kuang Shene and Professor Chaoli Wang. The work in Chapter 9 was jointly achieved by

Yi Gu, Jun Ma under the supervision of Professor Chaoli Wang, Professor Robert J. Nemiroff and Professor David L. Kao. Yi Gu designed the algorithm and also built the framework. Jun Ma helped to extend the desktop program to a distributed system with a tiled display. The data used in the project is provided by Professor Robert J. Nemiroff.

Abstract

Three-dimensional flow visualization plays an essential role in many areas of science and engineering, such as aero- and hydro-dynamical systems which dominate various physical and natural phenomena. For popular methods such as the streamline visualization to be effective, they should capture the underlying flow features while facilitating user observation and understanding of the flow field in a clear manner.

My research mainly focuses on the analysis and visualization of flow fields using various techniques, e.g. information-theoretic techniques and graph-based representations. Since the streamline visualization is a popular technique in flow field visualization, how to select good streamlines to capture flow patterns and how to pick good viewpoints to observe flow fields become critical. We treat streamline selection and viewpoint selection as symmetric problems and solve them simultaneously using the dual information channel [81]. To the best of my knowledge, this is the first attempt in flow visualization to combine these two selection problems in a unified approach. This work selects streamline in a view-independent manner and the selected streamlines will not change for all viewpoints. My another work [56] uses an information-theoretic approach to evaluate the importance of each streamline under various sample viewpoints and presents a solution for view-dependent streamline selection that guarantees coherent streamline update when the view changes gradually. When projecting 3D

streamlines to 2D images for viewing, occlusion and clutter become inevitable. To address this challenge, we design FlowGraph [57, 58], a novel compound graph representation that organizes field line clusters and spatiotemporal regions hierarchically for occlusion-free and controllable visual exploration. We enable observation and exploration of the relationships among field line clusters, spatiotemporal regions and their interconnection in the transformed space. Most viewpoint selection methods only consider the external viewpoints outside of the flow field. This will not convey a clear observation when the flow field is clutter on the boundary side. Therefore, we propose a new way to explore flow fields by selecting several internal viewpoints around the flow features inside of the flow field and then generating a B-Spline curve path traversing these viewpoints to provide users with closeup views of the flow field for detailed observation of hidden or occluded internal flow features [54]. This work is also extended to deal with unsteady flow fields.

Besides flow field visualization, some other topics relevant to visualization also attract my attention. In iGraph [31], we leverage a distributed system along with a tiled display wall to provide users with high-resolution visual analytics of big image and text collections in real time. Developing pedagogical visualization tools forms my other research focus. Since most cryptography algorithms use sophisticated mathematics, it is difficult for beginners to understand both what the algorithm does and how the algorithm does that. Therefore, we develop a set of visualization tools to provide users with an intuitive way to learn and understand these algorithms.

Chapter 1

Introduction

A *vector field* is an assignment of a vector to each point in a subset of the Euclidean space. If the vector at each point in the field varies over time, we call such a vector field an *unsteady vector field* or *time-varying vector field*. Otherwise, it is a *steady vector field*. A *flow field* is a special vector field where the vectors indicate flow directions and magnitudes. A vector field may exhibit several special types of flow pattern, which are called *critical points*. A critical point is a singularity in the vector field when the vector at that point is zero. A critical point may be a *source* (where vectors emanate from a point), *sink* (where vectors converge into a point), *saddle* (where vectors repel each other at a point), and *spiral* (where vectors revolve along a point). Figure 1.1 shows a vector field in the 2D plane and the corresponding critical points. The red line in each image indicates a streamline in the corresponding flow

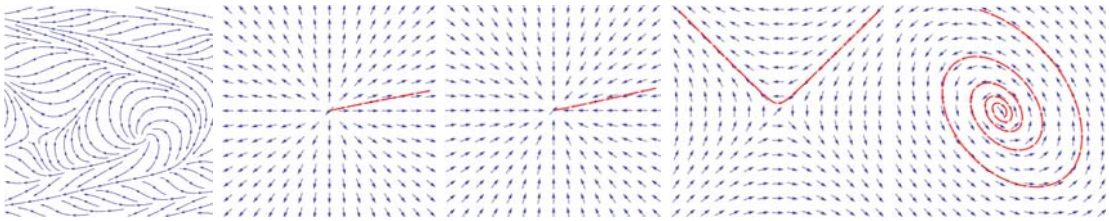


Figure 1.1: (a) a 2D vector field. (b)-(e): a source, sink, saddle and spiral.

field.

In many scientific, engineering and medical disciplines such as climate modeling, turbulent combustion, automobile design and vessel aneurysm diagnosis, visualizing vector fields plays an essential role in visual interpretation and understanding of the underlying flow features and patterns. Well-known vector field visualization techniques include geometry-based methods, texture-based methods [11, 87], integration-based methods [60] and image-based flow visualization (IBFV) [88]. Among these techniques, visualization of *streamlines* and *pathlines* is still the most commonly used method because they are easy to compute and can be rendered at various resolutions with interactive rates.

A streamline is a curve tangent to a flow field everywhere. Intuitively, it is the path that a particle will follow if released in a steady flow field [3]. Streamline visualization is widely used to reveal underlying flow features and patterns for steady flow fields. A pathline is the trajectory of a massless particle which is placed into a unsteady flow field. For example, If a soft ball is dropped into a dynamic water flow whose vectors

change over time, the ball will follow the direction of the flow at each time step and its floating trajectory is called a pathline [3].

Effective *streamline visualization* can be formulated as the problem of *seed placement* or *streamline selection*. Seed placement aims at carefully placing seeds in the domain to generate streamlines that capture flow features. There exist several effective seeding strategies for 2D and 3D vector fields including image-guided [41, 49, 86] and flow-guided [91, 100] algorithms. For 3D flow fields, seeding too many or too few streamlines is not able to reveal flow features and patterns well either because it easily leads to visual clutter in rendering (too many) or it conveys little information about the flow field (too few). Not only does the number of streamlines placed matter, their spatial relationships also influence our understanding of the flow field. Ideally, a streamline seed placement algorithm should retain important features in the vector field so that desired insights can be gained.

An alternative to seed placement is streamline selection. That is, we first place a large number of seeds either randomly or uniformly in the domain to produce a pool of streamlines. We then either automatically select representative or interesting streamlines from the pool [15, 59] or manually sketch a pattern to match similar streamlines for selective display [94]. Although the task is shifted from selecting good seeds to selecting good streamlines, the goal remains the same: we aim to produce a set of streamlines that capture flow features and patterns.

Besides streamline selection, selecting good viewpoints is also critical for understanding large and complex 3D flow fields. This is because automatically guiding the viewers to good viewpoints improves both the speed and the efficiency of data understanding. While viewpoint selection for volume data has been extensively studied [7, 40, 77, 92], the same issue for flow visualization remains to be thoroughly investigated.

Although streamline visualization gains the popularity due to its simplicity to compute using standard numerical integration and providing an its intuitiveness to understand the underlying flow field, a fundamental challenge for scaling vector field from the simple 2D plane to a more complex 3D space still remains unsolved due to occlusion and clutter [45, 61]. Specifically, when depicting a 3D flow field using streamlines, it is often possible to reduce spatial occlusion (e.g., through streamline seeding or filtering) but not eliminate it. This prevents an occlusion-free observation and comparison of the relationships among streamlines, a critical task commonly found in many flow field applications. In order to overcome this problem and help users observe the underlying flow patterns clearly, algorithms have been developed for finding characteristic viewpoints under which users can capture most flow features in a less ambiguous way. However, most existing solutions of viewpoint selection for volume and flow data are restricted to external viewpoints, excluding potentially more effective observation with internal viewpoints.

My work falls into several above flow visualization areas. In Chapter 3, a unified approach to solve streamline selection and viewpoint selection in an information-theoretic framework is introduced. Chapter 4 provides a detailed description of our coherent view-dependent streamline selection based on an importance-driven method. FlowGraph, a compound hierarchical graph for supporting effective 3D flow field exploration is discussed in Chapter 5. Chapter 6 introduces FlowTour, which demonstrates a automatic guide for exploring internal flow features in detail. Its extension is also discussed in Chapter 7. Chapter 8 and Chapter 9 introduce two of my minor research focuses, developing pedagogical visualization tools and dealing with big data using a distributed system and a tiled display wall, respectively.

Due to the extensive use of *information theory* in our work, an introduction to several critical concepts of information theory is provided in Appendix A.

Chapter 2

Related Work

In this section, a general review of previous work related to streamline visualization, viewpoint and view path generation, and flow field exploration are provided. Furthermore, related algorithms which apply information theory and graph drawing techniques to flow visualization are also presented. The following is the organization of this Chapter. In Section 2.1, we list several previous techniques from two major streamline visualization areas: seed placement and streamline selection and also introduce our corresponding solution and point out the differences between our solution and previous methods. Viewpoint selection and view path generation techniques are generally discussed in Section 2.2 along with an introduction of our FlowTour framework. Section 2.3 presents a number of recent work on flow field exploration and Section 2.4 provides a overview of previous work applying information theory into

visualization. At the end of this section, a short description of our unified approach by utilizing information theory to solve streamline selection and viewpoint selection simultaneously is also presented. In the last section, several graph drawing techniques are discussed first and followed by a brief introduction of our FlowGraph tool.

2.1 Streamline Visualization Techniques

Seed Placement: A main focus on flow visualization is seed placement. For example, Jobard and Lefer [41] presented an evenly-spaced seeding algorithm. They took a greedy strategy to place seeds in the neighborhood of previously placed streamlines. A distance threshold is used to explicitly control the density of streamlines. Liu et al. [52] proposed another evenly-spaced streamline placement algorithm for fast, high-quality and robust layout of flow lines. Their solution features double queues to prioritize topological seeding and adaptive distance control to minimize discontinuities. Their loop detection algorithm also helps address closed or spiraling streamlines. Spencer et al. [76] also proposed an efficient algorithm to generate evenly spaced streamlines over surfaces by performing streamline integration in the image space. Similarly, Wu et al. [96] presented a streamline placement algorithm that produces evenly spaced long streamlines while preserving topological features of a flow field. The flow field is decomposed into several topological regions and in each region seeds are placed along a seeding path. Liu and Moorhead [51] proposed an interactive view-driven

evenly spaced streamline placement algorithm for 3D surface flows. Their algorithm integrates streamlines in 3D space while controlling the streamlines density in the 2D view space. They adopted an inter-frame physical-space seeding strategy based on streamline reuse and lengthening on top of their intra-frame view-space seeding, which not only enables coherent flow navigation but also speeds up placement generation.

Besides evenly-spaced techniques, some other placement algorithms were also proposed. Verma et al. [91] argued that the goal of streamline placement is to clearly reveal flow features such as critical points. Therefore, they proposed a flow-guided streamline seeding algorithm that explicitly detects critical points first and then applies different seeding templates to different types of critical points for feature highlighting. This approach was later extended to 3D streamline seeding by Ye et al. [100]. Mebarki et al. [62] took a farthest seeding strategy and placed the seed successively at the place that is farthest away from all previously placed streamlines (i.e., the center of the biggest void region in the field). Schlemmer et al. [71] presented another seeding solution that leverages a user-specified scalar function to control the streamline density. Streamlines are prioritized accordingly and those in the most important regions are drawn first to depict flow features. Li and Shen [49] presented an image-based 3D streamline placement strategy that resolves visual clutter due to streamline projection by placing seeds in the 2D image space. Li et al. [48] proposed illustrative streamline placement to depict the flow patterns succinctly. This algorithm places a new streamline only when it represents flow characteristics that

have not been shown by previously placed streamlines. Xu et al. [98] presented an information-theoretic approach for streamline seeding. Their approach first uses seed templates to place streamlines near regions of high *entropy* (refer to Appendix A) values, then successively places more streamlines according to the conditional entropy between the original flow field and the field reconstructed from previously placed streamlines. Rosanwo et al. [68] proposed a greedy streamline seeding strategy based on so-called dual streamlines that are orthogonal to the given vector field as opposed to primal streamlines which run tangential. Since seeds for new streamlines are only placed along the dual streamlines, their seeding space is reduced to a net of curves. By iteratively refining the dual streamlines, their method can provide a good domain coverage and a high degree of continuity and uniformity.

Streamline Selection: An alternative to seed placement is to either uniformly or randomly place seeds in the field and then adjust the resulting streamlines or select a subset of streamlines for informative visualization. Turk and Banks [86] proposed to use an energy function to guide streamline placement. Their algorithm starts with uniformly or randomly seeded streamlines and then follows an iterative process to improve the visualization by taking several primitive streamline operations (move, insert, delete, lengthen, shorten and combine) and gradually reducing the energy. The energy is defined as the difference between a low-pass filtered version of the streamline image and the desired visual density. Chen et al. [15] selected streamlines from randomly-seeded candidates based on their distance, shape and orientation to

accentuate regions of interest. Their similarity-guided approach produces streamlines that accentuate regions of interest without explicit feature detection and extraction. In the work of Furuya and Itoh [27], they presented a streamline selection technique for integrated scalar and vector field visualization. They first selected and rendered several semitransparent isosurfaces which are used to represent subsets of the given scalar field and then generated a set of streamlines from random seeding. Next, they evaluated each streamlines importance based on its visibility and selected the streamlines with high importance values. The methods proposed by Marchesin et al. [59] and Lee et al. [46] both utilized information theory to evaluate streamline importance and were applied to view-dependent streamline selection. Marchesin et al. [59] presented a view-dependent solution for streamline selection. They defined the contribution of each streamline as how easily this streamline can help the user understand the vector field and selected streamlines based on their contribution in order to reduce visual clutter. Starting from a pool of randomly seeded streamlines, they first removed low contribution streamlines that have small 3D entropy values or have a large overlap with other streamlines given the view, then added new streamlines of high contribution which cover empty areas to provide more context information of the underlying flow field. Lee et al. [46] presented a view-dependent algorithm that minimizes the occlusion and reveals important flow features for 3D flow fields. They utilized Shannon's entropy as a measure of vector complexity and derived an entropy field from the input vector field. Using the maximal entropy projection

(MEP) framebuffer that stores maximal entropy values as well as the corresponding depth values for a given viewpoint, they developed a view-dependent algorithm to evaluate and choose streamlines guided by the MEP framebuffer.

Our Solution: Our coherent view-dependent streamline selection method falls into the category of streamline selection. Unlike the work by Marchesin et al. [59] which only evaluates 3D linear and angular streamline entropies, we evaluate the information loss when 3D streamlines are projected to the 2D image plane. Our strategy is similar to the work of Furuya and Itoh [27]. Instead of only considering a streamline's projected length using entropy [27], we take into account both direction and magnitude of the vectors along the 3D streamline and its 2D projection using *mutual information* (refer to Appendix A). Moreover, we also incorporate the streamline's *shape characteristic* (refer to Appendix A) to obtain our streamline importance measure. Since our solution takes into account view changes in streamline importance evaluation and we carefully select streamlines under each viewpoint by considering the overlap of streamlines between the current and previous viewpoints, we are able to produce *coherent* transition between viewpoints as the user rotates the flow field, which is not achieved in the work of Marchesin et al. [59] and Lee et al. [46]. Both Xu et al. [98] and our method uses information theory for importance evaluation. While the evaluation of information content in Xu et al. [98] is on the flow field, we evaluate the information content on the integrated streamlines.

2.2 Viewpoint Selection and View Path Generation

Viewpoint Selection: Viewpoint selection is another important problem in flow visualization as well as some other areas such as object recognition [4], 3D modeling [23, 95] and mesh saliency [22], volume visualization [7] and cinematography [32]. Vázquez et al. [90] defined the viewpoint entropy based on the projected areas of the polygons of the original models and selected viewpoints with high entropy values. Takahashi et al. [77] first decomposed the whole volume into a set of feature components and computed the locally best viewpoints for each component. The globally best viewpoint was obtained by compromising between locally best viewpoints. Ji and Shen [40] developed a solution to select viewpoints for time-varying data based on a dynamic programming algorithm. Viola et al. [92] found characteristic viewpoints based on an *information channel* (refer to Appendix A) between the model in the scene and the viewpoint set. They measured the viewpoint mutual information for each viewpoint and selected the characteristic viewpoints which have less mutual information values.

View Path Generation: After finding the best viewpoints, the next critical task is to generate a good path traversing all selected viewpoints. van Wijk and Nuij

[89] designed a novel method to generate a path from one viewpoint to another on a 2D map, which guarantees smooth and efficient zooming and panning between two viewpoints along the transformation. The algorithm presented by Ji and Shen [40] suggested a method that combines static view selection with dynamic programming to select time-varying viewpoints and produce a smooth animation. Viola et al. [92] found a transformation path between two selected viewpoints by utilizing an intermediate viewpoint so that the camera path changes smoothly by switching the focus from one feature to another.

Our Solution: All methods are proposed for either 2D images or 3D and 4D volume data sets. There is no work to explore a flow field by traversing selected viewpoints along a predefined view path. We develop our FlowTour to provide the user a automatic guide for exploring the flow field. Furthermore, unlike the work of Viola et al. [92] which only considers the external viewpoints outside of the volume data, our method selected best internal viewpoints and aimed to provide the user a closeup exploration experience for observing hidden and occluded internal flow features and patterns. No only considering steady flow fields, we also extend our work to help the user explore unsteady flow fields automatically using a hybrid optimization strategy.

2.3 Flow Field Exploration Techniques

Even though many popular flow field visualization techniques are proposed to provide a convenient observation and understanding of the flow field, visual exploration of 3D flow fields remains quite a challenge due to occlusion and clutter of 3D streamlines. In order to overcome this problem, a variety of solutions have been proposed. Heiberg et al. [34] proposed to locate, identify and visualize a set of predefined structures in 3D flows using vector pattern matching. Schlemmer et al. [71] presented the idea of invariant moments for analyzing 2D flow fields which allows extraction and visualization of 2D flow patterns, invariant under translation, scaling and rotation. Rössl and Theisel [69] mapped streamlines to points based on the preservation of the Hausdorff metric in the streamline space. The image of the set of streamlines covering the vector field is a set of 2-manifolds embedding in $\mathbf{R}^n$ with characteristic geometry and topology.

Other researchers investigated sketch-based interface and interaction for intuitive flow field exploration. For example, Schroeder et al. [72] presented a sketch-based interface for illustrative 2D vector field visualization which allows illustrators to draw directly on top of the data. Their interface design strikes a good balance between supporting artistic freedom and maintaining the accuracy with respect to the underlying vector field data. Wei et al. [94] presented a user-centric approach to exploring 3D vector

field. By providing the user the freedom to sketch 2D curves, their method can identify and extract all similar field lines based on a pattern matching algorithm between the user defined curves in 2D and field lines clustering in 3D. They also explored another way that creates streamline templates hierarchically to support on-the-fly partial streamline matching in a progressive manner. In the work of Mattausch et al. [60], they presented several strategies to interactively explore 3D flow. Users can interactively change appearance and density of the streamlines in order to assist their flow exploration. Flow features like velocity and pressure are mapped to streamline properties such as width, opacity and density. They also applied animation, depth cueing and halo effect to streamlines.

To enable greater control of interesting flow features and patterns for detail examination, researchers also applied different focus+context techniques into flow field exploration. For example, Mattausch et al. [60] use focus+context to avoid occlusion problems in their 3D vector field exploration method. Fuhrmann and Gröller [26] presented magic lenses and magic boxes to examine the region of interest with greater detail by showing denser streamlines. This technique was extended to magic volumes of varying focus regions such as cubes, prisms and spheres [60]. Laramee et al. [44] leveraged feature-based techniques [5] to extract interesting flow regions, such as stagnant flow, reverse-longitudinal flow and regions of high pressure gradient as the focus and achieved focus+context rendering through interactive thresholding. Correa et al. [16] introduced physical and optical operators to intuitively visualize the

internal 3D flow through illustrative deformation. By cutting along flow traces, they allowed clear observation of the internal 3D flow through optical transformation and elastic deformation. To explore blood flow in cerebral aneurysms, Gasteiger et al. [29] proposed an interactive 2D widget for flexible visual filtering and visualization of the focus+context pairs (i.e., relevant hemodynamic attributes). Their widget supports local probing and conveys changes over time for the lens region.

2.4 Information Theory in Visualization

In recent years, information theory has gained a lot of attentions and permeated into various scientific fields, such as engineering, computer science, mathematics, physics and art. An overview of information theory in visualization can be found in [14, 93]. Chen and Jänicke [14] presented an information-theoretic framework for visualization. Their work outlined the correlation between visualization and the major applications using information theory and proved that information theory can be a useful tool in explaining a number of phenomena in visualization. Wang and Shen [93] complemented Chen's work by presenting a comprehensive summary which reviewed the key concepts in information theory and discussed the applications of information theory in visualization. By demonstrating how information theory is applied to measure the uncertainty quantitively, they connected data communication and data visualization into a single framework and introduced a new direction of

data analysis research based on information theory. Xu et al. [98] focused their work on flow field visualization using information theory and proposed an information-theoretic framework for streamline generation. They formulated a vector field as a distribution of directions and measured the information content in the field based on Shannon's entropy. The conditional entropy is also used to indicate the amount of information in the original data set remains hidden after streamline selection.

Information channel is another critical concept in information theory and has been widely used in visualization recently. Viola et al. [92] used an information channel built between viewpoints and volumetric objects and evaluated viewpoints using mutual information computed from that channel. Feixas et al. [22] proposed an information-theoretic framework for polygonal data in which a channel from viewpoints to polygons and its inverted channel were built, and mutual information of viewpoints and polygons were defined respectively. Ruiz et al. [70] applied a similar method to define voxel information in volume visualization. Several challenges still exist when applying this information-theoretic framework to flow fields.

Our Solution: Unlike voxels which are fine-grained elements and polygons which are fairly localized data items, a streamline could stretch across the entire field and have a very complex shape. This makes it difficult to analyze the conditional probability for a streamline. In addition, there exists no inherent concept of neighbors for streamlines because no connectivity information is given. Therefore, we contribute to

the state of the art flow visualization by introducing a new way to evaluate streamline information and viewpoint information based on the mutual information shared by the 3D streamline and their 2D projection, and constructing two interrelated information channel between a set of streamlines and a set of viewpoints. Our approach treat streamline selection and viewpoint selection as symmetric problem and solve them in a unified framework simultaneously.

2.5 Graph Drawing Techniques

Graph drawing is one essential topic in information visualization and has profound impact on various fields. Battista et al. [5] presented a survey on basic graph drawing algorithms whose goal is to produce aesthetically pleasing graphs. Tollis [83] also provided a brief introduction to the relationship between graph drawing and information visualization. In terms of graph layout, various strategies have been proposed, such as force-based layout [25], spectral layout [6], tree layout [35] and circular layout [21]. Gu and Wang [30] proposed a graph-based representation named TransGraph to visualize hierarchical state transition relationships for time-varying data sets. They utilized a classical force-directed layout algorithm to draw TransGraph and enabled user interaction to connect the graph representation and the volumetric data through brushing and linking. Xu and Shen [99] proposed a node-link graph named flow web for 3D flow field exploration. In their flow web, a node represents a region in the

domain and the strength of a link between two nodes indicates the number of particles traveling between the two regions. Similar graph representations have also been employed for workload estimation in parallel and out-of-core streamline generation [13, 66]. Since the flow web does not explicitly store information about streamline clusters, queries such as identifying streamline bundles become a trial-and-error process. It works for structural flow fields where a path going through a list of nodes may indeed indicate streamline passing through the corresponding regions in order. However, for turbulent flow fields, this may not be true anymore.

Our Solution: Rather than only considering field line clusters or spatial regions, our FlowGraph integrates both field line clusters and spatial regions as nodes simultaneously and thus presents a more complete picture. By integrating the temporal information in the graph, FlowGraph could also provide the user a simple way to observe the flow pattern evolution over time. In this regard, the flow web is actually a subgraph of the FlowGraph. FlowGraph not only provides a visual mapping that abstracts field line clusters and spatiotemporal regions in various levels of detail, but also serves as a navigation tool that guides flow field exploration and understanding.

Chapter 3

A Unified Approach to Streamline Selection and Viewpoint Selection for 3D Flow Visualization

3.1 Overview

Give a set of streamlines, selecting good streamlines to represent flow features and picking good viewpoints to observe flow fields are two major tasks in flow visualization. To treat these two selections as symmetric problems, we present a unified

[0]The material contained in this chapter was previously published in *IEEE Transactions on Visualization and Computer Graphics 2013*.

information-theoretic framework, which solves the problems of streamline selection and viewpoint selection simultaneously by constructing two interrelated information channels between a set of streamlines and a set of viewpoints [81] (©2013 IEEE). Based on the information channel from streamline to viewpoint, we define streamline information as a measure of streamline quality to guide streamline selection. Similarly, in the inverted channel from viewpoint to streamline, we define viewpoint information to guide viewpoint selection for the selected streamlines. Leveraging the two channels, we also present a unified algorithm for streamline clustering and viewpoint partitioning. In addition, a camera path is designed for automatic exploration of the flow field. Our unified approach results in a rigorous and robust framework for selecting good streamlines and viewpoints, clustering streamlines, and partitioning viewpoints, which we demonstrate with flow fields of different characteristics. Since information theory is introduced in Appendix A.2, please refer to that part for how we utilize it to compute streamline importance and build the channel between a streamline set and a viewpoint set. This work has been published in *IEEE Transactions on Visualization and Computer Graphics 2013*. All figures used in the chapter are from the original publication.

3.2 Best Streamlines Selection

For streamline selection, we start from a pool of randomly or uniformly traced streamlines and select the *best streamlines* for display. For streamline tracing, we use the Runge-Kutta method to integrate streamlines as long as possible until they leave the domain or reach critical points. The "best" streamlines are those that best capture flow features by passing through the vicinity of critical points or interesting regions. In this section, we propose two methods to evaluate each individual streamline and then introduce our selection process.

Our first method uses the probability distribution $p(S)$. Since $p(s|v)$ indicates how interesting streamline s is from viewpoint v, $p(s)$ gives us the summation of importance of s from all viewpoints V. If the distribution $p(V)$ is not uniform, $p(s)$ can be considered as a weighted summation, in which a more interesting viewpoint has a higher weight.

Our second method uses the *streamline information* (SI). In the information channel $S \rightarrow V$, we define SI as

$$I(s;V) = \sum_{v \in V} p(v|s) \log \frac{p(v|s)}{p(v)}, \tag{3.1}$$

which represents the degree of dependence between streamline s and the set of viewpoints V. Intuitively, SI indicates the quality of s with respect to V. Note that $I(s;V)$ is the contribution of streamline s to $I(S;V)$, which expresses the degree of correlation between the set of streamlines S and the set of viewpoints V. Low values of SI correspond to streamlines seen by a large number of viewpoints in a *balanced* way. The term "balance" indicates that the conditional probability distribution $p(V|s)$ is similar to $p(V)$. This similarity can be expressed by the Kullback-Leibler divergence [43] between $p(V|s)$ and $p(V)$, which equals zero when $p(V|s) = p(V)$. Conversely, a high value of $I(s;V)$ means a high degree of dependence between s and V. Therefore, streamline s that shows more information over the set of viewpoints V have a lower value of SI.

After streamline evaluation, we sort all the streamlines S into a priority queue. If $p(s)$ is used, the streamlines are sorted in the *decreasing* order of $p(s)$, where a streamline with a higher value of $p(s)$ is preferred. If SI is used, the streamlines are sorted in the *increasing* order of SI, since a streamline with a lower value of SI is better.

We can now select the best streamlines according to the sorted order. Furthermore, we check the pairwise dissimilarity between two streamlines to avoid selecting streamlines that are very similar to each other. To measure streamline dissimilarity, we use the *mean of closest point distances* as suggested by Moberts et al. [64] in DTI fiber clustering. Our selection process starts from selecting the first streamline in the

priority queue. Then, we check the next streamline and select it if its distance to the first one is larger than a given distance threshold d_s. At each step, we consider one new streamline, and compute the distance between it and every streamline previously selected. This streamline is selected if and only if the distances are all larger than d_s. The selection process stops when a given number of streamlines is selected or all streamlines in the pool are considered.

3.3 Best Viewpoints Selection

Similar to SI, in the information channel $V \rightarrow S$, we can define the *viewpoint information* (VI) as

$$I(v; S) = \sum_{s \in S} p(s|v) \log \frac{p(s|v)}{p(s)}, \tag{3.2}$$

which represents the degree of dependence between viewpoint v and the set of streamlines S. Note that in our scenario, the set of streamlines now is actually the set of *selected* streamlines, not the original pool of streamlines. For simplicity, we still use the notation S in this section when referring to the selected streamlines.

Similar to streamline selection, the best viewpoints can be defined either by $p(v)$ or VI. If we use $p(v)$ to select the best viewpoints, we mainly consider the amount of information about the set of streamlines S revealed by viewpoint v. As a result, the best viewpoints are those that show more information of S than others. If we use

VI to select best viewpoints, VI indicates the quality of viewpoint v with respect to the set of streamlines S. Low (high) values of VI correspond to more independent (coupled) viewpoints. Thus, viewpoints with low values of VI are considered as better ones.

Till now, our algorithm could select very similar neighboring viewpoints as the best viewpoints which is clearly not desirable. To avoid this, we make use of the distribution $p(S|v)$ computed in the last step. Considering $p(S|v)$ as a vector associated with each viewpoint, i.e., $p(S|v) =< p(s_1|v), p(s_2|v), \ldots, p(s_n|v) >$, the difference between two viewpoints can be expressed as the Euclidean distance between their corresponding vectors. Thus, a viewpoint is not selected if its distance to any of the selected viewpoints falls below a given threshold d_v.

3.4 Streamline Clustering

We propose a streamline clustering algorithm using the information channels built between S and V. The first stage of our algorithm is to find the *representative streamlines*. Unlike the "best" streamlines (Section 3.2) which are evaluated individually, the "representative" streamlines are defined as a small set of streamlines in which the streamlines as an entirety best characterize the flow field. This is formed by selecting the streamlines such that their merging minimizes the distance to the

target distribution $p(V)$. That is, our selection algorithm should select n' streamlines ($n' << n$) so that their merging $\hat{s}$ minimizes $I(\hat{s}; V)$. Since finding an optimal solution to this problem is NP-complete [70], we adopt a greedy strategy by selecting successive streamlines to minimize $I(\hat{s}; V)$. At each merging step, we aim to maximize the Jensen-Shannon divergence between the set of previously merged streamlines and the new streamline to be selected.

Our solution proceeds as follows. First, we select the best streamline s_1 with distribution $p(V|s_1)$ corresponding to the minimum $I(s; V)$. Next, we select s_2 such that the mixed distribution $\frac{p(s_1)}{p(\hat{s})}p(V|s_1) + \frac{p(s_2)}{p(\hat{s})}p(V|s_2)$ minimizes $I(\hat{s}; V)$, where $\hat{s}$ represents the merging of s_1 and s_2 and $p(\hat{s}) = p(s_1) + p(s_2)$. At each step, a new mixed distribution

$$\frac{p(s_1)}{p(\hat{s})}p(V|s_1) + \frac{p(s_2)}{p(\hat{s})}p(V|s_2) + \ldots + \frac{p(s_i)}{p(\hat{s})}p(V|s_i), \tag{3.3}$$

where $p(\hat{s}) = p(s_1) + p(s_2) + \ldots + p(s_i)$, is produced until the *streamline information ratio* (SIR), denoted as $I(\hat{s}; V)/I(S; V)$, is lower than a given threshold or we have selected n' streamlines. The SIR can be interpreted as a measure of the representativeness of the selected streamlines.

The second stage of our algorithm is to cluster other streamlines to the representatives we have identified in the first stage. Following the data processing inequality [17], we know that any clustering of streamlines reduces the mutual information $I(S; V)$ between the set of streamlines S and the set of viewpoints V. Therefore, a good

clustering is the one that minimizes this mutual information loss. Assuming that two streamlines s_1 and s_2 are merged into one cluster $\hat{s}$, the reduction of mutual information can be described by

$$\begin{aligned} \delta I(s_1; s_2) &= I(S; V) - I(\hat{S}; V) \\ &= p(s_1)I(s_1; V) + p(s_2)I(s_2; V) \\ &\quad -p(\hat{s})I(\hat{s}; V), \end{aligned} \tag{3.4}$$

where $\hat{S}$ is the resulting streamline set and $p(\hat{s}) = p(s_1) + p(s_2)$. Note that $\delta I(s_1; s_2)$ is small if the two streamlines have very similar distributions, i.e., $p(V|s_1) \approx p(V|s_2)$, and it reaches zero if the two streamlines share the same distribution, i.e., $p(V|s_1) = p(V|s_2)$. At each step, we pick a streamline s and calculate $\delta I(s; s')$ for each of the streamlines s' in the representative set. Then, s is merged into the cluster in which $\delta I(s; s')$ between s and its representative s' is minimal.

We use the elbow criterion to determine the proper number of clusters. That is, we should choose a number of clusters so that adding another cluster does not greatly increase the percentage of variance explained (i.e., the ratio of the between-group variance to the total variance). In practice, we run from two to ten clusters from which we choose the appropriate number of clusters.

3.5 Viewpoint Partitioning

Similar to streamline clustering, we can perform viewpoint partitioning in two stages. The first stage is the selection of representative viewpoints and the second stage is clustering other viewpoints to the representatives. The most representative viewpoints are a small number of viewpoints ($m' << m$) that provide the best representation of the selected streamlines. Leveraging the VI measure (Equation 3.2), our solution for viewpoint selection is the same as the greedy solution we propose for identifying representative streamlines (Section 3.4) with the only difference being the swap of notations for streamline and viewpoint. The viewpoint selection process stops when the *viewpoint information ratio* (VIR), denoted as $I(\hat{v}; S)/I(V; S)$, is lower than a given threshold or we have selected m' viewpoints. Similar to the SIR, the VIR can be interpreted as a measure of the representativeness of the selected viewpoints.

For viewpoint partitioning, we measure the difference between two viewpoints by the reduction of mutual information, where the reduction $\delta I(v_1; v_2)$ is defined in the same way as $\delta I(s_1; s_2)$ (Equation 3.4). Then, we apply the same procedure of streamline clustering to partitioning viewpoints: at each step, a viewpoint v is merged into the partition whose representative v' minimizes the information loss measured by $\delta I(v; v')$. Similarly, we use the elbow criterion to identify the proper number of partitions for all the viewpoints.

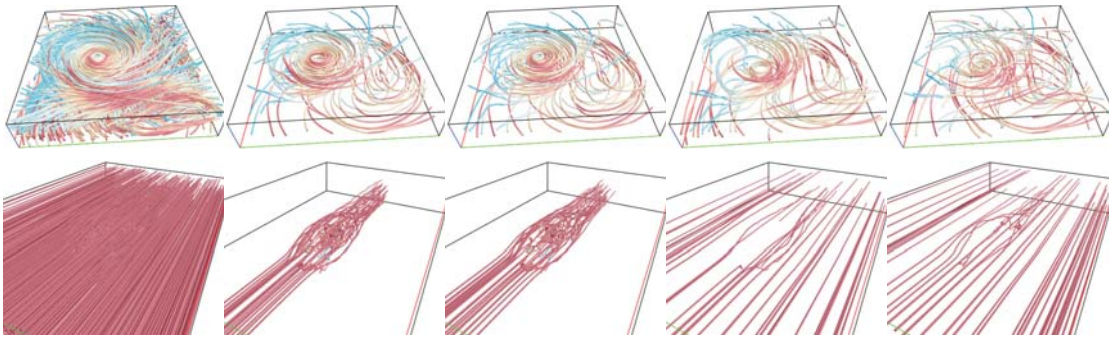


Figure 3.1: Streamline selection (©2013 IEEE).

3.6 Camera Path

Given a set of best (Section 3.3) or representative (Section 3.5) viewpoints, we construct a smooth camera path that goes through all selected viewpoints for automatic flow field exploration. Our algorithm creates a graph by treating all sample viewpoints as nodes and their neighboring relationships as edges. The weight of an edge is defined as the Jensen-Shannon divergence between the two viewpoints. With this graph, we can define the camera path by finding the shortest path among the set of selected viewpoints using Dijkstra's algorithm.

Table 3.1
The ten flow data sets and their parameter values (©2013 IEEE).

data set	dimension	initial # lines	selected # lines	rep. # lines	avg. # pts.	rep. #views	line distance threshold d_s	view distance threshold d_v
five critical points	$51 \times 51 \times 51$	800	140	5	112.9	3	4.1	0.2
tornado	$64 \times 64 \times 64$	500	60	7	295.2	3	5.8	0.1
two swirls	$64 \times 64 \times 64$	500	100	6	157.3	3	3.8	0.18
supernova	$100 \times 100 \times 100$	500	140	5	184.5	4	4.5	0.15
car flow	$368 \times 234 \times 60$	600	140	5	185.5	3	5.9	1.0
crayfish	$322 \times 162 \times 119$	800	100	7	208.7	3	18.7	0.15
solar plume	$126 \times 126 \times 512$	600	140	4	100.2	4	15.0	0.1
computer room	$417 \times 345 \times 60$	800	200	6	172.9	4	17.3	0.15
hurricane	$500 \times 500 \times 100$	600	140	5	341.1	3	31.6	0.15
ABC flow	$1024 \times 1024 \times 1024$	800	140	7	179.8	4	68.3	0.15

Table 3.2
The timing result for different data sets (©2013 IEEE).

	task	tornado	crayfish	hurricane
CPU	best selection $P(S)$	137min	185min	324min
	best selection $I(S;V)$	165min	248min	368min
	clustering	4.0sec	6.4sec	2.5sec
GPU	best selection $P(S)$	6.2sec	9.3sec	7.0sec
	best selection $I(S;V)$	6.2sec	9.3sec	7.0sec
	clustering	0.01sec	0.01sec	0.008sec

3.7 Results

3.7.1 Timing Performance and Parameter Choices

We experimented our approach with ten flow data sets. The five critical points data set [100] is a synthesized flow field consisting of two spirals, two saddles and one source. The tornado data set is from a simulation of a tornado event. The two swirls data set is from a simulation of swirls resulting from wake vortices. The supernova

data set is from a simulation of the explosion of stars. The car flow data set is from a simulation of the air flow around a running car. The crayfish data set is from a simulation of the heat flow around a cooking crayfish. The solar plume data set is from a simulation of down-flowing solar plumes for studying the heat, momentum and magnetic field of the sun. The computer room data set is from a simulation of air flows inside a computer room. Finally, the hurricane data set is from a simulation of Hurricane Isabel, a strong hurricane in the west Atlantic region in September 2003. The ABC flow data set is from a synthesized flow simulation which consists several saddles.

For all these data sets, the initial pool of streamlines was generated by dense placement of seeds randomly. In Table 3.1, we list the parameter values used for each data set. We used 320 viewpoints for all data sets, and determined the number of initial streamlines based on the SIR. Normally, a larger number of initial streamlines should be generated for a data set with a larger spatial dimension or a more complicated structure. For the streamline distance threshold d_s, we used the average of all pairwise mean of closest point distances between streamlines divided by a constant factor. This constant factor should not be too small, in case that the distance becomes the dominant factor in streamline selection. It should not be too large, so that it remains effective in reducing the occlusion. Unlike the streamline distance threshold, the viewpoint distance threshold d_v was chosen as a constant, since the average distance between viewpoints does not vary that much for different data sets.

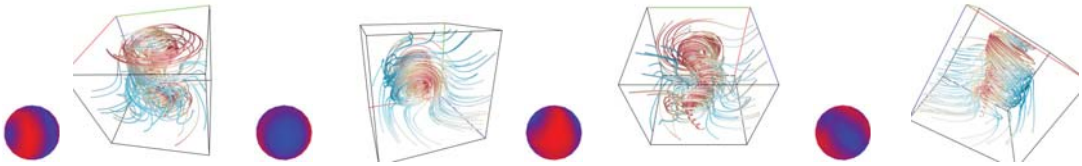


Figure 3.2: Viewpoint ranking of the tornado data set (©2013 IEEE).

Please refer to Table 3.1 for the actual values we used for d_s and d_v for each data set.

Table 3.2 shows the timing results on three benchmark data sets for both CPU and GPU versions of our implementation, tested on a PC with an Intel Core 2 Q6600 quad-core CPU running at 2.4GHz and an nVidia GeForce GTX 465 graphics card.

3.7.2 Streamlines Selection Results

Figure 3.1 shows the comparison of streamline selection results for four different methods: best selections based on $p(s)$ and $I(s;V)$, representative (REP) selection, and random selection. For the hurricane data set (first row, 600 streamlines with 60 selected), both selections based on $p(s)$ and $I(s;V)$ yield similar results. The two circling patterns of the velocity field are clearly visible. REP selection produces less accentuated circling patterns due to the need to cover the domain more evenly in order to best represent the entire field. Random selection leads to a similar result as REP selection, but the circling pattern in the right side of the image is much less obvious. For the car flow data set (second row, 600 streamlines with 40 selected),

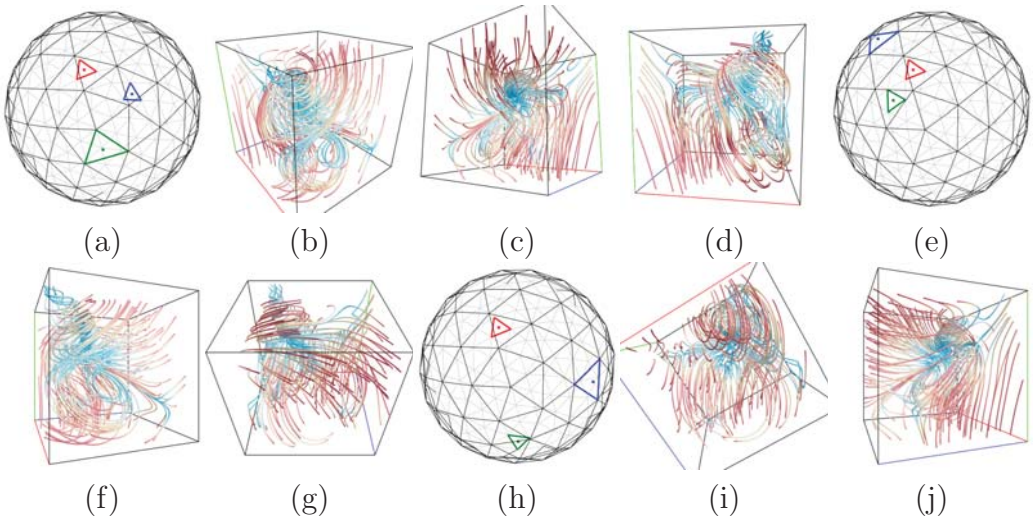


Figure 3.3: Viewpoint selection of the five critical points data set (©2013 IEEE).

both selections based on $p(s)$ and $I(s;V)$ are similar and are clear winners. REP and random selections produce undesirable results because interesting flow features are fairly localized in the domain. Overall, we conclude that streamline selection based on $I(s;V)$ achieves the most consistent results.

3.7.3 Viewpoint Selection Results

In Figure 3.2, we show the ranking of viewpoints based on $p(v)$ and $I(v;S)$ for the tornado data set, together with the corresponding best and worst viewpoints. As expected, the view sphere images indicate that neighboring viewpoints have similar rankings and the viewpoint ranking varies gradually over the view sphere. Although the two methods based on $p(v)$ and $I(v;S)$ give less similar results, the best and worst

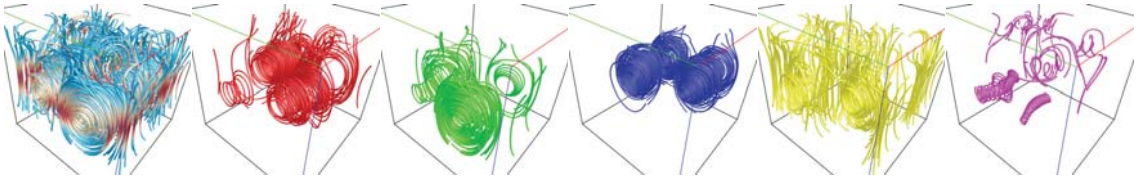


Figure 3.4: Streamline clustering of the two swirls data set (©2013 IEEE).

viewpoints convey the same meaning. That is, the best viewpoint corresponds to a view which clearly reveals the swirling pattern, while the worst viewpoint corresponds to a view where the swirling pattern is least clear. Our result is consistent with the viewpoint evaluation work reported in [46].

Figure 3.3 shows the comparison among best viewpoint selections based on $p(v)$ and $I(v; S)$, and REP viewpoint selection. For each method, we marked where the first three choices (in red, green, and blue) are on the view sphere. (a) shows best viewpoints based on $p(v)$. (b)-(d): three best viewpoints in (a). (b) is the first view and (d) is the third view. (e) shows best viewpoints based on $I(v; S)$ and the first view in (e) is the same as (b). (f) and (g) are the second and third views in (e). (h) shows representative viewpoints. (i) and (j) are the second and third views in (h) and the first view in (h) is the same as (b). We observe that all viewpoints selected are good as they reveal some new information about different critical regions that are not immediately visible from the previous selected viewpoints.

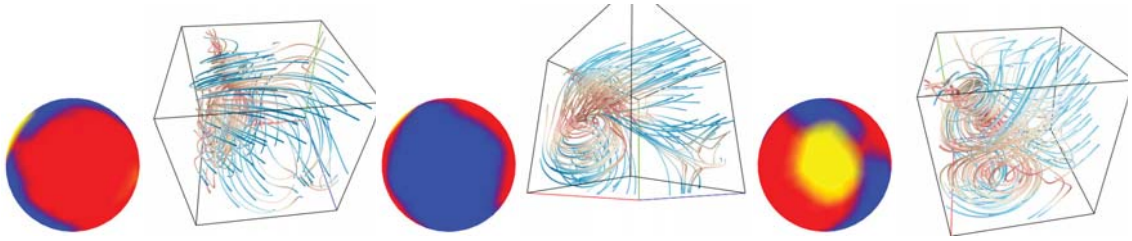


Figure 3.5: Viewpoint partitioning of the five critical points data set (©2013 IEEE).

3.7.4 Streamline Clustering and Viewpoint Partitioning

The streamline clustering results of the two swirls data set (500 streamlines) are shown in Figure 3.4. The blue, yellow, and pink clusters are quite distinct which capture the internal swirls, external swirls, and outliers, respectively. The red and green clusters are in between the blue and yellow ones. Figure 3.5 shows the result of viewpoint partitioning with 1280 total viewpoints. Three partitions are denoted in red, blue and yellow. We show a selected viewpoint from each partition to highlight the distinction among the three partitions.

3.7.5 Camera Path for Visual Exploration

Figure 3.6 shows the camera paths we derived using the shortest path strategy for three data sets . The shortest path is not based on geodesic distances, but according to the Jensen-Shannon divergences. Representative viewpoints were used to plan

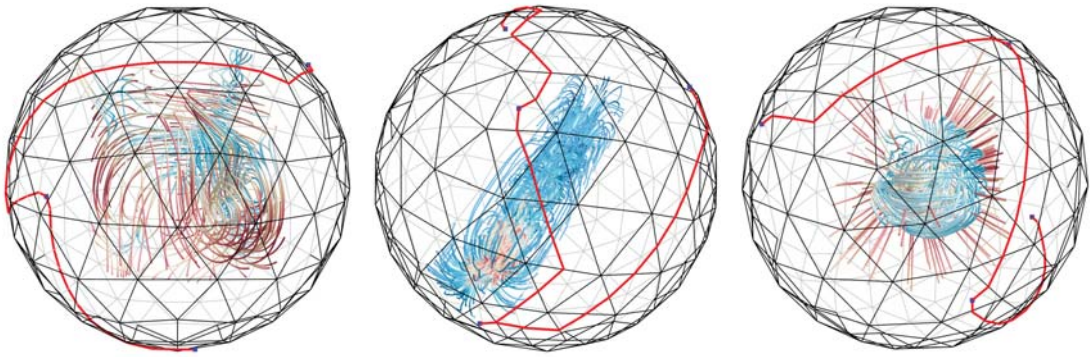


Figure 3.6: Camera paths for the five critical points data set (left), the solar plume data set (middle), and the supernova data set (right) (©2013 IEEE).

the camera path. Each path visits the representative viewpoints one by one. The resulting camera path is smooth because the shortest path between any two target viewpoints ensures that the change along the path is minimized.

Chapter 4

Importance-based Coherent View-dependent Streamline Selection

4.1 Overview

As described in Chapter 3, our unified approach evaluates streamline importance based on the global information of all sample viewpoints and therefore provides a view-independent fashion for streamline selection. This means the final selected streamline

[0]The material contained in this chapter was previously published in *IS&T/SPIE Conference on Visualization and Data Analysis 2013*.

set remains the same for all viewpoints. However, since each streamline has a range of views that show its characteristics in the least ambiguous manner, keeping selected streamline set intact will not generate good results when the viewpoints change gradually. Therefore, we also introduced an importance-driven approach to interactive 3D streamline selection and visualization in a view-dependent manner [56]. This work has been published in *IS&T/SPIE Conference on Visualization and Data Analysis 2013*. All figures used in the chapter are from the original publication.

Our goal is to perform selective streamline display which could not only reduce visual clutter, but also well characterize view-dependent vector field features. We also aim to maintain coherent streamlines updates between adjacent viewpoints. We derive the view-dependent importance of a streamline by computing the amount of information shared by the 3D streamline and its 2D projection under different viewpoints. Taking into account the *shape characteristic* of the streamline under different viewpoints as well, we obtain an importance measure that allows us to identify the intrinsic views of the streamline. Based on this importance measure, we present solutions for both view-independent and view-dependent streamline selection and visualization. For the view-independent case, our solution selects a set of overall important streamlines among all viewpoints and treats it as the globally optimal streamline set. For the view-dependent case, our algorithm dynamically selects important streamlines on the fly and is able to maintain coherent update of streamlines displayed by considering the relationships between local viewpoints and the global streamline set as well

as the continuity between two adjacent viewpoints. We experiment our algorithm with several synthesized and simulated flow fields of different characteristics. The effectiveness of our algorithm is demonstrated through qualitative and quantitative results and comparison with a naïve view-dependent streamline selection algorithm that selects streamlines solely based on the information at the current viewpoint.

4.2 Algorithm Overview

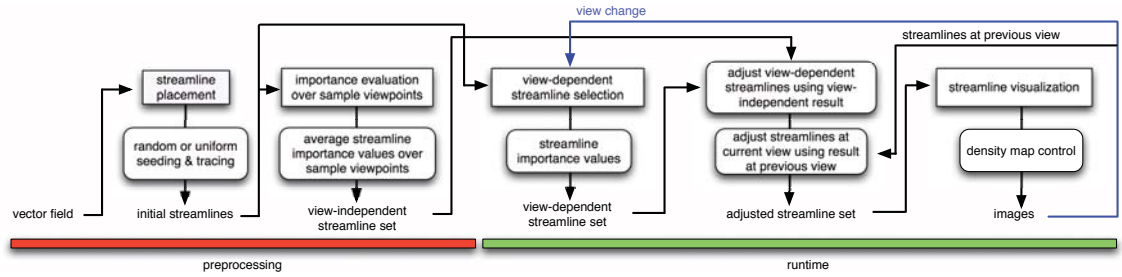


Figure 4.1: The overview of the coherent importance-driven streamline selection.

We sketch the main steps of our algorithm in Figure 4.1. Given an input 3D vector field, we first produce a large number of randomly or uniformly seeded and traced streamlines over the field. To favor long streamlines that better reveal the continuity of the flow field, we integrate each streamline as long as possible until it leaves the domain or the velocity becomes zero. This step of streamline placement can be stopped until a target number of streamlines has been generated or the streamline pool produced is dense enough (e.g., every voxel has been passed through by at least

one streamline). Then we evaluate the importance for each streamline and order them into a priority queue for every single sample viewpoint. More important streamlines are those whose 3D information is high and in the meanwhile, whose 2D projections correspond to their respective intrinsic views that reveal most of their 3D information. In other words, more important streamlines are those whose 2D projections are able to present more 3D shape information of the underlying flow field at the current viewpoint.

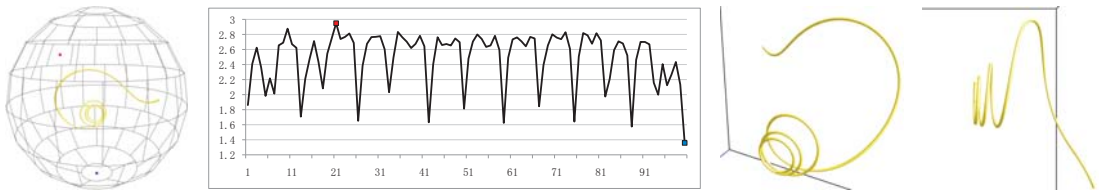
With the streamlines prioritized, we are able to perform view-independent or view-dependent streamline selection and visualization. For the view-independent scenario, our algorithm selects best streamlines considering all sample viewpoints. Overall, the selected streamlines are important from different viewpoints. For the view-dependent scenario, our algorithm dynamically selects important streamlines. We leverage a 2D *density map* and its effective area to control the density of streamlines displayed in the image plane. Since the view-dependent selection is based on both the global streamline information and the continuity between local adjacent viewpoints, our algorithm is able to maintain coherent update of streamline displayed when the view changes gradually.

4.3 View-dependent Streamline Importance

Given a viewpoint, we evaluate the importance of each streamline by considering two criteria: streamline mutual information A.2.3 and streamline shape characteristics A.2.4. The former is to measure how much information of the 3D streamline is revealed in the 2D projection and the latter is to indicate how stereoscopic the streamline shape is reflected under the given viewpoint. With mutual information and shape characteristic defined above, we obtain the view-dependent importance $M(X, v)$ of streamline X under viewpoint v as

$$M(X, v) = \frac{\alpha(\tilde{X}, v) I(X; X_v)}{\sum_{X \in \mathbf{X}} \alpha(\tilde{X}, v) I(X; X_v)}, \tag{4.1}$$

where X_v denotes the 2D projection of X under v and $\mathbf{X}$ denotes the streamline pool. In Figure 4.2, we show an example streamline and the variation of its importance value with all sample viewpoints. In (a) and (b), the best and worst views are marked in red and blue, respectively. Two views corresponding to the best and worst cases are also given. As we can see, the best case corresponds to an intrinsic view having an almost 45° angle with the streamline where much of the 3D streamline characteristics (curvature and torsion) is revealed in the 2D projection. The worst case hides most of the 3D information and displays the streamline in the least certain way.



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

Figure 4.2: (a) a streamline and all 100 sample views based on the spherical partition. (b) the streamline's importance values for all the viewpoints. (c) the best view of the streamline. (d) the worst view of the streamline.

4.4 View-independent Streamline Selection

Our view-independent streamline selection serves as the first step for view-dependent streamline selection. Streamlines selected in the view-independent manner will be used to adjust the view-dependent selection results so that the selected streamlines under each viewpoint always inherit the global "flavor".

After importance evaluation, all streamlines are sorted in a priority queue based on their importance values (Equation 4.1) for each viewpoint. As the view changes, the priority queue gets updated as well. To select streamlines in a view-independent manner, we go over all sample viewpoints and compute the accumulated importance value for each streamline. The final priority queue for the view-independent selection is derived based on the *average* importance value of each streamline under all the viewpoints. We add a distance check to avoid selecting redundant streamlines even though their accumulated importance values are high. To achieve this, we compute

the *minimum* distance of the current streamline under consideration to all streamlines that have been selected. The distance between two streamlines is defined as the Euclidean distance between their corresponding importance values under all sample viewpoints. If this minimum distance is larger than a given distance threshold δ_s, then the current streamline is selected. Otherwise, it is discarded. The creation of streamline priority queue based on all sample viewpoints can be done during the preprocessing (refer to Figure 4.1). At runtime, we simply select a certain number of top-ranked streamlines that pass the distance check for the viewing.

4.5 View-dependent Selection Algorithm

Our view-dependent selection algorithm consists of five steps. First, we obtain a global streamline set S from the view-independent selection algorithm. Second, all the streamlines are sorted based on their importance values under a given viewpoint. Third, we combine the top-ranked streamlines of the first and second steps and put them into a streamline set S_i. Forth, in order to consider the coherence between current and previous viewpoints, we create a new streamline set which is the combination of streamline sets S_i under the current viewpoint and S_{i-1} under the previous viewpoint. Finally, we dequeue each streamline in the new set from step four and leverage a density map to determine whether the streamline should be displayed in the final image or not. By adjusting parameters of the density map, the user can

easily control streamline density in the display. In the following, we describe our view-dependent streamline selection algorithm in detail.

- Step 1: Sort the initial N streamlines in the pool based on the view-independent selection algorithm and obtain a global set S by choosing a certain number of top-ranked streamlines. This global streamline set is the initial reference for view-dependent streamline selection. The number of streamlines selected in S is chosen large enough for the rest of steps. In this work, since the number of finally selected streamlines is usually 1/4 of the total streamline number N, we double this value and set the size of S to $N/2$.

- Step 2: Given a viewpoint v_i, update the priority queue for all streamlines according to their view-dependent importance values in the descending order. Choose the first $N/2$ streamlines as the initial streamline set S_i under v_i. The reason for us to choose $N/2$ streamlines is to ensure that S and S_i share the same size.

- Step 3: Compute the overlap between S and S_i and keep the common streamlines in S_i. Then for the rest of streamlines, remove a certain number of streamlines from S_i based on the mean of the closest point (MCP) distances [64]. Specifically, we compute the MCP distance for each streamline s_i in S_i to all streamlines in S and define the distance from s_i to S as the *maximum* MCP

distance. To maintain the global streamline information under the current viewpoint, we always prefer the streamlines in S_i with small distance values since they are close to S. By contrast, the streamlines in S_i with large distances to S will be discarded. Next, the same number of streamlines from S will be added to S_i based on their view-independent importance. That is, we traverse each streamline in S according to the decreasing importance value and check whether the streamline is shared by S and S_i. If not, we add it into S_i. We keep doing this until we reach the required adding number. Now the newly selected streamlines in S_i contain both view-dependent and view-independent characteristics. The adding or removing number is user-defined. A larger value indicates that the new set S_i is more similar to S while a smaller value means that S_i preserves more of the local information. We test several candidate values and find that 1/5 of the size of S is appropriate which well balances global and local streamline characteristics in S.

- Step 4: In order to maintain a coherent streamline update between two adjacent viewpoints, we compare S_i under viewpoint v_i with S_{i-1} under its previous viewpoint v_{i-1}. This procedure is almost the same as Step 3. First of all, we compute the streamline overlap between S_i and S_{i-1} and keep the common streamlines in S_i. Then for the rest of streamlines, we remove a certain number of streamlines from S_i based on their MCP distances to S_{i-1}. Next, we add the same number of streamlines from S_{i-1} to S_i according to their importance

values. Now we obtain a new S_i which considers the coherence of the current and previous viewpoints.

- Step 5: During this step, we compute the final streamline set for view-dependent display. We propose to use a *density map* to keep track of which regions in the rendered image have been covered by streamline projections and which regions have not. We define the effective area of a density map under a specific viewpoint as the projection area of the date set's bounding box. This would allow us not only to control the streamline number based on the effective area but also to balance streamline selection by reducing visual clutter while revealing interesting flow features and patterns. Note that we do not require the density map to have the same resolution as the final image. A low resolution density map can speed up its update and the subsequent streamline selection process. We assume that each streamline projection L_p has its own *influence region* on the density map. For simplicity, we use a $m \times m$ local mask for each pixel along the projection where the actual mask size is proportional to the final image size. Figure 4.3 shows an example with a 5×5 mask. The weight assigned to each pixel in the influence region is inversely proportional to its Manhattan distance to the central pixel. The weighted average mask is used to compute the importance of the streamline projection to the density map. This step includes the following sub-steps:

  - Sub-step 1: Initialize the density map with an equal density value for all

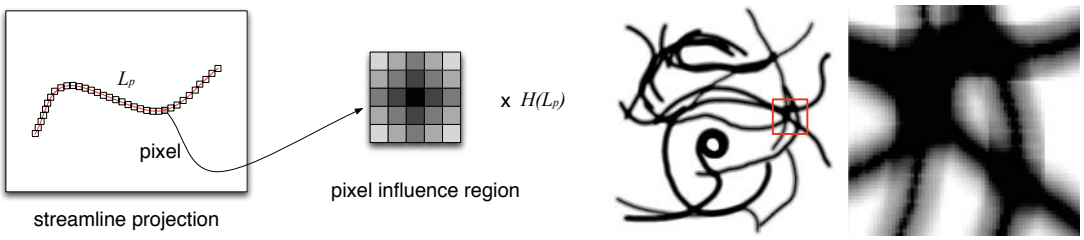


Figure 4.3: Left: the 5×5 influence region for each pixel along the streamline projection. Right: the density map of a hurricane data set and a zoom-in to its middle-right region.

pixels. In the following sub-steps, a streamline L will gain some density value from the pixels it passes through and we define the total density value gained by L as its importance value. Compute the overall effective density value as the summation of all pixels' density values inside the effective area.

– Sub-step 2: Dequeue the streamline L with the highest priority value from S_i and compute its 2D projection's entropy value $H(L_p)$.

– Sub-step 3: Maintain a pixel list that records each pixel along L_p in the image plane. We also define the influence region of the pixel in the list as a $m \times m$ local square centered at that pixel. Then for each pixel in L_p, use a weighted average mask (the influence region) multiplied by $H(L_p)$ to accumulate the importance value gained by L from the density map (see Figure 4.3). Normally, the weight for the central pixel in the mask is set to 1.0. The importance value gained by L from one pixel is bounded above by a maximum importance threshold δ_i.

– Sub-step 4: Subtract the importance values in the pixel list from the density map. Each pixel's density value is bounded below by zero. The summation of total density value loss is defined as the final importance value gained by streamline L from the density map. If this value is above a given density threshold δ_d (i.e., L gains enough importance from the density map), L is selected. Otherwise, L is discarded.

– Sub-step 5: Go to Sub-step 2 until the total importance gained by all selected streamlines is above a given threshold. In this work, we set this threshold to be 2/3 of the overall effective density value. The user can adjust this value to control the density of streamlines displayed.

With this view-dependent streamline selection algorithm outlined above, the final streamlines set S_i is determined not only by the local importance of streamlines but also by their relationships with the global streamline set as well as the streamline set under the previous viewpoint. The motivation for using the initial density map with an equal value is to favor evenly-placed streamlines across the image instead of being cluttered in any one location. This is similar to the image-guided streamline placement algorithm introduced by Turk and Banks [86]. We assign a larger importance value to a streamline with a higher 2D projection entropy. Such a streamline, if selected, would be less likely to be occluded by other streamlines. Setting a maximum importance threshold for each influenced pixel is to ensure that heavily self-occluded

Table 4.1
The threshold and timing results of eight flow data sets for view-independent streamline selection.

data set	dimension	threshold δ_s	initial # lines	average # pts per line	initial # views	selected # lines	importance evaluation time	line selection time
five critical pts	$51 \times 51 \times 51$	40.0	500	110	360	250	11.27s	0.006s
two swirls	$64 \times 64 \times 64$	40.0	500	157	360	250	11.52s	0.008s
tornado	$64 \times 64 \times 64$	40.0	500	295	360	250	12.05s	0.008s
supernova	$100 \times 100 \times 100$	50.0	500	184	360	250	12.20s	0.008s
crayfish	$322 \times 162 \times 119$	50.0	800	209	360	400	19.12s	0.012s
solar plume	$126 \times 126 \times 512$	50.0	600	100	360	300	13.90s	0.007s
computer room	$417 \times 345 \times 60$	60.0	800	173	360	400	18.60s	0.010s
hurricane	$500 \times 500 \times 100$	60.0	600	341	360	300	14.18s	0.010s

streamlines would not get an excessively high importance value. Furthermore, the use of effective area helps us balance the number of streamlines selected under different viewpoints based on the projection of the volume's bounding box.

Table 4.2
The thresholds and timing results of eight data sets for view-dependent streamline selection.

	density map				timing	
data set	dimension	mask	threshold δ_i	threshold δ_d	importance evaluation	line selection
five critical pts	400×400	3×3	1.0	80.0	0.031s	0.340s
two swirls	400×400	3×3	1.0	80.0	0.032s	0.427s
tornado	400×400	3×3	1.0	100.0	0.033s	0.514s
supernova	600×600	7×7	1.0	120.0	0.034s	0.485s
crayfish	600×600	7×7	1.0	250.0	0.053s	0.897s
solar plume	800×800	15×15	1.0	120.0	0.039s	0.982s
computer room	800×800	15×15	1.0	200.0	0.052s	0.857s
hurricane	800×800	15×15	1.0	200.0	0.039s	1.215s

4.6 Results

We experimented our approach with eight flow data sets which are listed in Table 4.1. In the following, we present the machine configuration and timing results, followed by streamline selection results using our view-independent and view-dependent algorithms.

4.6.1 Configuration and Timing

We used a hybrid CPU-GPU solution in our computation with the following hardware configuration: Intel Core i7 quad-core CPU running at 3.20GHz, 24GB main memory, and an nVidia GeForce GTX 580 graphics card. For the view-independent case, the global streamline set was computed using the GPU. For the view-dependent case, all computations were done using the CPU due to the sequential nature of streamline selection with the use of the density map. The timing results are reported in Tables 5.1 and 4.2.

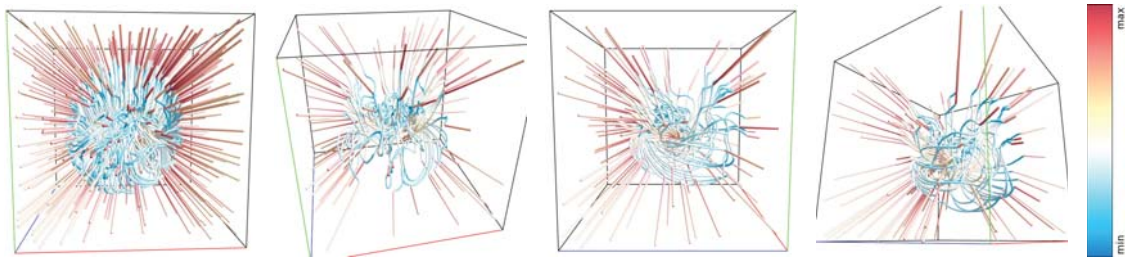
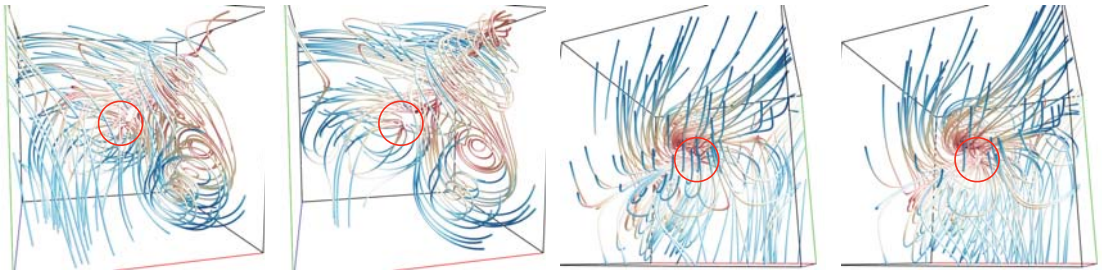


Figure 4.4: View-independent streamline selection with the supernova data set.

4.6.2 View-independent Selection Results

Figure 4.4 shows the results of view-independent streamline selection with the supernova data set. A total of 100 streamlines are selected from the initial pool of 500 randomly seeded streamlines. The first image shows the overall streamline pool while the rest three snapshots show the selected streamlines under three different viewpoints. We map velocity magnitude to streamline color: blue to white to red is for low to medium to high magnitude. Our streamline selection favors "interesting" streamlines that reveal critical flow feature and patterns in a less cluttered view. Redundant streamlines, even with high importance values, are pruned to avoid repetition. However, since this view-independent selection algorithm only considers the global information, it is possible that the results may miss some flow patterns due to the lack of considering the view-dependent information, such as local clutter and occlusion under some particular viewpoints.



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

Figure 4.5: View-independent vs. view-dependent streamline selection with the five critical points data set under two different viewpoints.

4.6.3 View-dependent Selection Results

In Figure 4.5, we compare streamline selection under view-independent and view-dependent cases with the five critical points data set. Two different viewpoints are shown in the figure. In both cases, we can observe that one critical point (source) near the center of the vector field is occluded in the view-independent selection results shown in (a) and (c). By contrast, this source is clearly visible in the view-dependent selection results shown in (b) and (d). The source is highlighted in a red circle. Since the streamlines with high priority mainly go through local critical regions, e.g. the source, and they gain the most importance value from the density map, the streamlines with low priority will not obtain enough importance value to be selected. This is the reason why the local interesting regions are less occluded by dense streamlines. The view-independent selection, however, tends to select more interesting streamlines in regions even though they are already pretty dense in the projection. This is because the view-independent selection only cares the overall

importance of the streamlines but never considers local streamline occlusion under a given viewpoint.

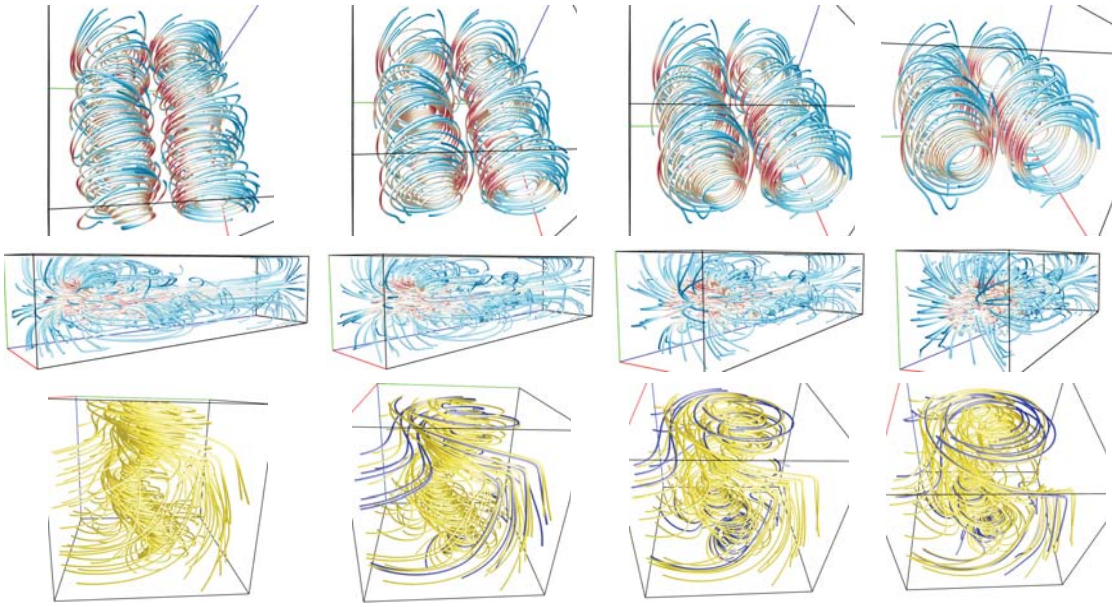


Figure 4.6: Coherent streamline update of three data sets.

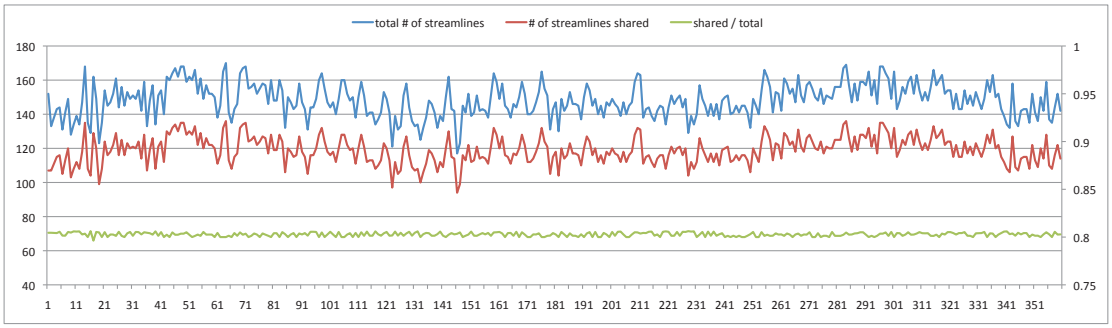


Figure 4.7: The statistics of the numbers of streamlines selected and shared with the supernova data set over 360 sample viewpoints.

4.6.4 Coherent Streamline Update between Adjacent Viewpoints

Figure 4.6 shows the streamline update along four consecutive viewpoints (from left to right) with the two swirls, solar plume and tornado data sets. In order to show our coherent streamline update effect in a more intuitive way, for the tornado data set, we differentiated the streamlines selected from the previous viewpoint in gold and the newly selected streamlines in blue. Clearly, the less number of blue streamlines is, the better the current viewpoint preserves the previous viewpoint's information and the more coherent the view-dependent selection results are.

Figure 4.7 shows the statistics of the numbers of streamlines selected and shared with the supernova data set. We can see that the number of streamlines shared closely follows the trend of the number of streamlines selected. This is also confirmed by their ratio which remains flat over all sample viewpoints. These results show that our algorithm can guarantee coherent streamline update between consecutive viewpoints.

Chapter 5

FlowGraph: A Graph-Based Interface for Visual Analytics of 3D Streamlines and Pathlines

5.1 Overview

When depicting a 3D flow field using streamlines, it is often possible to reduce spatial occlusion (e.g., through streamline seeding or filtering) but not eliminate it. This prevents an occlusion-free observation and comparison of the relationships among

[0]The material contained in this chapter was previously published in *IEEE Pacific Visualization Symposium 2013* and *Transactions on Visualization and Computer Graphics 2014*.

streamlines, a critical task commonly found in many flow field applications. This challenge was echoed in recent state of the art reports on flow visualization [8, 61]. Furthermore, even though streamlines can be organized into a hierarchy to facilitate the understanding [33, 82, 103], visual exploration could still remain a significant challenge due to the lack of capability to observe streamlines and their spatial relationships in a controllable fashion. Pathlines are even more challenging than streamlines due to the addition of the time dimension. In this case, we need to examine and explore pathlines and their spatiotemporal relationships.

Therefore, we present FlowGraph, a visual representation and an interface for effective exploration and analytics of a 3D flow field [57, 58] (). FlowGraph [57] has been published in *IEEE Pacific Visualization Symposium 2013* and its extension [58] has been published in *Transactions on Visualization and Computer Graphics 2014*. All figures used in the chapter are from the original publications. The design target of FlowGraph is to address the intrinsic limitations of 3D occlusion and lack of control when using the standalone field line view for field line exploration, comparison and examination. Our solution works with both streamlines for steady flow fields and pathlines for unsteady flow fields. In conjunction with the standard view of field lines, FlowGraph transforms field line clusters and spatiotemporal regions into a compound hierarchical graph representation to support effective relationship overview and detailed exploration.

We specifically design a set of functions that enable hierarchical exploration of field line clusters, spatiotemporal regions and their interconnection, detailed comparison among field line clusters in terms of their paths passing through different spatiotemporal regions, and close examination of spatiotemporal regions by comparing different field line clusters passing through them. Through brushing and linking, the user can easily make connection between the graph view and the field line view. Animation is used to help intuitive comprehension of graph transition and path illustration. A graph layout algorithm is realized to maintain stable graph update during the level-of-detail exploration. We also introduce animated transition that switches between the entire compound graph and the field line cluster or spatiotemporal region subgraph, allowing observation of the subgraphs in a less cluttered view.

To demonstrate the effectiveness of FlowGraph, we perform several case studies on flow field data sets of various characteristics and conduct an empirical expert evaluation. Our results and the feedback from the expert show that FlowGraph can substantially augment the ability to understand and explore a flow field in different levels of detail, providing the clarity and flexibility previously unavailable.

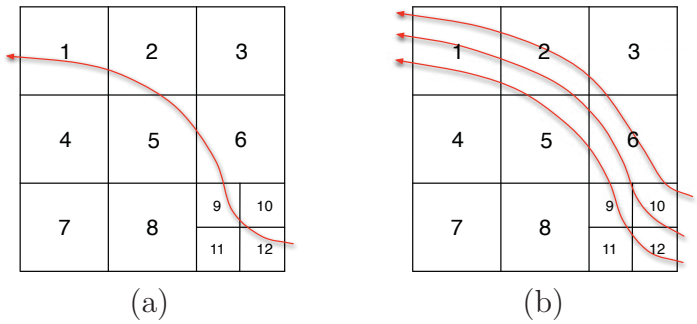


Figure 5.1: Illustration of L-node signature with a 2D space partitioning example (©2014 IEEE).

5.2 FlowGraph Definition and Construction for Steady Flow Field

We define the steady *FlowGraph* as a compound hierarchical graph that consists of two kinds of nodes and three kinds of edges:

- *R-nodes:* A R-node represents a spatial region. We partition the volume space hierarchically using octree and each non-leaf R-node consists of eight child R-nodes. Each R-node maintains three lists recording the streamlines going in, staying inside or going out of the R-node, respectively.

- *L-nodes:* A leaf L-node corresponds to a single streamline, and a non-leaf L-node represents a cluster of streamlines. We organize streamlines hierarchically and each non-leaf L-node usually consists of a different number of child L-nodes. Each L-node maintains a R-node string which indicates the leaf-level regions

which the L-node goes through. If the L-node is a single streamline, the string records a *sequence* of the leaf-level regions it traverses in order. Otherwise, this string records a *set* of the leaf-level regions traversed by all streamlines in the L-node without ordering. We call this string the *signature* of the L-node and define the size of the L-node as the size of its signature, i.e., the number of leaf-level regions. Figure 5.1 illustrates these two kinds of L-node signatures in a 2D scenario. The signature of the streamline in (a) is an ordered sequence $(12, 10, 9, 6, 5, 2, 1)$ and the signature of the streamline cluster in (b) is an unordered set $(1, 2, 3, 5, 6, 9, 10, 11, 12)$.

- *R-R edges:* A R-R edge is formed between two R-nodes at the same level of the space hierarchy. The edge weight records the number of common streamlines shared by these two R-nodes.

- *L-L edges:* A L-L edge is formed between two L-nodes at the same level of the streamline hierarchy. The edge weight records the number of common leaf-level regions traversed in order by these two L-nodes.

- *L-R edges:* A L-R edge is formed between a L-node and a R-node to show their interconnection. The edge weight records the number of streamlines in the L-node passing through the R-node.

5.2.1 Space Hierarchy Construction

We form the space hierarchy by partitioning the spatial domain evenly in a top-down manner using octree. Starting from the entire volume as a single region, we compute the flow entropy based on the joint distribution of vector magnitudes and directions for all vectors within. We partition each region further only if its entropy value per voxel is larger than a given entropy threshold δ_e. The smallest size of a spatial region is also given as another termination condition. This produces a spatial partition similar to an adaptive mesh refinement (AMR) grid.

5.2.2 Streamline Similarity

To construct the streamline hierarchy, we group spatially neighboring and geometrically similar streamlines in a bottom-up manner. Specifically, we define the following two types of similarity to measure the distance between streamlines and the distance between streamline clusters, respectively:

† *Streamline similarity* (for leaf level L-nodes): We consider two factors when computing the similarity between two streamlines l_1 and l_2: the *longest common subsequence* (LCS) of the signatures of l_1 and l_2 and the *mean of closest region distances* (MCR) between l_1 and l_2. We define the distance between two regions

as the distance of their center points. The MCR is a approximation of the *mean of the closest point distance* (MCP) [64] between two streamlines. Specifically, we treat each streamline as a point sequence which consists of the center points of all leaf regions in the streamline's signature. We compute the MCR of two streamlines as the MCP between their center point sequences. Since the number of regions for a streamline is much smaller than the number of points on the streamline, our MCR incurs a much lower computation cost than the MCP does. Furthermore, since the MCR is always computed by using regions at the finest level, its accuracy is also acceptable as judged from the generated streamline clustering results. The final similarity between two streamlines l_1 and l_2 is defined as

$$\Phi(l_1, l_2) = \frac{\mathrm{LCS}(l_1, l_2)}{\max(|l_1|, |l_2|)} - \frac{\mathrm{MCR}(l_1, l_2)}{\mathrm{MCR}_{\max l}}, \tag{5.1}$$

where $\max(|l_1|, |l_2|)$ is the maximum signature size of l_1 and l_2, and $\mathrm{MCR}_{\max l}$ is the maximum MCR among all pairs of streamlines.

† *Streamline cluster similarity* (for non-leaf level L-nodes): Given two streamline clusters c_1 and c_2, we consider two factors for determining their similarity. The MCR is the first factor and we apply the same method used in calculating streamline similarity to the two representative streamlines, one for c_1 and the other for c_2. To determine the spatial overlap of c_1 and c_2, we define the second

factor as the *shared set* (SS) of the signatures of c_1 and c_2. Unlike the LCS computation which considers the order in the signature, the shared set records all common leaf-level regions shared by the two signatures. Finally, we define the similarity between two streamline clusters c_1 and c_2 as

$$\Phi(c_1, c_2) = \frac{\mathrm{SS}(c_1, c_2)}{\max(|c_1|, |c_2|)} - \frac{\mathrm{MCR}(c_1, c_2)}{\mathrm{MCR}_{\max c}}, \tag{5.2}$$

where $\max(|c_1|, |c_2|)$ is the maximum signature size of c_1 and c_2, and $\mathrm{MCR}_{\max c}$ is the maximum MCR among all pairs of streamline clusters.

As we can see, these two similarity definitions are very similar. We replace LCS with SS in the cluster similarity computation. This is because multiple traversal orders may exist for a cluster containing more than one streamline. For the rest of the paper, we do not distinguish these two similarity definitions explicitly and simply state them as the similarity between two L-nodes.

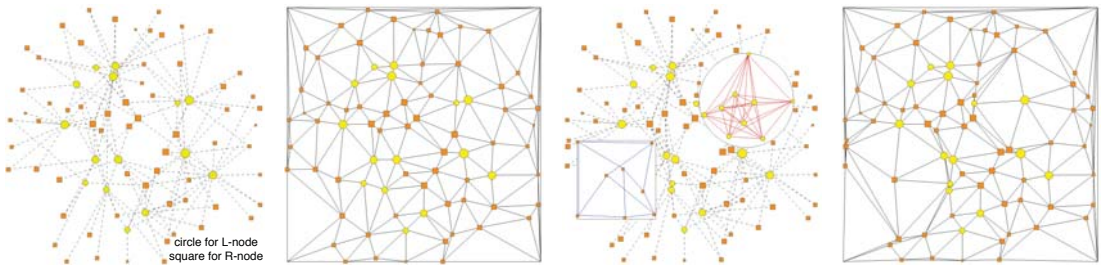
5.2.3 Streamline Hierarchy Construction

With streamline similarity and streamline cluster similarity defined, we take a bottom-up approach to group streamlines level by level to construct the streamline hierarchy.

For each level, we pick the L-node with the *longest* signature size as the first representative and put it into the representative pool. Then, for all other L-nodes, we compute their similarity to the representative pool. We define $\Phi(l, p)$, i.e., the similarity of one L-node to the representative pool, as the *maximum* similarity of this L-node to all representatives currently in the pool, where l denotes the L-node and p denotes the pool. By combining $\Phi(l, p)$ with the L-node signature size $|l|$, we define the representative value of l as

$$\upsilon_l = \left(1 - \frac{\Phi(l, p)}{\max\{\Phi(l, p)\}}\right) + \frac{|l|}{\max\{|l|\}}, \tag{5.3}$$

where $\max\{\Phi(l, p)\}$ denotes the maximum $\Phi(l, p)$, and $\max\{|l|\}$ denotes the maximum L-node signature size among all representative candidates. The next representative is the one with the maximum υ_l which means this L-node is not only dissimilar with any representatives in the pool (a low value of $\Phi(l, p)$) but also traverses a relatively long path (a large value of $|l|$). Then we put the new representative into the pool and repeat this process until we identify enough representatives for this level (the number is usually 1/10 to 1/5 of the number of L-nodes in the lower level). Now we cluster each of the rest of L-nodes into one of the representatives which this L-node is most similar to. Finally, we obtain a new set of L-node clusters and make it the input set for the clustering at the next level. We repeat the entire process until a certain



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

Figure 5.2: Force-directed layout and its adjustment based on triangulation for the solar plume data set (©2014 IEEE).

number of streamline levels is created.

In practice, for constructing FlowGraph, it is desirable for spatial regions or streamline clusters to have three to five levels in their respective hierarchy. This is suggested through empirical observations of the resulting graph's size and complexity. For the streamline hierarchy, the actual number of levels could be larger while we only use several levels at the topmost of the hierarchy for FlowGraph drawing. This would allow us to draw FlowGraph in an efficient way and maintain a good balance between clarity and complexity.

5.3 FlowGraph Drawing for Steady Flow Field

We apply the Fruchterman-Reingold algorithm, a classical force-directed graph layout algorithm [25] to draw the compound FlowGraph in 2D. To distinguish among different kinds of nodes, we use nodes of different colors and shapes: orange squares for

R-nodes and yellow circles for L-nodes. An example is shown in Figure 5.2 with the solar plume data set. Figure 5.2 (a) shows the initial layout produced using the force-directed graph layout algorithm. The size of each node in the graph is proportional to the number of children within. Figure 5.2 (b) is the triangle mesh produced from the initial node positions. Figure 5.2 (c) is the adjusted layout after two nodes are selected and expanded for detail examination. Figure 5.2 (d) shows the underlying triangle mesh used to maintain the topology of the graph during layout adjustment. We also use edges of different colors and styles. In Figure 5.2, L-R edges are drawn in gray dashed lines. For the underlying graph representation, L-L edges and L-R edges are undirected while R-R edges are directed. Given two regions r_1 and r_2, we differentiate between streamlines going from r_1 to r_2 and streamlines going from r_2 to r_1. For simplicity, instead of using double directed R-R edges, we draw a single undirected R-R edge using the summation of the numbers of streamlines passing through these two regions. While all L-L edges and L-R edges are used for computing the layout, for R-R edges, we only use edges that across *neighboring* spatial regions. This prevents the force model from pulling two R-nodes together although they are far away in the spatial domain. The resulting FlowGraph will better reflect the underlying structural relationships among different R-nodes.

At runtime, the user explores the streamline hierarchy or the space hierarchy by clicking a node in the FlowGraph to expand and examine finer detail. Therefore, we need to adjust the layout to accommodate such level-of-detail explorations. A good

layout should maintain a good balance between preserving the structural information of the graph and revealing the dynamics while reducing overlap or occlusion. We generate the initial layout for the coarsest level of the FlowGraph. To achieve stable update, we apply a triangulation scheme [73] to this initial graph and use the result of the triangulation to perform constrained layout adjustment. The four corners of the drawing area are considered as pseudo-nodes in the triangulation. When a node is expanded in the FlowGraph, its initial size is proportional to the number of children in its next level of detail. All nodes expanded are assigned the same scaling factor. The user can also shrink an expanded node back by clicking the empty region inside of the expended node. The surrounding nodes which are pushed away due to the expansion will be pulled back to their respective positions as much as possible.

Similar to the work presented in [18], we consider four kinds of forces to reposition the nodes to reduce their overlap while maintaining the topology of the coarsest level of FlowGraph. These forces include: a *bidirectional repulsive force* which pushes away two nodes u and v from each other and is effective iff u and v overlap each other, a *unidirectional repulsive force* which pushes away a node u without detail shown from a node v with detail shown and is effective iff u is inside of v, a *spring force* which offsets the two repulsive forces introduced by reducing the gap between every pair of nodes in the graph, and an *attractive force* which maintains the topology of the underlying triangle mesh by flipping a triangle back if it is flipped. Figure 5.2 shows an example of layout adjustment during the level-of-detail exploration. As we can

see, the expanded nodes expel other nodes outside of their regions while the global structure of FlowGraph is still preserved. We apply this same layout adjustment strategy recursively to nodes at different hierarchical levels.

5.4 FlowGraph Exploration and Interrogation for Steady Flow Field

The FlowGraph contains a wealth of information that can be effectively utilized for flow field exploration and interrogation. By simply observing the graph, we can already obtain some helpful hints. In a single subgraph (e.g., only R-nodes with R-R edges, or only L-nodes with L-L edges), the size and degree of nodes indicate their importance or significance in the flow field. For instance, if the degree of a R-node is high which means that this R-node has connection to many other R-nodes in terms of streamlines passing through them, it is likely that either this R-node is close to the center of the volume or this R-node contains some critical points such as a sink or source. If the size of a L-node is large, we know that this L-node represents a large streamline cluster. The distance between two nodes also indicates how close their relationship is or how tight their connection is. To extract further information and knowledge about the underlying flow field, we provide the following ways of exploring the graph view and the streamline view.

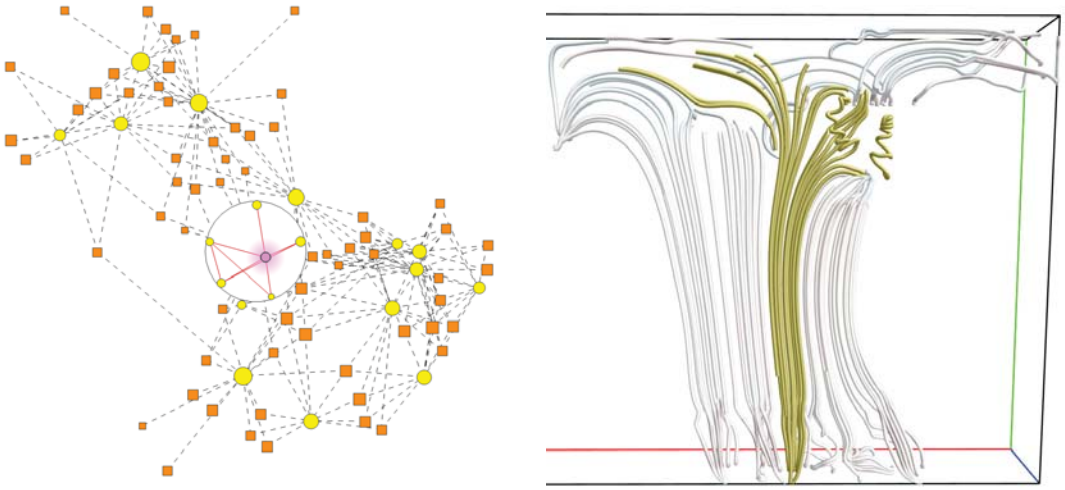


Figure 5.3: Hierarchical exploration of the computer room data set (©2014 IEEE).

Hierarchical Exploration: The FlowGraph organizes L-nodes and R-nodes hierarchically. The user can select a node of interest and expand it to see its next level of detail recursively. In a similar way, the user can further explore each of the nodes at the higher level of detail and make connection to the spatial streamline view. We provide the hierarchical exploration in both the compound graph and a single subgraph. Keyboard shortcuts are added to support convenient traversal through sibling nodes as well as ancestor or descendent nodes.

To provide better context when exploring streamline clusters, we give the option to show the two consecutive levels of streamline clusters in two different colors: the child cluster in a bright color and the rest in a low saturated color. Figure 5.3 shows such an example. The constrained layout adjustment algorithm (Section 5.3) guarantees smooth update of the FlowGraph layout when the user explores nodes at various levels

of the hierarchy. Similarly, we support the same strategy of hierarchical exploration in the streamline view by allowing the user to visit streamline clusters or spatial regions in various levels of detail.

Brushing and Linking: Brushing and linking are the standard technique to make connection among multiple views. We dynamically connect the graph view and the streamline view together: when the user clicks a L-node (R-node) in the graph view, its corresponding streamline cluster (spatial region) is highlighted in the streamline view. Conversely, the corresponding L-node (R-node) will be highlighted in the graph view when the user selects a streamline cluster (spatial region) in the streamline view.

As an option, when a streamline cluster is selected, the corresponding spatial regions which the cluster traverses will be highlighted in the streamline view and at the same time, the corresponding paths passing R-nodes will also be highlighted in the graph view. Similar hints on the corresponding streamline clusters will be provided when a spatial region is selected. Through brushing and linking, especially combined with hierarchical exploration, the user can quickly build up their mental connection between the intuitive streamline view and the abstract graph view.

Filtering and Querying: Filtering and querying the graph helps reduce the complexity of both the graph view and the streamline view, allowing the user to focus on the nodes and edges of interest for detail exploration. We provide a set of queries, including node query (by degree or weight) and edge query (by weight).

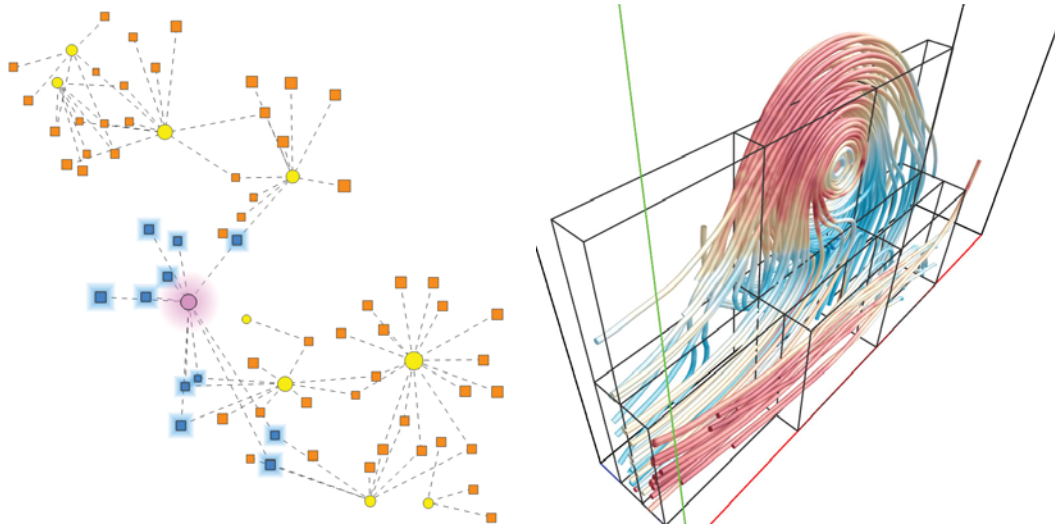
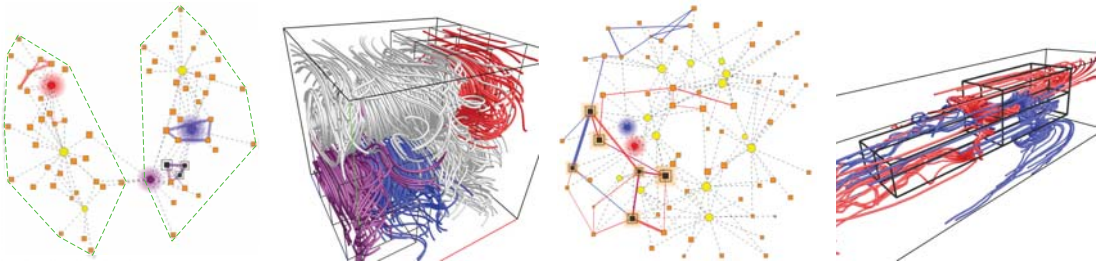


Figure 5.4: Filtering L-R edges by weight in the FlowGraph (©2014 IEEE).



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

Figure 5.5: Path comparisons for the two swirls and the solar plume data sets (©2014 IEEE).

Figure 5.4 shows such an example for filtering L-R edges. Eleven R-nodes (in blue) that have strong connection with the L-node of interest are highlighted. As we expect, these R-nodes are nearby the L-node in the graph view since our force-directed layout algorithm assigns larger attractive forces to node pairs with higher edge weights.

Path Comparison and Region Comparison: Due to the occlusion-free 2D display of the FlowGraph, we enable the user to compare streamline clusters in terms of their paths going through different regions or compare spatial regions in terms of

streamline clusters passing through them in a clear manner.

For path comparison, the user clicks a L-node in the graph and its corresponding paths passing through different R-nodes are highlighted. With hierarchical exploration, we allow comparing L-nodes at different levels of detail. Besides showing the actual paths the streamline cluster passing through, we also implement an algorithm similar to the *maximum spanning tree* algorithm to capture the main structure of the streamline cluster when the paths become cluttered. In addition, we filter out R-R edges of small weights as needed so that paths with very few streamlines passing through can be omitted. We draw undirected edges between R-nodes where the edge thickness indicates the strength of the path (i.e., the number of streamlines passing through in both directions). Multiple L-nodes can be selected simultaneously for path comparison. The paths are highlighted in the graph view and displayed in the spatial streamline view as well when the user mouses over the corresponding L-node. Furthermore, the user can also expend a L-node and check detail path information in a finer level.

For region comparison, the user clicks a R-node in the graph and the L-nodes passing through it are highlighted. Again, in conjunction with hierarchical exploration, we allow comparing R-nodes at different levels of detail. By selecting multiple R-nodes, the user can visually compare the streamline clusters passing through them in both the graph view and the streamline view.

Figure 5.5 (a) and (b) show path comparison with the two swirls data set. We can see that the graph view is highly correlated with the streamline view: the two swirls are well separated in the spatial domain and the corresponding L-nodes and R-nodes form two distinct connected components. Another example of path comparison with the solar plume data set is shown in Figure 5.5 (c) and (d). Unlike the streamline clusters in the two swirls data set, the two streamline clusters in the solar plume data set stretch a wide spatial range and their paths passing over many R-nodes. Six R-nodes shared in common by the two streamline clusters are highlighted in both views. The shared paths are blended of red and blue colors.

For region comparison, the user clicks an R-node in the graph and the L-nodes passing through it are highlighted. Again, in conjunction with hierarchical exploration, we allow comparing R-nodes at different levels of detail. By selecting multiple R-nodes, the user can visually compare the streamline clusters passing through them in both views.

Graph Transition and Path Illustration: We introduce two different animation schemes to facilitate the understanding of the FlowGraph. The first scheme is *graph transition* where we show an animated transition from the compound graph to a single subgraph and vice versa. The motivation is to allow observation of the streamline cluster or spatial region subgraph in a less cluttered view. In addition, compared with the compound graph, the single subgraph layout for L-nodes (R-nodes) forms a

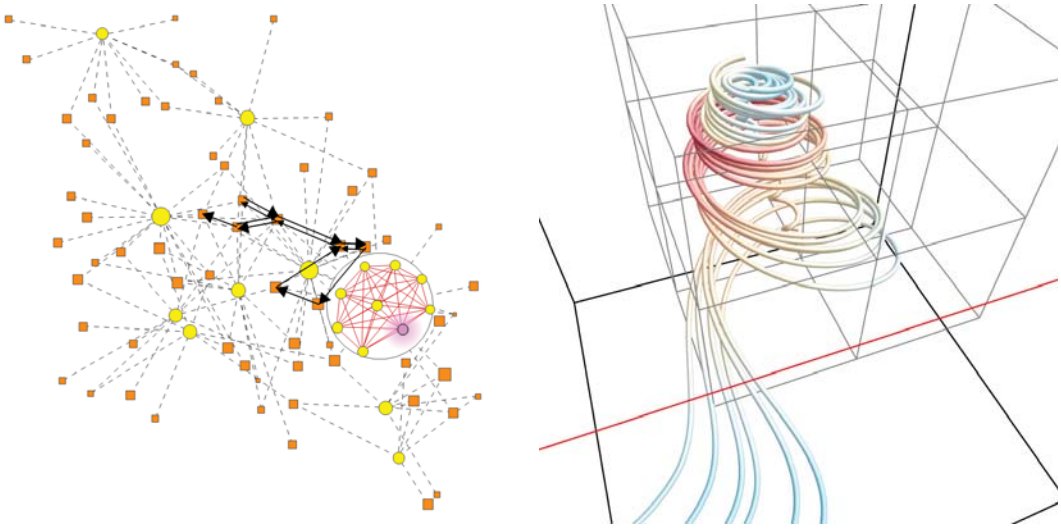


Figure 5.6: The detail path of a child L-node (shown in purple) of the tornado data set and the corresponding streamline cluster (©2014 IEEE).

better organization of node positions for observation of L-L edges (R-R edges).

The second scheme is *path illustration* where we show the detail path information for one streamline or a streamline cluster. For instance, Figure 5.6 shows an example of detail path. The directed black edges in the compound graph indicate the detail path information of the streamline cluster selected. The user can play an animation which indicates how the flow goes through the paths. The animation can be played in both the compound graph and a single subgraph. For the single streamline path animation, we also provide the function to traverse a streamline using animation in the streamline view. This streamline visualization is synchronized with the corresponding path animation shown in the graph view. Such an animation is very intuitive for the user to acquire a solid understanding of the relationships between the streamline or streamline clusters and the corresponding flow regions.

5.5 FlowGraph Extension to Unsteady Flow Field

Due to the high-dimensional nature of unsteady flow fields, providing a visual exploration tool to explicitly show the relationships between pathline clusters and their corresponding spatiotemporal regions becomes a major challenge. To overcome this problem, in this section we extend our FlowGraph to handle 3D unsteady flow fields.

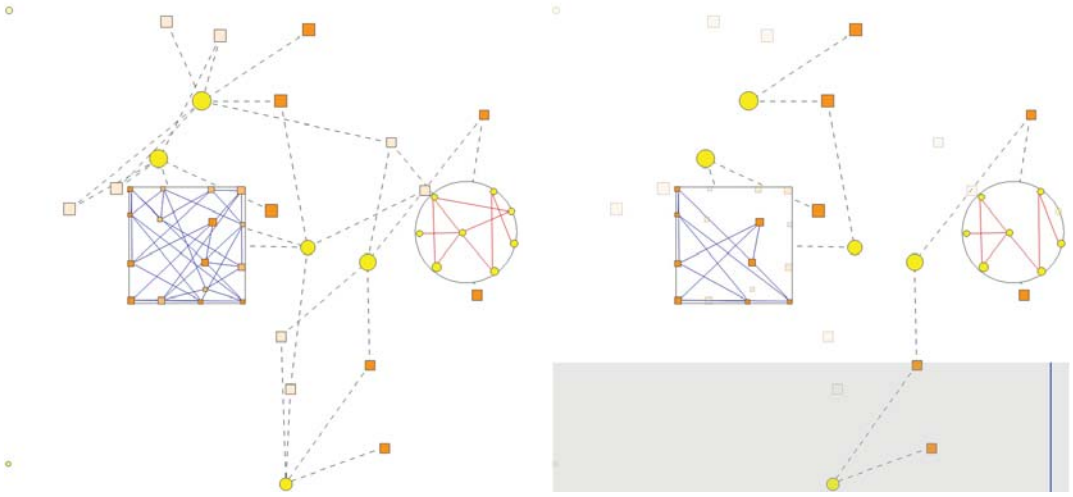
5.5.1 FlowGraph Definition and Construction

Our FlowGraph for unsteady flow fields is also a compound hierarchical graph that consists of two kinds of nodes (R-node and L-nodes) and three kinds of edges (R-R edges, L-L edges and L-R edges). We modify the definitions for these nodes and edges as follows. An R-node now represents a spatiotemporal region. We use a 4D octree (i.e., 16-tree) to partition the unsteady flow data from both spatial and temporal dimensions simultaneously. Each leaf R-node maintains a list recording all pathlines going through the corresponding spatial region within a particular time interval. By treating 3D pathlines as 4D streamlines, we construct L-nodes in the same way as we do for 3D streamlines. Specifically, a leaf L-node represents a single pathline, and a non-leaf L-node indicates a pathline cluster. Furthermore, each L-node records the range of time interval for the pathline or the pathline cluster it corresponds to.

Each L-node also keeps a signature it traverses through where the spatial regions are replaced by the spatiotemporal regions. In terms of edges, by replacing streamlines and spatial regions with pahtlines and spatiotemporal regions, we follow the same definitions in the steady FlowGraph.

5.5.2 Space-Time Hierarchy Construction

Similar to the octree partition in the steady FlowGraph we obtain the space-time hierarchy by partitioning the spatiotemporal domain evenly in a top-down manner using 16-tree. Specifically, we treat the unsteady flow data as a 4D continuous space which contains x, y, z and t (time) components. Starting from the entire 4D data set, we evenly divide it along each dimension at each iteration. The partition termination criteria is still based on the entropy value of the spatiotemporal region or the given threshold for the smallest spatiotemporal size. Intuitively, each partitioned region is a spatiotemporal region group which occupies a cubic volume in space and spans across a certain time interval. In practice, over hundreds of thousands of leaf regions could be generated. We therefore use a 4D tree data structure to store all the leaf regions for fast access.



(a) (b)

Figure 5.7: (a) FlowGraph for the unsteady solar plume data set. (b) FlowGraph with the timeline bar (©2014 IEEE).

5.5.3 Pathline Hierarchy Construction

Since we treat 3D pathlines as 4D streamlines, pathline hierarchy construction follows the same scheme in Sections 5.2.2 and 5.2.3. The difference is that similarity computation is now based on both spatial and temporal information of the corresponding pathlines or pathline clusters. We compute the LCS between two pathline L-nodes' signatures as usual. In terms of MCR computation, rather than only considering spatial distance between two regions, we compute the distance based on both spatial and temporal information by using each region's center as a 4D point (x, y, z, t). We also follow the same solutions used for streamlines to group pathlines level by level for hierarchy construction and to select the representative pathline from each cluster.

5.5.4 FlowGraph Drawing

Several new features are provided to highlight temporal information for FlowGraph drawing. Node and edge drawing follows the same style as before. Color saturation is used to distinguish nodes based on their time spans across the entire time sequence of the data set. Specifically, a node with an early (later) time span is drawn in low (high) saturation. In terms of layout computation, we still apply the Fruchterman-Reingold algorithm where an R-R edge connects two neighboring regions based on both spatial and temporal information. Figure 5.7 (a) shows such an example. To help the user explore the graph at a specific time step, we provide a timeline bar to indicate the current time step and filter out graph nodes whose time spans do not cover it by making them semitransparent. An example is shown in Figure 5.7 (b) where the horizontal direction of the timeline bar from left to right corresponds to early and later time steps. The blue line shows the current time step. Semitransparent nodes indicate the corresponding spatiotemporal regions or pathlines whose time spans do not cover the current time step.

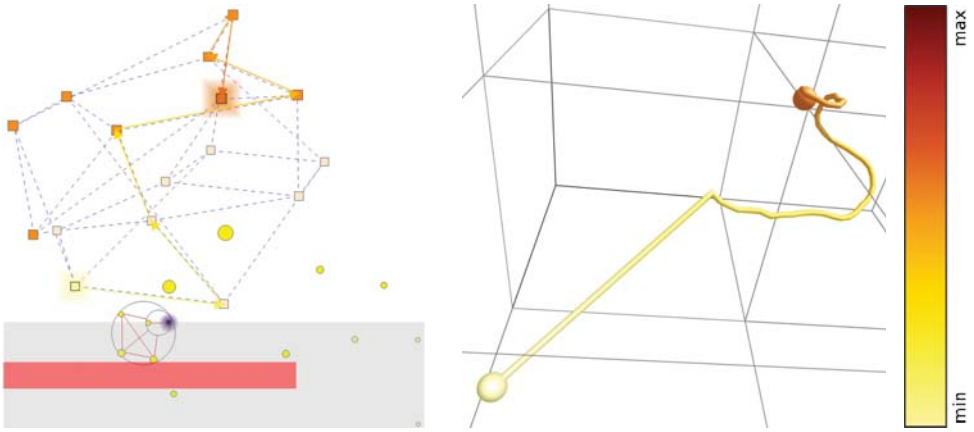
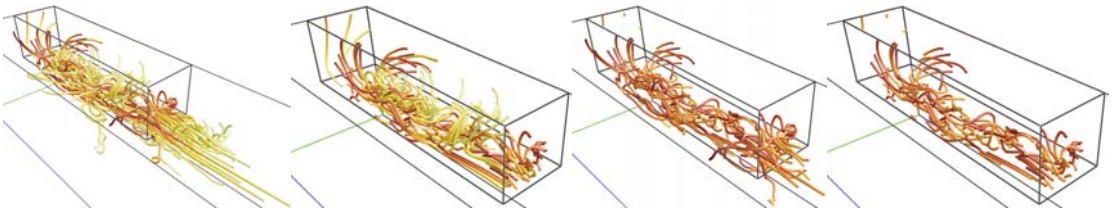


Figure 5.8: The colored single path for the unsteady supernova data set (©2014 IEEE).



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

Figure 5.9: (a) pathlines going through the selected spatiotemporal region for the unsteady solar plume data set. (b) to (d): pathline segments inside of the region's spatial boundary, temporal boundary, and spatial and temporal boundaries, respectively (©2014 IEEE).

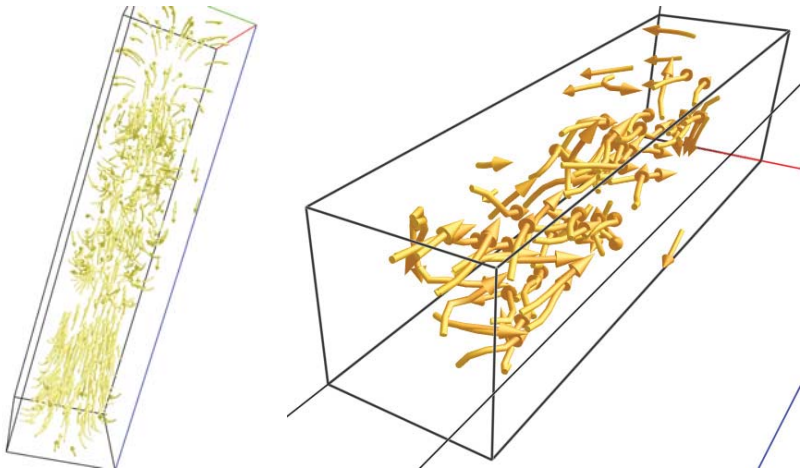
5.5.5 FlowGraph Exploration and Interrogation

Our FlowGraph for pathlines and spatiotemporal regions keeps all the exploration and interrogation functions for streamlines and spatial regions. Furthermore, by plugging the time information into the graph, FlowGraph conveys more information and provides the user with more flexibility to observe and explore the unsteady flow field. For example, since we use the entropy to determine the size of an R-node, two

R-nodes which occupy the same spatial region but cover different time spans could indicate the change of entropy in the same spatial region over time. For hierarchical exploration, when the user selects a node, we also show its time span in red in the timeline bar. We provide a time slider to help the user select a spatiotemporal region at any specific time step. For path comparison and region comparison, when demonstrating the path for a single pathline in the graph view, instead of drawing the path in black, we colorize the path using the same color mapping for pathline drawing to show the time correspondence. An example is shown in Figure 5.8. In order to differentiate pathlines from streamlines, we use a different colormap for pathline drawing. Yellow indicates the earliest time step and brown indicates the latest time step. The red interval in the timeline bar indicates the time span of the selected pathline.

Besides these existing functions for FlowGraph, we add the following new features to handle graph exploration involving the temporal aspect:

Pathline Spatial and Temporal Filtering. We provide pathline filtering option to allow the user to focus on the pathlines for a specific spatial region or time interval. Basically, when the user selects a region in the pathline view, FlowGraph shows all the pathlines passing through this region by default. However, sometimes it could be difficult for the user to observe clearly the flow patterns inside of the selected region. Possible reasons are that there may be too many pathlines going through the selected



(a) (b)

Figure 5.10: (a) all the pathlets in a specific time interval for the unsteady solar plume data set. (b) the pathlets inside of a selected spatiotemporal region (©2014 IEEE).

region and these pathlines may also pass through some other regions and thus make the view cluttered. To alleviate this issue, we render the portions of pathlines that are only inside of the region by filtering out pathlines segments that are outside of the region's spatial or temporal boundary. Figure 5.9 shows an example of this filtering.

Pathlet Rendering and Animation. Rather than only showing the entire pathline indicating the whole trajectory of a particle, we also draw the pathlet to show a segment of the trajectory over a short time interval. The arrow of the pathlet indicates the current flow direction. Figure 5.10 (a) shows the pathlets for the unsteady solar plume data set. By utilizing pathlet rendering, our FlowGraph allows the user to only focus on the flow patterns in some specific time interval. Moreover, using a time slider, the user can obtain pathlet animation to indicate the evolution of flow over time. Pathlet rendering and animation could be combined with other functions

Table 5.1
The parameters and timing results for FlowGraph construction of eleven data sets (©2014 IEEE).

data set	dimension	init. # lines	minimum region	δ_e	GPU entropy field	CPU Lnodes	CPU Rnodes	CPU edges	graph size (MB)
car flow	$368 \times 234 \times 600$	600	$11 \times 7 \times 18$	0.2	0.1s	271s	0.01s	51s	25
computer room	$417 \times 345 \times 60$	800	$13 \times 10 \times 1$	0.9	0.1s	324s	0.04s	52s	36
five critical pts	$51 \times 51 \times 51$	500	$1 \times 1 \times 1$	1.0	0.1s	244s	0.02s	52s	37
hurricane	$500 \times 500 \times 100$	600	$15 \times 15 \times 3$	0.8	0.3s	231s	0.01s	51s	27
solar plume	$126 \times 126 \times 512$	600	$3 \times 3 \times 16$	1.1	0.1s	884s	0.03s	53s	30
supernova	$100 \times 100 \times 100$	500	$3 \times 3 \times 3$	0.8	0.1s	244s	0.02s	52s	24
tornado	$64 \times 64 \times 64$	500	$2 \times 2 \times 2$	1.0	0.1s	779s	0.03s	54s	24
two swirls	$64 \times 64 \times 64$	500	$2 \times 2 \times 2$	1.3	0.1s	325s	0.01s	51s	24
hurricane	$500 \times 500 \times 100 \times 48$	800	$15 \times 15 \times 3 \times 1$	0.8	14.8s	1540s	91.69s	281s	175
solar plume	$126 \times 126 \times 512 \times 29$	600	$7 \times 7 \times 32 \times 1$	0.2	3.6s	380s	20.03s	31s	122
supernova	$216 \times 216 \times 216 \times 105$	500	$3 \times 3 \times 3 \times 3$	0.8	12.5s	1359s	85.76s	112s	103

to provide the user with a more comprehensive understanding of the flow fields. For example, when the user selects a region and wants to observe the corresponding pathlines going through this region, she can first applies pathline spatial and temporal filtering and then uses pathlets to demonstrate how the flow patterns change inside of this region over its time span. Figure 5.10 (b) shows such an example.

5.6 Results

We experimented our approach with eight steady flow data sets and three unsteady flow data sets which are listed in Table 5.1.

We used a hybrid CPU-GPU solution in our computation with the same hardware configuration used in Chpater 4. The parameter setting and timing performance are reported in Table 5.1. For all steady data sets, we randomly placed the seeds to

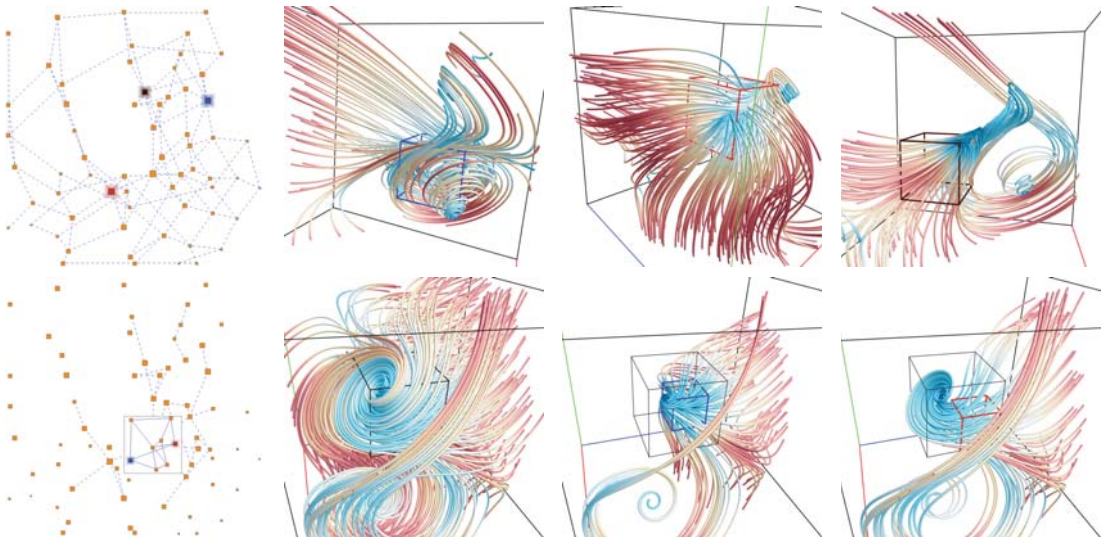
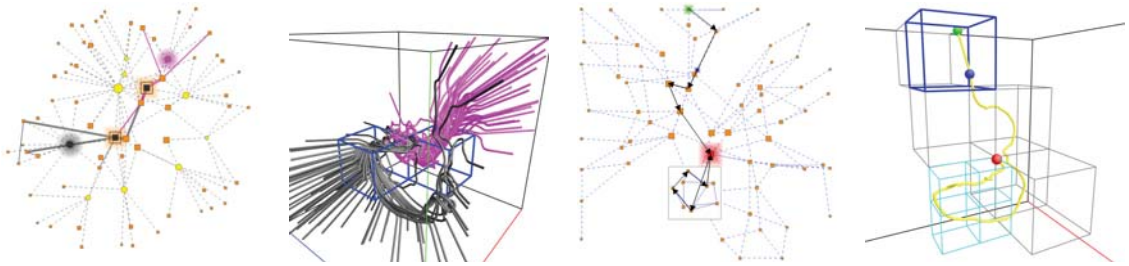


Figure 5.11: Exploration of the five critical points data set (©2014 IEEE).

trace streamlines over the field. For unsteady data sets, seeds are randomly placed at the first time step for pathline tracing. The entropy calculation was performed in the GPU, while FlowGraph construction was performed in the CPU. At runtime, all tasks including graph drawing, layout adjustment and user interaction in both views are interactive.

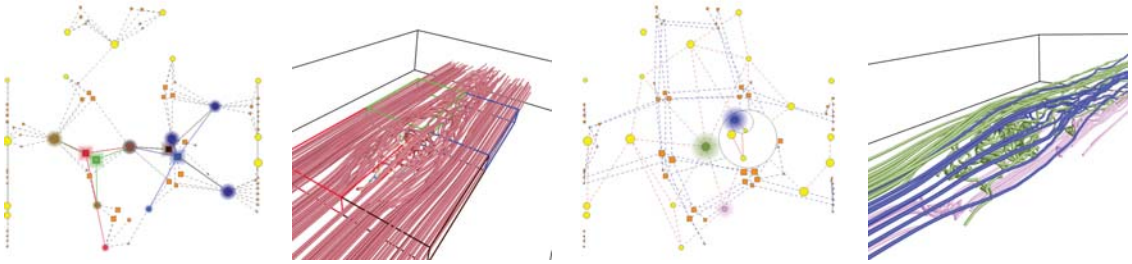
In the following, we present three case studies on three other steady data sets to demonstrate the capability of FlowGraph in assisting flow field exploration, path comparison and feature identification. We also give two case studies on two unsteady data sets to show the exploration of relationships between pathlines and spatiotemporal regions.

Case Study 1 — Five Critical Points Data Set. For the five critical points data set, we experience how we can use FlowGraph to easily identify these critical



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

Figure 5.12: Exploration of the steady supernova data (©2014 IEEE).



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

Figure 5.13: Exploration of the interesting flow pattern in the car flow data set (©2014 IEEE).

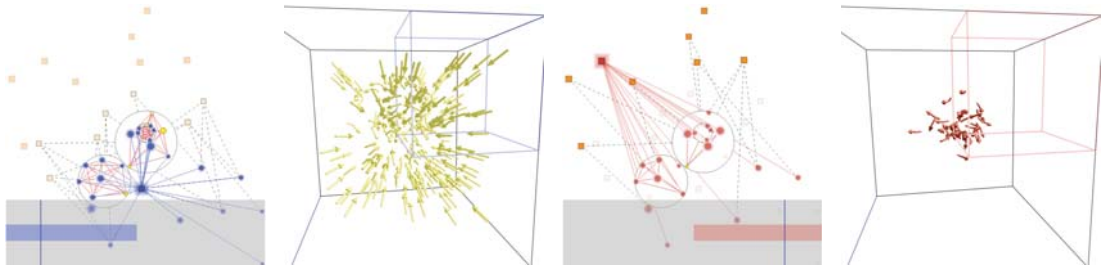
points from randomly traced streamlines that densely cover the field. In the first row of Figure 5.11, we show our exploration results that highlight three spatial regions (shown in blue, red and brown) that contain critical points. These spatial regions are important R-nodes in terms of centrality in the graph view. Normally, these R-nodes are close to the center of the graph and have strong connections to other nodes. As we can see in the streamline visualization, these three regions correspond to a spiral, a saddle and a source from left to right, respectively. In the second row of Figure 5.11, we identify one R-node that has strong connection with its neighbor by filtering R-nodes based on the R-R edge weight. Its corresponding spatial region is close to the center of the volume. The streamlines passing through this R-node

are displayed. Since the number of streamlines displayed is fairly large, we further explore the child nodes of this R-node. Two child R-nodes and the streamlines passing through each of them are shown. It is clear that with the level-of-detail exploration, it becomes convenient for the user to explore the relationships between streamlines and spatial regions in an adaptive manner. This capability is very necessary in order to achieve flexible control when exploring large and complex 3D flow fields where dense streamlines are commonly exhibited throughout the entire volume.

Case Study 2 — Steady Supernova Data Set. For the supernova data set, we first compare the paths of two streamline clusters. As shown in Figure 5.12 (a) and (b), these two streamline clusters (shown in black and magenta) both start from the volume boundary and get more intertwined as they get closer to the center. The compound graph view clearly shows the two R-nodes these two streamline clusters share in common. The highlighted path results also match the spatial arrangement of these two clusters. The paths start from the surrounding of the graph and advance to the center where the two clusters meet at the two spatial regions highlighted. In Figure 5.12 (c) and (d), we switch to the spatial region subgraph and show the snapshot of path animation of a single streamline over spatial regions. Green, red and blue squares (graph view) and spheres (streamline view) indicate the starting, ending and current animation points, respectively. The current animation point is marked in blue. An R-node is further expanded to show the path information in the next level of detail. The corresponding spatial regions are highlighted in cyan. Observe

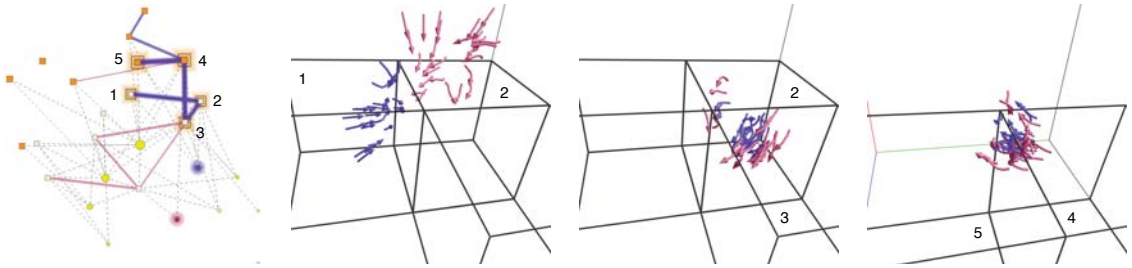
how close the path drawn in the 2D graph view "matches" the 3D streamline view. In general, we find that drawing the subgraph which only consists of R-nodes and R-R edges forms a better arrangement of node positions. This helps the user build the connection between 2D paths and 3D streamlines between the views.

Case Study 3 — Car Flow Data Set. For the car flow data set, our goal is to identify spatial regions and streamline clusters that capture the essential interesting flow pattern passing through the car. In Figure 5.13, we can see that FlowGraph exhibits an interesting layout: many L-nodes and R-nodes are pushed to the boundary of the drawing region. This is due to the fact that many of the streamlines we trace over the volume only form the straight pattern, i.e., they are simply passing by rather than passing through the car. These streamlines and spatial regions surround the interesting flow regions located around the center of the volume. These L-nodes and R-nodes only have a few connections to their neighboring nodes. In contrast, L-nodes and R-nodes around the center of the graph correspond to streamline clusters and spatial regions in the center of the volume. They have more connections to their neighboring nodes and are important nodes for our visual exploration. In Figure 5.13 (a), we select four R-nodes of interest. Eight L-nodes that have strong connections to the selected R-nodes are highlighted. The streamline view shown in (b) clearly indicates the correspondence of these nodes to interesting flow regions. In (c) and (d), we further explore three L-nodes and filter out streamline clusters at two different levels of detail that well capture the flow pattern passing through the



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

Figure 5.14: Exploration of two spatiotemporal regions in the unsteady supernova data set (©2014 IEEE).



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

Figure 5.15: Path comparison for two pathline clusters in the unsteady supernova data set (©2014 IEEE).

car. With the visual guidance of FlowGraph and dual interaction with the streamline view, exploring the underlying flow field to identify features of interest becomes more intuitive, convenient and effective.

Case Study 4 — Unsteady Supernova Data Set. For the unsteady supernova data set, we first utilize the relationships between R-nodes and L-nodes combined with pathlet animation to detect one sink at the core of the supernova. In Figure 5.14 (a) and (c), we highlight two R-nodes (shown in blue and red) which occupy the same spatial region but cover different time spans. The L-nodes connecting to them are also shown in the same color. Two L-nodes are expanded to provide pathline

cluster observation at finer levels of detail. We can see that the blue R-node from the early time span has more connections to the L-nodes than the red R-node from the later time span. This implies that with the time passing by, some pathline clusters which go through one region in the early time steps may no longer be inside of that region later on. One possible answer to this phenomenon is that there is a sink inside of the region. With the help of our pathlet animation, we can verify the correctness of this assumption. In Figure 5.14 (b), we show all the pathlets going through the blue R-node in (a). The colormap shown in Figure 5.8 is used to indicates the time steps of the pathlets. Some pathlets are outside of the blue region because we only show one time step of the animation. Each of these pathlets should go through the region at some specific time step. We can see that most of the pathlets are moving toward the center of the volume. In Figure 5.14 (d), the spatiotemporal region of the red R-node in (c) is shown. It represents the same spatial region in (b) but covers later time steps. The corresponding pathlets going through this region are also shown. It is clear that many of the pathlets disappear at this time step. Based on this observation, we confirm that there is a sink in the center of the supernova where most of the pathlets are trapped.

Figure 5.15 (a) shows the two selected L-nodes and their corresponding paths in blue and purple, respectively. Five R-nodes shared in common are marked with 1 to 5. The clusters (pathlets) and their corresponding paths are shown in the same color. Figure 5.15 (b) to (d) show the moving of the pathlets from two clusters as

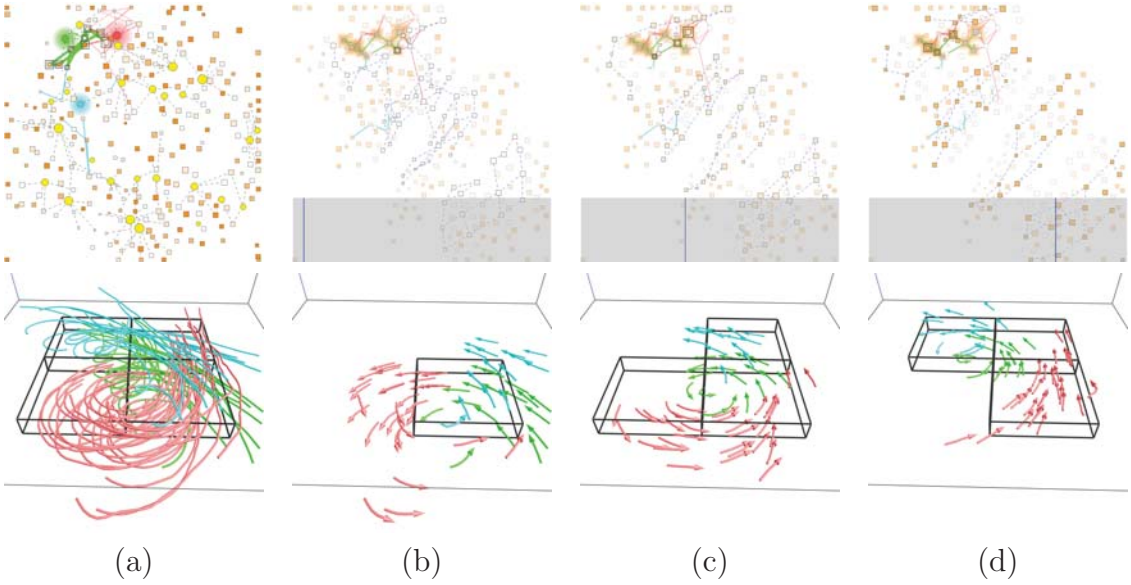


Figure 5.16: Path comparison for three pathline clusters in the unsteady hurricane data set (©2014 IEEE).

the time evolves. The shared R-nodes are also highlighted in both views with their correspondence labeled by number. From the figure, we can see that the flow actually follows a circular pattern around the center of the volume according to the order of the labeled spatiotemporal regions (i.e., from 1 to 5). Note that regions 3 and 4 are the same spatial region but cover different time spans.

Case Study 5 — Unsteady Hurricane Data Set. For the unsteady hurricane data set, we increase the number of nodes in the initial graph for detailed exploration by starting the layout from a finer level of node hierarchy. With the help of FlowGraph, we demonstrate how the trajectory of the hurricane center is detected. Figure 5.16 (a) shows path comparison for three L-nodes (shown in green, cyan and red) in the compound graph and the corresponding pathlines of the selected L-nodes. The

R-nodes shared by the three paths are also highlighted in both views. Figure 5.16 (b), (c) and (d) show the R-node subgraph layouts for three selected time steps, respectively. The timeline in the timeline bar indicates the current time step. R-nodes whose time spans do not cover the current time step are drawn semitransparently for clear observation. The shared regions in the current time step are also highlighted in the pathline view. One interesting finding is that R-nodes in the subgraph are grouped into four well-isolated clusters. Actually, these four clusters form the four horizontal layers along the z dimension of the data set. This indicates that the flows of hurricane almost only move along the xy plane and there is little exchange of flows vertically. In the corresponding three snapshots of pathlet animation, the shared R-nodes are highlighted as the black spatiotemporal regions. We can see that the pathlets follow the hurricane center. With the evolution of time, the trajectory of hurricane follows the order of the shared regions from the lower-right corner of the volume to the upper-left corner.

5.7 Empirical Expert Evaluation

To evaluate the effectiveness of FlowGraph from a practical aspect, we collaborated with a domain expert in biofluids and biomedical engineering, Professor Jingfeng Jiang. Professor Jiang's research focus is on transforming raw biomedical imaging data into useful clinical parameters, such as blood flow characteristics. The evaluation

consists of two major stages. In the first stage, we demonstrated the functions of FlowGraph with several steady and unsteady flow data sets and helped him get used to user interface and interactions of the program. In the second stage, we provided several other flow data sets for the expert to freely explore by himself. For each data set, we set one or two tasks (such as finding critical points, comparing paths or regions, or identifying flow patterns) for him to fulfill with the help of FlowGraph functions. The following is a summary of the feedback.

In general, FlowGraph is a useful and novel tool to explore flow field. It is very helpful in terms of finding the critical patterns within the regions of interest. For simple data sets, the correlation between the graph view and the field line view works well and the connectivity between nodes can show the flow direction clearly. For complex flow fields, adding visual aids will help the user quickly grasp the graph. The node and edge filtering function allows the user to reduce the number of node connections which is crucial for locating important nodes in a complex graph. For unsteady flow field exploration, the spatial constraint of pathlines enables the user to visualize self-contained particles by the spatial regions. The temporal constraint of pathlines helps the user easily separate slow particles from fast particles if all of them are released from the same position.

FlowGraph may be directly applied to visualization of cardiovascular flows. The technique will probably work well with the heart and aneurysms. Particularly, the

Raynolds number of the physiological flow in the heart is large so that the flow is highly disturbed. Therefore, using streamline and pathline clustering and visualization may help the user track hierarchical structures of the flow. The other potential application is to track diffusion flows—another important application in biomedical engineering. FlowGraph for unsteady flow fields provides the time-resolved information. This can help the user observe the correlation between the particle moving and its residence time, which is relevant to large protein accumulation and subsequent biological effects, e.g., clotting and inflammatory responses. Path comparison will help the user study flow mixing. More specifically, if each cluster represents a source of incoming flow, flow paths can visualize how the mixing of flow takes place. Drug delivery will be a good application for this function.

Chapter 6

FlowTour: An Automatic Guide for Exploring Internal Flow Features

6.1 Motivation and Goals

Flow field exploration provides a convenient way for the user to observe and understand the underlying flow flied patterns and features. In the previous chapters, we introduced several approaches to assisting the user to explore flow fields, such as

[0]The material contained in this chapter was previously published in *IEEE Pacific Visualization Symposium 2014*.

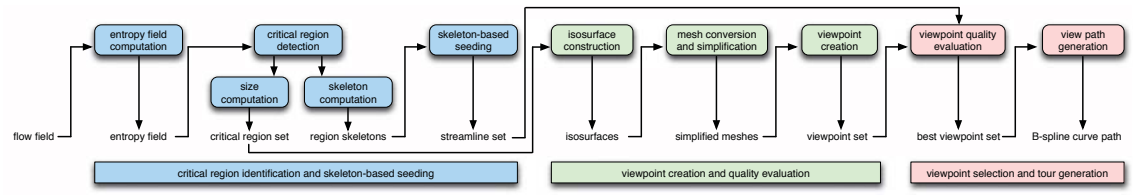


Figure 6.1: The overview of our three-stage algorithm (©2014 IEEE).

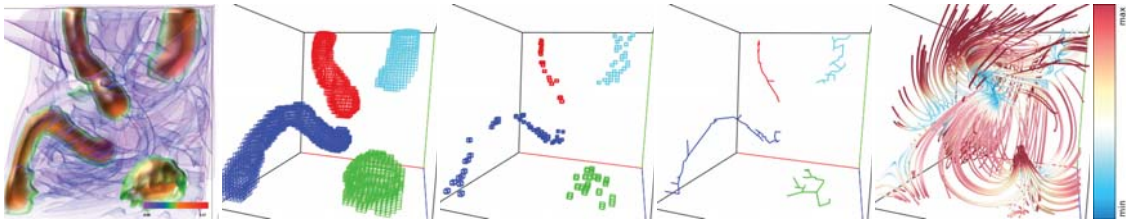
streamline selection, viewpoint selection and graph-based representation. In terms of viewpoint selection, most of such techniques only focus on external viewpoints which are normally selected from the volume's bounding sphere and look at the center of the volume, just like described in Chapter 3 and Chapter 4. This will not convey a clear observation when the flow field is clutter on the boundary side and prevent the user from detecting flow patterns.

In this chapter, we define the *internal viewpoint* as the viewpoints inside of the flow field and introduce our FlowTour [54] (©2014 IEEE), a novel framework that provides an automatic guide for exploring internal flow features. This work has been published in *IEEE Pacific Visualization Symposium 2014*. All figures used in the chapter are from the original publication. Our algorithm encompasses feature identification, streamline placement, viewpoint selection and tour generation into a single and unified framework. Since our viewpoint selection places its focus on internal viewpoints, the final tour going through multiple critical regions is similar to a roller coaster tour in an amusement park which flies through the scene. In this way, we give the user closeup views of the flow field for detailed observation of hidden or occluded internal flow features and patterns.

6.2 Approach Overview

The framework of our algorithm is shown in Figure 6.1. There are three major stages: critical region identification and skeleton-based seeding, viewpoint creation and quality evaluation, and viewpoint selection and tour generation. Each stage consists of several substeps. At the first stage, given an input 3D flow field, we compute its entropy field and identify *critical regions* which correspond well to interesting flow features and patterns such as the vicinities of critical points. We detect large critical regions for the flow field and compute an isosurface and a skeleton for each of them. A *skeleton-based seeding* algorithm is carried out to purposefully generate a set of streamlines for the subsequent viewpoint evaluation and tour design.

At the second stage, the isosurfaces of critical regions are first converted into a triangle mesh to obtain their surface connectivity information. Then we simplify the mesh and initialize vertices on the simplified mesh as candidate viewpoints associated with the critical regions. A series of *offset viewpoints* with different zooming levels is computed for each viewpoint to construct a *viewpoint set*. We evaluate the quality of viewpoint by considering how much information of the streamlines seeded from the corresponding critical region could be revealed. We also consider *foreground streamline occlusion* and *background streamline noise* in the evaluation.



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

Figure 6.2: (a) the entropy field of the five critical points data set. (b) the critical regions identified from the entropy field. (c) and (d): skeleton points and lines extracted from critical regions, respectively. (e) the streamlines seeded along the skeletons (©2014 IEEE).

At the last stage, we select one viewpoint with the highest quality value as the *representative* for each viewpoint set. We then pick several best viewpoints from all the representative viewpoints for the corresponding critical region. There are two criteria for selecting the best viewpoints. First, their quality should be high. Second, the distance between any two best viewpoints should be sufficiently large. The final view path is constructed by interpolating a cubic B-spline curve traversing all viewpoints. For all critical regions in the field, a global B-spline curve path traversing all these regions is generated by picking the one that has the minimal *cost of traversal*.

6.3 Critical Region Identification and Skeleton-based Seeding

Entropy Field Computation: Refer to Appendix A.2.2 for the computation of the entropy field for a flow field. Figure 6.2 (a) shows the entropy field for the five critical

points data set.

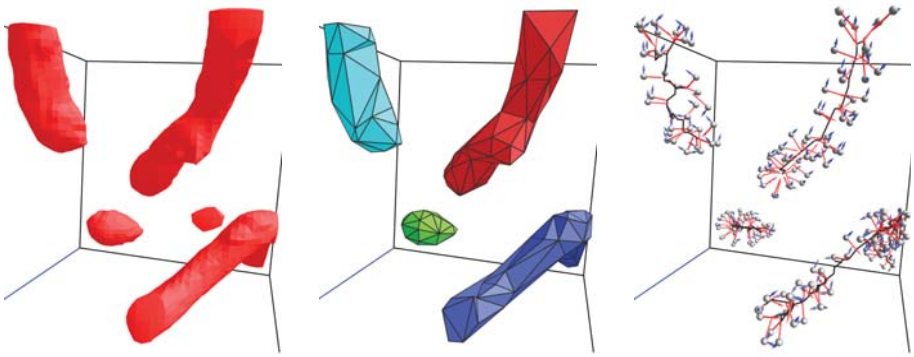
Critical Region Detection: With the entropy field derived, we define *critical regions* in the volume as local neighborhoods in which all the voxels' entropy values are greater than a given threshold. Intuitively, a critical region is a sub-volume in the flow field which contains rich information compared with the remaining non-critical ones.

- Region size computation: Since the shape of a critical region may not be regular, its size also varies dramatically. In our algorithm, we do not consider the regions with small volume size and they are filtered out from the *critical region set R*. In order to compute the size of a critical region, we apply a region growing algorithm which approximates the region's volume at the voxel level. The algorithm works as follows: we first randomly pick one voxel inside of the flow field and check if its entropy value is greater than the given threshold δ_e. If it is false, we pick another one. Otherwise, this voxel must be in one critical region and we mark it with a volume ID. We start growing the region from this voxel by checking all its neighboring voxels and push the voxels with their entropy values greater than δ_e into a queue. Next, we dequeue one voxel from the queue and apply the region growing process to this voxel until the queue is empty. The size of the critical region is defined as the number of marked voxels. We apply the same process until we compute the sizes for all critical regions. An

example is shown in Figure 6.2 (b).

- Region skeleton extraction: In order to identify the shape pattern for each region r in R, we extract its *skeleton* by adopting a volume thinning algorithm developed by Gagvani and Silver [28]. The basic idea of this method is to compute the *distance transform* which is the distance from the internal voxel of the region to the boundary voxel. Skeleton points are those whose distance transform values are larger than a given *thinness parameter* δ_t, as shown in Figure 6.2 (c). Furthermore, by applying a *minimum spanning tree* (MST) algorithm to the skeleton points, we can eventually connect all skeleton points to form a tree-structured skeleton line, as shown in Figure 6.2 (d). Next, we identify two endpoints on the skeleton which have the longest Euclidean distance and define the *major direction* of the skeleton as a vector starting from one endpoint of the skeleton with lower y value to the other one. Skeleton extracting plays an important role in our algorithm since it not only indicates the shape pattern but also provides a central curve to focus on for all viewpoints associated with the critical region. As a matter of fact, the `look-at` centers of viewpoints for a critical region will always be positioned on its skeleton.

Skeleton-based Seeding: We adopt a *skeleton-based seeding* strategy by always dropping seeds along the skeleton of each critical region. In this way, we guarantee that all critical regions are well covered by streamlines. This strategy also helps reduce



(a) (b) (c)

Figure 6.3: (a) the isosurfaces constructed for the five critical points data set. (b) the simplified triangle meshes constructed from the isosurfaces in (a). (c) all viewpoints generated on the simplified mesh surface (©2014 IEEE).

redundant streamlines in uninteresting regions. Figure 6.2 (e) shows one example of the streamlines generated using skeleton-based seeding.

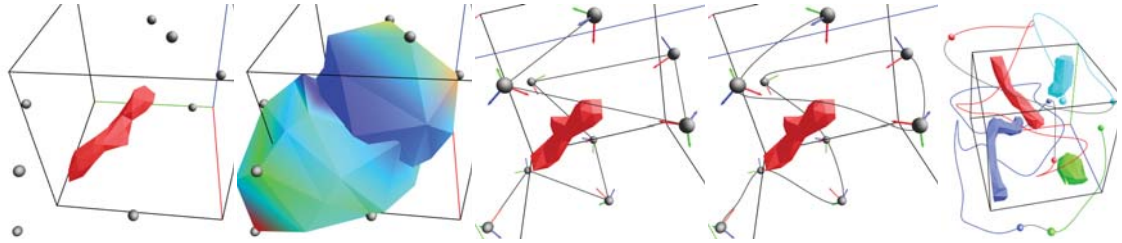
6.4 Viewpoint Creation and Quality Evaluation

Isosurface Construction: In volume visualization, isosurfaces are usually used to represent surfaces inside of a volume whose scalar value equals a given threshold, which is called the isovalue. We leverage the isosurface to indicate the shape and location of each critical region and define the isovalue as the given entropy threshold δ_e (Section 6.3). To obtain the isosurface, we use the classical marching cube algorithm [53]. Figure 6.3 (a) shows an example of the constructed isosurfaces. Similar to entropy field computation, for a large input data set which cannot be loaded int the memory once, we leverage CUDA to extract the isosurfaces block by block using

GPUs. We also utilize a k-d tree data structure to store all the resulting triangles for fast access. The marching cube algorithm only produces isolated triangles and no geometric connectivity information is readily available for use. Therefore, we convert the isosurfaces into triangle meshes.

However, since the isosurface construction is processed in the voxel level, there may be more than thousands of vertices in each mesh. To reduce the number of vertices to a manageable level, we apply a mesh decimation algorithm based on edge collapse introduced by Hoppe [36]. A *decimation factor* δ_s is provided to the user for controlling the simplification level of the final mesh. For our application, it is desirable to keep the final number of vertices on the simplified mesh surface to a few hundred. Figure 6.3 (b) shows an example of the simplified meshes. In our work, mesh conversion and decimation is performed using the open source OpenMesh library [2].

Viewpoint Creation: Given the critical region set R obtained in the first stage, our algorithm creates a list of viewpoints. First of all, by locating the viewpoints at the vertices on the simplified mesh surface, we obtain a set of viewpoints S. For each viewpoint in S, we compute its `look-at` center and `up` direction based on the region skeleton. For the `look-at` center, we compute the distance between the viewpoint location and the corresponding skeleton line. The point on the skeleton line closest to the viewpoint location is the `look-at` center for this viewpoint. This treatment guarantees that our viewpoint always focuses on the portion of critical region closest



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

Figure 6.4: (a) the best viewpoints for one critical region of the five critical points data set. (b) the corresponding offset surface where color mapping indicates viewpoint quality. (c) the straight view path by connecting selected viewpoints in (a). (d) the B-spline curve path generated from (c). (e) the final global B-spline curve path traversing all critical regions (©2014 IEEE).

to it. The `look-at` direction **l** is determined as the vector from the viewpoint location to the `look-at` center. To compute the `up` direction, we utilize the skeleton's major direction as a guidance. Specifically, we define the local skeleton direction **d** at the `look-at` center as the vector along the skeleton which starts from the `look-at` center and points toward the skeleton's major direction. We then project **d** onto a plane perpendicular to **l**. The final `up` direction is the projected vector on the plane. In Figure 6.3 (c), we show all the viewpoints generated at the vertices on different surfaces. The corresponding `look-at` and `up` directions are also displayed in red and blue.

Once we finish computing each viewpoint v in S, we also generate several *offset viewpoints* associated with v by offsetting v along the opposite direction of its `look-at` direction **l** for some levels. Offset viewpoints share the same `look-at` center and `up` direction with v and their locations are simply pushed away from v. Intuitively, each offset viewpoint of v is a zoom-out view. We define the offset viewpoints along with v

as a *viewpoint set* V. For each viewpoint in V, we evaluate its quality and then select the viewpoint with the highest quality as the *representative* of V. This procedure is applied to all viewpoint sets on the mesh surface. We then connect the representatives by following the original mesh connectivity information to form a new *offset surface*. Figure 6.4 (b) shows such an example.

Viewpoint Quality Evaluation: For each critical region r, we evaluate the quality of viewpoints associated with it based on the amount of information revealed from streamlines seeded from its skeleton. We also consider foreground streamline occlusion and background streamline noise as penalties to avoid visual clutter and distraction. Specifically, we utilize the *mutual information* $I(X;Y)$ between 3D streamline X and its 2D projection Y as the measure of information revealed (Appendix A).

By utilizing the streamline mutual information, we compute the final *viewpoint quality* for a viewpoint v as follows

$$Q(v) = S_{\text{focus}} - (P_{\text{fore}} + P_{\text{back}}), \tag{6.1}$$

where S_{focus}, P_{fore}, P_{back} are *focus region score*, *foreground occlusion penalty* and *background noise penalty*, respectively. S_{focus} indicates how much information revealed by the streamlines seeded from the corresponding critical region is preserved under v. Since our viewpoints are mostly located inside of the flow field, it is inevitable that some streamlines seeded from other critical regions will block our view when we look

at the critical region r in focus from a given viewpoint v. Additionally, for streamlines behind r, they would potentially distract our attention by adding some "noise" in the final image. We quantize these two effects by defining P_{fore} and P_{back} and select viewpoints with low values from these two terms. We compute these two terms in a single phase. First, we transform the standard OpenGL view projection plane with dimension of 2×2 to a predefined $n \times n$ projection plane $\mathbf{P}$. We then record the minimum Z value for each pixel in $\mathbf{P}$ covered by streamline segments computed in S_{focus}. For pixels not covered, we set an infinitesimal value for them. Next, for each streamline s seeded out of r, we check the Z value for every point along s inside of the viewing frustum and compare it with the Z value of the corresponding pixel in $\mathbf{P}$. If the new value is larger than the one in $\mathbf{P}$, we set that value to $\mathbf{P}$ and mark the point as an occlusion or noise point. If the point is between the viewpoint v and our critical region r in focus, it is an *occlusion point*. If it is at the back of r, it is a *noise point*. P_{fore} and P_{back} are obtained as the summation of the MI values of these two kinds of points, respectively.

6.5 Viewpoint Selection and Tour Generation

Best Viewpoints Selection: At the end of viewpoint creation step, we construct a new offset surface by connecting all representative viewpoints from each viewpoint set V. Next, we sort all these representatives by their quality and pick the final

best viewpoints with the highest values. However, if we take the quality value as the only criterion for best viewpoints selection, neighboring viewpoints with similar high values will be selected together. To avoid this, we define the distance between two viewpoints and leverage this measurement to keep any two selected best viewpoints different enough from each other. In Figure 6.4 (a) and (b), we depict how the best viewpoints are arranged in space and on an offset surface, respectively.

Tour Path Generation:

- **Straight path generation:** From a set of best viewpoints selected, we form a path traversing all of them using line segments. In order to guarantee that the path follows the skeleton's shape pattern, we utilize the skeleton's major direction as a guidance for path generation. Specifically, we pick the viewpoint whose center is closest to one end of the skeleton's major direction as the starting point of the view path and set it as a *pivot viewpoint* v_p. We then connect the viewpoint v whose `look-at` center is closest to the current v_p's `look-at` center (i.e., $d_c(v_p, v)$ is the smallest) and set it as the new v_p. To avoid the zig-zag path shape, we force the angle formed by every three consecutive viewpoints along the path to be larger than a given threshold δ_α. Figure 6.4 (c) shows an example of the straight path for the five critical points data set. One drawback of using the straight-line as the view path is that in some cases, line segments may intersect with the skeleton (i.e., the view path will get fairly close to the

flow feature). To avoid this, we replace the straight-line with a B-spline curve to "push" the path away from the skeleton when they are too close to each other.

- **B-spline curve path generation:** In order to guarantee that the view path could always keep some distance away from the corresponding critical region's skeleton, we add one intermediate viewpoint between each pair of adjacent viewpoints along the straight path. Essentially, for each line segment along the straight path, we compute one intermediate viewpoint which always keeps some distance from the skeleton. Let the point on the line segment be p_l and the point on the skeleton be p_s. We push p_l away from the skeleton along the direction of vector $\overrightarrow{p_s p_l}$ for some distance Δ_d, where Δ_d is inversely proportional to the shortest distance between the segment and the skeleton ($d_{l,s}$). Formally, we define $\Delta_d = C/d_{l,s}$ where C is a parameter to control the inverse proportion weight between Δ_d and $d_{l,s}$. In this work, we use 0.5 for all data sets. Intuitively, p_l is far away from the skeleton if its corresponding line segment is close to the skeleton. After creating all intermediate viewpoints, a B-spline curve path which traverses all viewpoints including intermediate ones is interpolated. Figure 6.4 (d) shows an example of the final curve path. In order to maintain a smooth animation when we move viewpoints along the path, we also interpolate several viewpoints between each two adjacent best viewpoints with equal arc length.

- **From single region to multiple regions:** The preceding algorithm operates on a single region. If there are more than one critical regions in the field, a global

B-spline curve path traversing all these regions is generated. To achieve this, we first compute and sort the best viewpoints for each critical region as we describe in the straight path generation phase. Next, we order all the regions and define the *cost of traversal* between two regions. By utilizing this measurement, we construct the final global path which could traverse all the best viewpoints in a smooth and efficient way. Specifically, we define the *cost of traversal* $C(r_1, r_2)$ from region r_1 to r_2 by considering two factors: the distance $D(r_1, r_2)$ between the positions of the last viewpoint of r_1 and the first viewpoint of r_2, and the angle $A(r_1, r_2)$ between the `look-at` directions of these two end viewpoints. That is,

$$C(r_1, r_2) = D(r_1, r_2) * A(r_1, r_2), \tag{6.2}$$

where both distance and angle are normalized by their corresponding maximum values. Intuitively, we connect two regions if their end viewpoints are spatially close to each other and their angle change is small. The cost of the global path is obtained by adding up the costs of traversal for all the pairs of critical regions traversed in order. We compute the costs for all possible region orderings, i.e., for each critical region. From all these orderings, we select the final global path as the one with the smallest cost value. Finally, a global B-spline curve is interpolated by considering all the viewpoints on the global path as data points. Figure 6.4 (e) shows an example of the global path.

Table 6.1
The parameter settings for seven flow data sets (©2014 IEEE).

data set	dimension	entropy δ_e	thickness δ_t	decimation δ_s	distance δ_d	angle δ_α
five critical pts	$51 \times 51 \times 51$	2.845	0.7	0.320	10.0	$\pi/4$
two swirls	$64 \times 64 \times 64$	2.930	0.7	0.328	10.0	$\pi/4$
tornado	$64 \times 64 \times 64$	2.098	0.7	0.328	10.0	$\pi/6$
supernova	$200 \times 200 \times 200$	2.459	0.9	0.335	15.0	$\pi/3$
solar plume	$126 \times 126 \times 512$	2.939	0.9	0.335	12.0	$\pi/3$
ABC flow	$64 \times 64 \times 64$	2.306	0.7	0.320	10.0	$\pi/4$
electron	$64 \times 64 \times 64$	2.092	0.7	0.320	10.0	$\pi/4$

6.6 Viewpoint Traversal and Path Animation

Given the final global B-spline tour path, we traverse all viewpoints by moving the camera along the path. Whenever there is an abrupt change of viewing angles between two adjacent viewpoints, we interpolate intermediate viewpoints for a smooth transition. We render streamlines as tubes. To help the user focus on the currently traversed region, we render the streamlines seeded from the current focus region with a large tube radius and all other streamlines with a small tube radius. When the camera focus changes from one critical region to another, an animated transition indicating the changes of streamline thickness is shown. In the user study, we also provide the user with the freedom to change the animation speed, pause the animation, or play the animation in reverse order so that they can observe critical regions in a more flexible manner.

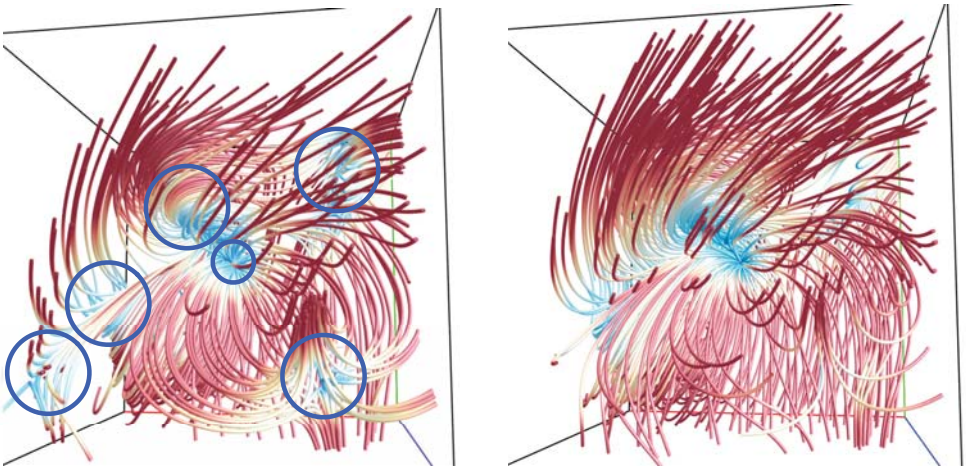
Table 6.2
The timing results for seven flow data sets. A * denotes out-of-core processing in the GPU using CUDA (©2014 IEEE).

data set	entropy field	critical region	initial #lines	isosurface	mesh	viewpoint creation	#best viewpoints/ #total viewpoints	viewpoint evaluation	tour path
five critical pts	0.47s	0.12s	800	0.085s	0.14s	0.02s	27/175	106.07s	0.08s
two swirls	0.51s	0.16s	500	0.16s	0.40s	0.03s	25/239	218.29s	0.07s
tornado	0.54s	0.22s	400	0.17s	0.18s	0.03s	12/244	98.95s	0.02s
supernova	7.89s*	570.83s	400	3.95s*	4.33s	0.14s	10/766	266.47s	0.01s
solar plume	8.25s*	494.91s	400	4.07s*	7.75s	0.13s	15/751	381.42s	0.02s
ABC flow	0.61s	0.13s	1000	0.17s	0.15s	0.01s	12/157	65.09s	0.09s
electron	0.56s	0.06s	450	0.16s	0.06s	0.01s	7/68	10.75s	0.01s

6.7 Results

6.7.1 Configurations

We implemented FlowTour on a CPU-GPU hybrid platform with the same hardware configuration used in Chapter 4. Entropy field computation, isosurface extraction, and viewpoint quality evaluation were implemented in the GPU using CUDA and all other computations were implemented in the CPU. Since we changed streamline thickness frequently, we utilized the *vertex buffer object* (VBO) to render streamlines and used the GPU to process their geometry changes in order to provide smooth streamline update. The timing results and parameter settings for the seven data sets we used are reported in Tables 7.2 and 7.1. All stages of processing can be finished within 15 minutes for each data set.



(a) (b)

Figure 6.5: (a) skeleton-based seeding. (b) random seeding (©2014 IEEE).

6.7.2 Skeleton-based Seeding vs. Random Seeding

In Figure 6.5, we compare our skeleton-based streamline seeding with random seeding. Clearly, our method conveys more information of the original flow field than random seeding since most of the streamlines in our method are located around the critical regions (highlighted with circles in Figure 6.5 (a)) which are the most interesting areas of the flow field. Furthermore, unlike random seeding which places streamlines arbitrarily in the field, our method also reduces streamline occlusion since we never put seeds in uninteresting regions. This would help the user observe flow field patterns in a less ambiguous way.

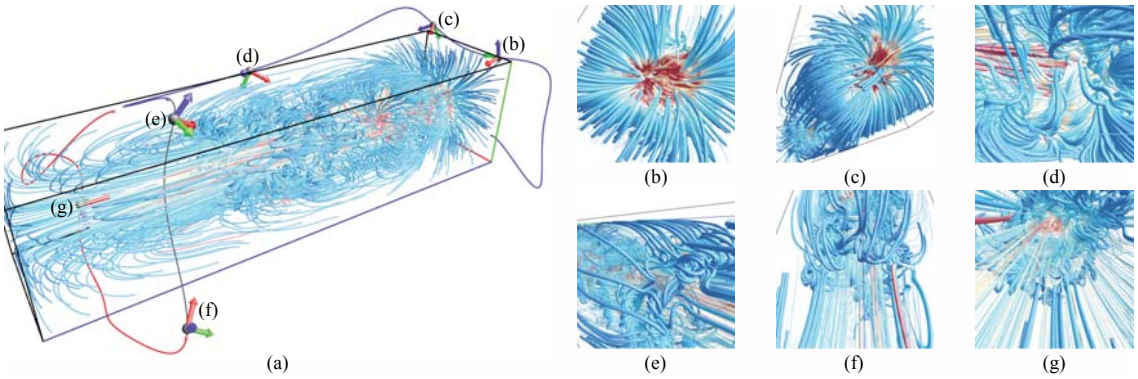


Figure 6.6: Screenshots of our internal view exploration for the solar plume data set (©2014 IEEE).

6.7.3 Tour Path Exploration

Figure 6.6 shows the final B-spline tour path and six screenshots along the internal view exploration for the solar plume data set. As we see in Figure 6.6 (a), the curve tour provides a smooth traversing path which effectively covers most features of the flow field. Figure 6.6 (c) depicts the major flow pattern, i.e., the head of the solar plume, which provides the user with a good overall view of the flow field. This was made possible with our offset viewpoints which make sure that the viewpoints are not too close to the scene. Figure 6.6 (b) shows the zoom-in effect of the head of the solar plume. The "flower"-like pattern is clearly depicted and the velocity variation around the core of the head is also nicely revealed. In Figure 6.6 (d), some small spiral patterns inside of the head of the solar plume are shown. Since there are many streamlines around this region, it is difficult for the user to observe such features from external views. In Figure 6.6 (e) and Figure 6.6 (f), some detailed patterns such as

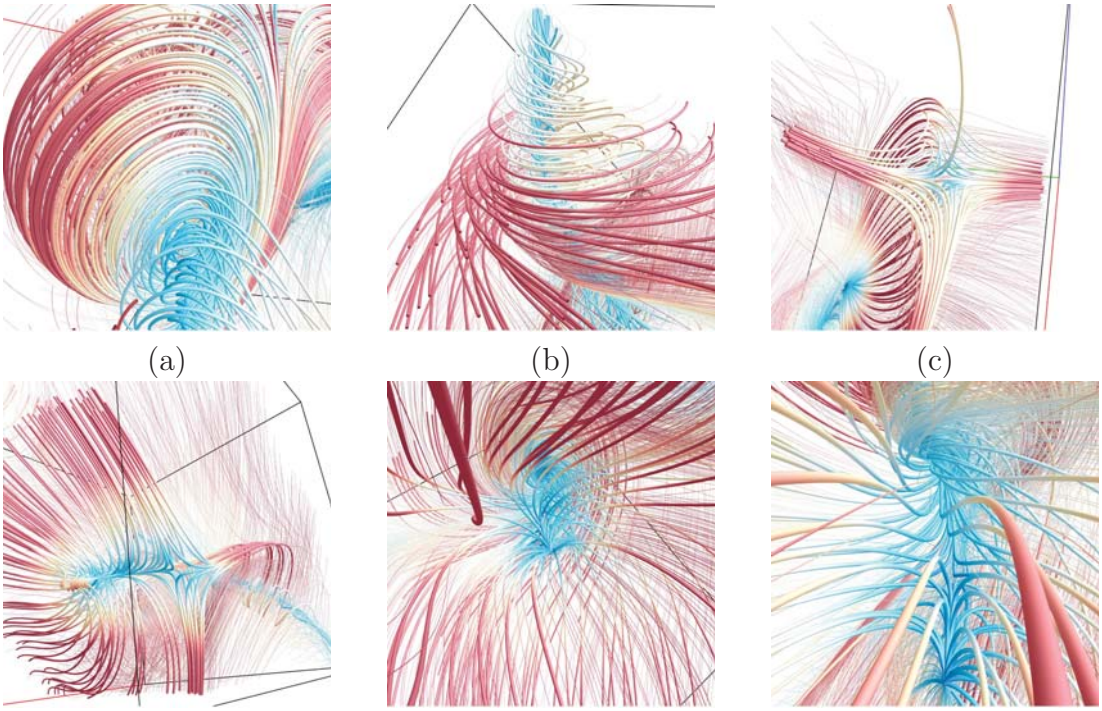


Figure 6.7: Screenshots of the internal view exploration for the five critical points data set (©2014 IEEE).

small spirals around the straight lines in the middle portion of the flow field are also clearly captured from two different viewpoints. Instead of always forcing the user to look at the flow field from outside, our tour path can also take the user to the "kernel" of the flow field and provide an expressive traversal experience which is not available using external view exploration. Figure 6.6 (g) gives such an example. The user can now clearly observe the flow patterns of the internal hollow shaft by "standing" right inside of it.

In Figure 6.7, we show the results of our internal view exploration for the five critical points data set. The five critical points are clearly shown (two spirals in (a) and (b),

two saddles in (c) and (d), and a source in (e)). Furthermore, our tour also captures the connection between the source and a spiral as shown in (f). Since this connection is in the center of the flow field and is occluded by many surrounding streamlines, it is more difficult to detect such a connection if we only use external views.

6.8 User Study

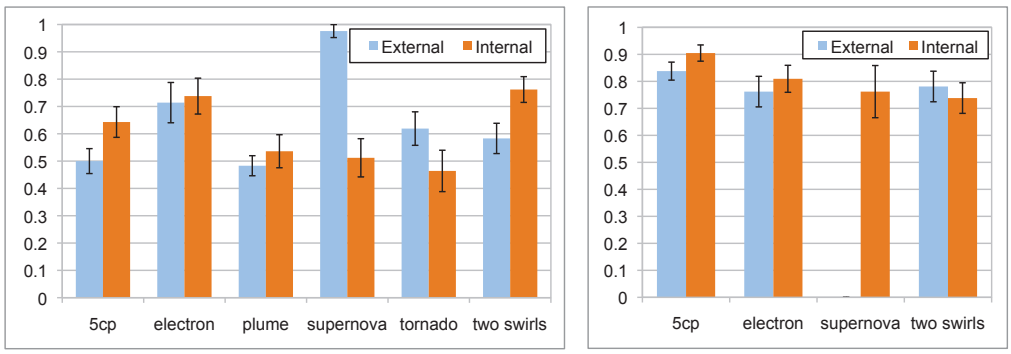
We conducted a user study to evaluate the effectiveness of the approach we have achieved by cooperating with James Walker, a PhD student and his advisor Dr. Kuhl. We used a design of 2 conditions (external tour vs. internal tour) $\times 2$ tasks (answering questions and identifying critical regions). Totally 21 new users were recruited to participate in the actual experiments: twenty of them were undergraduate or graduate students, and one professor from the computer science department. All experiments were conducted in our graphics and visualization laboratory using two standard desktop PCs with the same configuration. For each experimental session, users were shown seven flow field data sets (one for practice and six for evaluation). For each data set, users were first shown a tour of external views of the flow field, followed by a tour of internal views. After each tour, the users were asked several multiple-choice questions about features in the flow field, and were then asked to identify as many critical points as they could find within a set time limit. The questions asked were the same for both tours. After completing both tours of external

and internal views for a given data set, users were asked to give a written response whether they felt that certain questions were easier to answer with external or internal views. We hypothesized that some fine details of the flow fields, especially internal features, would be correctly identified with higher probability using internal views.

6.8.1 Experimental Procedure

For external or internal views, the user was shown an animation of the complete path through the data set. The speed of the animation could be adjusted using a slider if the user felt it was too fast or slow. After the animation, the user was presented with several questions and then asked to identify critical regions in the data set. The user had a limited time to perform these tasks. While performing these tasks, the user could revisit any part of the tour path using a slider. This functionality was useful for answering the questions and required to identify critical regions. After completing both tours of external and internal views, the user answered one final question requiring a written response before moving on to the next data set.

Users completed all seven data sets in one sitting. The entire experiment took approximately 90 minutes for most users, including initial paperwork, briefing, and the post-experiment questionnaire. The questionnaire asked the user to rate the comparative effectiveness of tours using internal and external views in several categories,



(a) (b)

Figure 6.8: (a) average proportion of correct answers of multiple-choice questions. (b) average proportion of critical regions identified correctly (©2014 IEEE).

and also asked for subjective feedback and suggestions for improvement regarding the experiment and the program's user interface.

6.8.2 Results and Discussion

We present the results from this study in four aspects: answer correct rate on multiple-choice questions, proportion of critical regions correctly identified, user responses to subjective questions, and differences when users are separated by expertise. We used two hypothesis tests to analyze statistical significance between the two conditions, ANOVA (ANalysis Of VAriance) and the Friedman non-parametric repeated measures test, with a standard significance level $\alpha = 0.05$. The two tests are applied to all four aspects and the null hypothesis H_o for all tests is $mean_{internal} = mean_{external}$ for each aspect. Here $mean_{internal}$ and $mean_{external}$ are the mean values for internal group and external group, respectively.

Table 6.3
Average subjective user scores of external vs. internal views (©2014 IEEE).

	External views	Internal views
Easy to find features	3.429	3.762
Fun/enjoyable to use	3.810	3.524
Understood flow fields well	3.714	3.762

Multiple-choice questions: The results are given in Figure 6.8 (a). Users performed better with external views in the supernova and tornado data sets, with the Friedman test yielding p-values of 0.0002747 and 0.02014, respectively. Users performed better with internal views in the five critical points and two swirls data sets. For the two swirls data set, the Friedman test yielded a p-value of 0.000532. The five critical points data set is the only case where the ANOVA and Friedman tests did not agree on statistical significance with a significance level $\alpha = 0.05$ (the ANOVA yielded a p-value of 0.0488 and Friedman yielded a p-value of 0.1083).

Critical regions identification: The analysis was done by comparing how many critical regions the user correctly identified compared with the total number of critical regions presented in the data set, such that a value of 1.0 means the user found every critical region. The results are given in Figure 6.8 (b). The solar plume and tornado sets were excluded from this analysis. Users successfully identified more critical regions with internal views in the five critical points and supernova data sets. The Friedman test yielded p-values of 0.03481 and 0.00006334, respectively. Neither electron nor two swirls exhibited statistical significance.

Subjective responses: In the post-experiment questionnaire, users were asked to rate various aspects of the external vs. internal views on a scale from 1 (strongly disagree) to 5 (strongly agree). The average responses are given in Table 6.3. No statistically significant differences were observed between any of the scores.

Effects of user expertise: To further analyze the data, we divided users into two groups: users who self-identified as having no or low familiarity with flow fields, and users who self-identified as being experts with flow fields. We then performed the same analyses described above to see if there were any major differences between experts and non-expert groups. In most cases, the results were the same. However, we identified two important exceptions. First, expert users exhibited a statistically significant difference in their subjective preference between external and internal views. Expert users preferred internal views for finding flow features with a p-value of 0.03524. Second, expert users showed greater resilience to the difference in views when answering questions about the tornado data set. Overall, users performed better with external views in this data set, but that is not the case with expert users. Expert users showed no statistically significant difference in their performance answering questions on the tornado data set between the external and internal conditions.

Discussion:

Based on a comprehensive statistical analysis, we found that for certain data sets, the internal flow tour exposes internal details which are not visible using external views

alone. With an internal tour, users are able to better discern complex internal details and find critical regions which are obscured from outside the flow field. However, internal exploration is not always appropriate, depending on the data set. For very simple data sets, or data sets which have discernible external patterns but exhibit only chaotic turbulence on the inside, the usefulness of internal views is limited. Even in these cases, however, internal views may still be necessary to identify internal hidden critical regions.

Lastly, we discovered that expert users preferred the internal view for finding flow features. Additionally, expert users performed better than non-experts when using the internal view to answer questions about the tornado data set. These findings suggest that expertise with flow fields is helpful for users to fully realize the benefits of using the internal flow tour.

Chapter 7

Moving with the Flow: An Automatic Tour of Unsteady Flow Fields

7.1 Overview

FlowTour described in Chapter 6 provides an effective way to help the user explore internal flow features automatically. However, it only considers three-dimensional

[0]The material contained in this chapter has been submitted to *Transactions on Visualization and Computer Graphics 2015*.

steady flow fields, leaving the design of an automatic tour for exploring three-dimensional unsteady flow field unsolved.

Unlike a steady flow field where the traced streamlines are steady over time, pathlines or pathlets traced over an unsteady flow field move or change over time. This poses several unique challenges which call for a new solution for designing an automatic tour for an unsteady flow field. First, for a steady flow field, we only need to place seeds once to generate streamlines which capture critical flow regions. For an unsteady flow field, we need to carefully place seeds over time to capture different critical flow regions at different time steps. Using pathlet animation, we also need to make sure that the *density of pathlet* is appropriate and varies smoothly over space and time. Therefore, new seeds need to be placed in subsequent time steps to highlight new critical regions. We also need to place additional seeds to account for disappearing pathlines which go beyond the domain boundary. Second, in the tour animation, since all streamlines remain unchanged, the shifting from one region of focus to another can be solved straightforwardly by simply considering how smooth the transition will be. For an unsteady flow field, we assume that the animation follows the order from the first time step of the data to the last time step. Since all pathlets change along the animation, we need to solve the issue of *dynamic shifting of focus* as the critical regions may merge or split over time. To address this issue, we need to first identify the correspondence among critical regions over time, then determine an optimal traversal order of these critical regions. Third, placing cameras along the tour should take

into account the size, orientation and life span of each critical region of focus so that the tour would highlight different critical regions *at the right time* and *for a right duration*. Only by doing so can we produce a tour path that is informative, i.e., capturing important time-dependent features along the tour to gain a comprehensive understanding of the underlying unsteady flow field.

In this chapter, we present a new framework that designs an automatic guide for exploring internal flow features for unsteady flow fields. This solution encompasses feature identification, pathlet placement, region traversal order determination, viewpoint selection and tour generation into a single framework. In particular, we propose a new optimal solution for determining the critical region traversal order that integrates energy minimization and dynamic programming tehniques. Specifically, we first define a traversal score (Section 7.4.2) for each critical region at each time step then form the optimization problem to be maximizing the overall traversal score along the final tour path. This step is critically important as it pretty much determines the outline of the tour path, leaving path details to be solved in the subsequent steps. A user study is performed to show the effectiveness of our solution and confirm the benefits of including internal viewpoints in the tour design. This work will be submitted to *IEEE Transactions on Visualization and Computer Graphics 2015* soon.

7.2 Algorithm Overview

Our algorithm consists of three stages: critical region identification (Section 7.3), region traversal order determination (Section 7.4), and viewpoint creation and tour generation (Section 7.5). At the first stage, we detect critical regions at each time step and construct their temporal correspondence. We also extract the skeleton from each region for skeleton-based seeding. At the second stage, we design a solution that integrates energy minimization and dynamic programming to obtain the optimal traversal order for critical regions. This stage is the most important one in our algorithm as the region traversal order essentially outlines the "tour" in the high level. Once the order of traversal is determined, we create sample viewpoints along each focal region at the third stage. We then select best viewpoints based on their quality to generate the actual tour path.

7.3 Critical Region Identification

7.3.1 Critical Region Detection

Given the input unsteady flow field, we adopt a 4D moving window ($5 \times 5 \times 5 \times 5$) centered at each voxel and calculate its entropy value by evaluating the variation of vector directions in each local window. After calculating the entropy values for all voxels in the flow field, an entropy field is produced. Since computing the entropy value of one voxel is independent of another, we implement entropy computation using CUDA in the GPU. We extracted critical regions at each time step based on the entropy field. For each time step, the same extraction procedure used in Section 6.3 is applied. To speed up the computation, we implement critical region detection using CUDA in the GPU. Specifically, each GPU thread takes the responsibility for one voxel in the volume. Each thread checks if the entropy value for the corresponding voxel is larger than a given isovalue treshold δ_e (Section 6.4). If yes, the voxel is marked as a critical voxel. At the same time, the boundary critical voxel is also marked if at least one of its neighboring voxels is not critical. Each critical voxel is then assigned a unique ID. Next, for a critical voxel v with ID v_{ID}, we check each of its neighboring critical voxel's ID n_{ID} and set $n_{\text{ID}} = v_{\text{ID}}$ if $n_{\text{ID}} > v_{\text{ID}}$. This is applied to all critical voxels in parallel. After one round, some voxels' IDs are changed to

(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 7.1: (a) to (c): the critical regions detected for three consecutive time steps of the supernova data set. (d) to (f): the corresponding volume thinning results of (a) to (c), respectively. (g) to (i): extracted skeletons of (a) to (c).

smaller values. We iteratively apply this operation for multiple rounds until there is no ID change for any voxel. At this moment, all voxels with the same ID form a critical region. Figure 7.1 (a) to (c) show the critical regions detected for three consecutive time steps. Different colors indicate different regions.

7.3.2 Temporal Correspondence Computation

For an unsteady flow field, it is common for a critical region at a time step to span several time steps and overlap with other regions at neighboring time steps. To identify such temporal correspondences, we detect all matching regions between consecutive time steps by computing their overlap rates [75]. Two regions are matched when their overlap rate is larger than a given region volume overlap rate threshold δ_o. In this way, we construct an *overlap table* for every pair of neighboring time steps t and $t+1$ indicating the matching relations among critical regions. There are five cases:

† *Continuation.* If region r at t has overlap with only one region r' at $t+1$, then r' is a continuation of r.

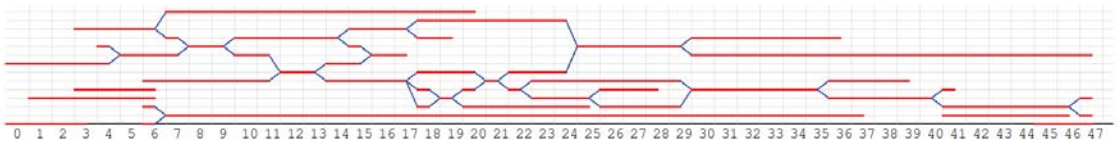


Figure 7.2: The temporal correspondence of critical regions extracted from the hurricane data set.

† *Bifurcation.* If region r at t has overlap with more than one region at $t+1$, then r is bifurcated into those regions at $t+1$.

† *Amalgamation.* If region r' at $t+1$ has overlap with more than one region at t, then those regions at t are amalgamated into r'.

† *Dissipation.* If region r at t overlaps with no region at $t+1$, then r is dissipated at $t+1$.

† *Creation.* If region r' at $t+1$ overlaps with no region at t, then r' is created at $t+1$.

Based on those overlap tables, we further apply the feature tracking algorithm developed by Silver and Wang [74] to build the temporal correspondence among critical regions. Figure 7.2 shows an example of the temporal correspondence of critical regions. The horizontal axis represents time step. Each red line represents the life span of one critical region. From Figure 7.1 (a) to (c), we observe three cases: continuation (e.g., the light blue region from (a) to (b)), bifurcation (e.g., the purple region in (a) to the yellow, red and blue regions in (b)), and amalgamation (e.g., the yellow and light blue regions in (b) to the green region in (c)).

7.3.3 Region Skeleton Extraction

In order to identify the shape of each critical region detected, we construct its skeleton by applying the same algorithm described in Section 6.3. In practice, for a critical region which continues for several time steps, we extract its 3D skeleton at each time step separately.

7.3.4 Skeleton-based Seeding

In order to trace pathlines, we apply a skeleton-based seeding strategy by evenly placing seeds along the skeletons extracted from critical regions. Our goal is to make sure that each focal region at every time step has enough pathlets to represent it for clear highlighting, and in the meanwhile, the entire flow field has approximately the same numbers of pathlets for each time step. To achieve this, we place two types of seed: *region seeds* and *random seeds*. First, we place region seeds along the skeleton of each focal region detected at every time step and ensure that the number of seeds for each region is proportional to its size. To ensure that the traced pathlets will capture each region of focus, we start to trace region seeds a few time steps earlier than the time step when the region starts to be focused on. For random seeds, we keep track of the number of pathlets in each time step. The number of pathlets could

be less as the time evolves since pathlets may go out of the domain boundary or get absorbed around the vicinity of a point (like a sink in a steady flow field). If the number is less than a given threshold, we add some more seeds randomly to ensure the same number of pathlets for each time step.

7.4 Region Traversal Order Determination

In general, we may have multiple critical regions at a single time step and a critical region may continue for multiple time steps. Our goal is to produce a "smooth" traversal of "good" critical regions along the time sequence, conveying the most information about the underlying unsteady flow field. Assuming that we have a total of n regions and m time steps and we select a single critical region to focus on for each time step, the search space for determining the region traversal order is bounded by $O(n^m)$. The worst case happens when each region occupies all the time steps. Obviously, the brute-force method has an exponential time complexity and is not practical. A straightforward greedy algorithm which always picks the "best" region for each time step could be applied. However, the greedy method does not guarantee a globally optimal result. Moreover, temporal correspondence among regions is not considered. Therefore, we present a novel method which integrates *energy minimization* and *dynamic programming* to obtain the optimal traversal order for critical regions.

7.4.1 Overview of Method

We define $I_{r,t}$ as the traversal score of region r at time step t. The range of $I_{r,t}$ is $[0, 1]$. Since the computation of $I_{r,t}$ requires multiple criteria, we define several constraints accordingly and use a linear system to find the optimal solution for this value (Section 7.4.2). Intuitively, a larger (smaller) value of $I_{r,t}$ indicates that r has a higher (lower) chance of being selected as the focus at t. We then divide the entire time sequence into multiple time windows and compute a local traversal order for each time window separately. Specifically, for a time window w, we minimize an energy function to compute the optimal scores for critical regions in w and then apply a dynamic programming algorithm to determine the region traversal order within w. To incorporate region temporal correspondence into score computation, the resulting traversal order for w will be feed as the input for score computation in the next time window $w + 1$. After all the time windows have been processed, we apply the same dynamic programming algorithm over all the region scores to identify the globally optimal traversal order for the entire time sequence.

There is a significant difference between traditional minimization methods and our method. Traditionally, energy functions are formulated with a set of initial state values. Then, a linear system is utilized to solve for the optimal solution. In our method, the traversal result from the current time window highly depends on the

result from the previous time windows. That is, selecting the current focal region should consider the regions picked before in order to provide a smooth and efficient traversal experience. As such, rather than solving the linear system in a single pass as usual, we adopt a hybrid approach which applies energy minimization and dynamic programming iteratively across different time windows to obtain the final optimization result.

7.4.2 Optimal Region Score Computation

To formulate the energy function for optimal region score computation, we define the following two types of constraints: *static constraints* and *dynamic constraints*. The static constraints consider the intrinsic properties of critical regions while the dynamic constraints incorporate the influence of the preceding region traversal order on the current region traversal order being determined.

7.4.2.1 Intrinsic Properties

Before we introduce the static constraints, we first define the following intrinsic properties of critical regions:

† *Size* ($S_{r,t}$). This term indicates the volume size of critical region r at time step

t. The larger the size, the more important the region. Therefore, region score $I_{r,t}$ should be proportional to $S_{r,t}$.

† *Average entropy* ($E_{r,t}$). This term computes the average entropy value over all the grid points inside of region r at time step t. Intuitively, the value of $E_{r,t}$ indicates the amount of information contained in r at t. Therefore, $I_{r,t}$ should be proportional to $E_{r,t}$.

† *Coefficient of variation* ($V_{r,t}$). We use this term to measure the normalized dispersion of the entropy value distribution inside of region r at time step t. $V_{r,t}$ is computed as follows

$$V_{r,t} = \frac{\delta_E(r,t)}{\mu_E(r,t)}, \tag{7.1}$$

where $\delta_E(r,t)$ and $\mu_E(r,t)$ are the standard deviation and mean of entropy values inside of r at t, respectively. We prefer to focus on the region with higher $V_{r,t}$ since such a region contains richer information about the underlying flow features. Thus, $I_{r,t}$ should be proportional to $V_{r,t}$.

† *Skeleton complexity* ($C_{r,t}$). Since a region skeleton represents the overall shape of the corresponding region, its complexity should also be considered when computing the region's score. A high value of $C_{r,t}$ indicates that the corresponding region has a complicated shape pattern and would be interesting to focus on. Region score $I_{r,t}$ should be proportional to $C_{r,t}$. We consider two factors for

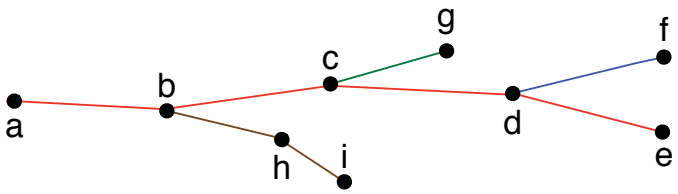


Figure 7.3: Four branches shown in different colors for a region's skeleton.

$C_{r,t}$: one is *shape complexity* and the other is *quantity complexity*. For shape complexity $C_{s,t}$, we define it as follows

$$C_{s,t} = \sum_{\mathbf{b} \in \mathbf{B}(r,t)} \frac{\sum_{i,j \in \mathbf{b}} d_{i,j} - d_{\mathbf{b}\,\max}}{d_{\mathbf{b}\,\max}}, \tag{7.2}$$

where $\mathbf{B}(r,t)$ is the set of branches in the skeleton of r at t, $\mathbf{b}$ is a branch in $\mathbf{B}(r,t)$, $d_{i,j}$ is the distance between two *consecutive* skeleton points i and j, and $d_{\mathbf{b}\,\max}$ is the largest distance among *all* points in $\mathbf{b}$. Since our skeleton is a tree structure, we can compute all the branches of a skeleton by applying the *depth-first-search* (DFS) algorithm. As we visit skeleton points along the tree until we reach a leaf, the traversal path formed by those points visited defines a branch. Then we pop out visited skeleton points one by one and in the meantime, continue to traverse unvisited points, if any, to identify other branches. In Figure 7.3, *abcde*, *df*, *cg* and *bhi* are branches of this skeleton. In addition, if we apply Equation 7.2 to branch *cg*, the shape complexity is zero because it is a straight branch. However, the shape complexity of *abcde* is non-zero due to its non-straight pattern. Quantity complexity $C_{q,t}$ records the number of branches

in a skeleton. For the example given in Figure 7.3, $C_{q,t} = 4$. Intuitively, more branches may potentially lead to a higher complexity. For both $C_{s,t}$ and $C_{q,t}$, we normalize them by dividing their corresponding maximum value at time step t, then $C_{r,t}$ is defined as follows

$$C_{r,t} = \alpha\overline{C_{s,t}} + (1-\alpha)\overline{C_{q,t}}, \tag{7.3}$$

where $\alpha \in [0, 1]$, and $\overline{C_{s,t}}$ and $\overline{C_{q,t}}$ are the normalized values. Normally, we consider shape complexity to be more important than quantity complexity. Therefore, we set $\alpha > 0.5$.

† *Time span* (T_r). This term indicates the time span that region r is alive. If T_r is small, r will be alive for only a few time steps. In order to capture this short-lived region, we should assign a higher region score $I_{r,t}$ to r so that it gets a chance to be focused on during its time span. Therefore, $I_{r,t}$ should be inversely proportional to T_r. However, if r with short time span has parents or children and these parents or children regions are long-lived, then even if we miss r, we can still capture similar flow features by focusing on its parents or children. Therefore, T_r should be only applied to regions with no parents or children.

7.4.2.2 Static Constraints

With the intrinsic properties defined above, we obtain the first static constraint:

Intrinsic property constraint (O_τ)**.** We introduce this constraint to enforce that region scores should be as close to the maximum value of 1 as possible according to their intrinsic properties. We first combine all intrinsic properties for a region into a single term $P_{r,t}$

$$P_{r,t} = \begin{cases} \lambda_S S_{r,t} + \lambda_E E_{r,t} + \lambda_V V_{r,t} + \lambda_C C_{r,t} + \lambda_T \frac{1}{T_r}, & r \text{ is alive at } t \\ 0, & \text{otherwise} \end{cases} \tag{7.4}$$

where λ_S, λ_E, λ_V, λ_C and λ_T are the weights for $S_{r,t}$, $E_{r,t}$, $V_{r,t}$, $C_{r,t}$ and T_r, respectively. In this paper, we set the first four weights to 1.0 and λ_T to 2.0 for all data sets. With $P_{r,t}$, we define the following energy term

$$O_\tau = \Sigma_{r \in \mathbf{R}, t \in \mathbf{W}} P_{r,t} \| I_{r,t} - 1.0 \|^2, \tag{7.5}$$

where $\mathbf{R}$ is the set of all critical regions and $\mathbf{W}$ is the set of time steps in time window w. Note that if region r is not alive at time step t, its initial score $I_{r,t}$ will be set to 0.

Summation constraint (O_σ)**.** This constraint is used to keep the summation of all the scores $I_{r,t}$ in the range of $[0, 1]$ as much as possible. To achieve this, we set the summation of scores for the regions at the same time step to 1. If the score is less than 0, we will set it to 0 to avoid negative scores. We define this energy term as follows

$$O_\sigma = \Sigma_{t \in \mathbf{W}} \| \Sigma_{r \in \mathbf{R}} I_{r,t} - 1.0 \|^2, \tag{7.6}$$

Again, if r is not alive at t, its initial score will be set to 0.

7.4.2.3 Normalized Traversal Frequency

The dynamic constraints rely on the preceding region traversal order. Therefore, we introduce how to compute $L_f(r, w)$ which records the frequency that region r has been traversed before the current time window w. Since for each time window, we will apply a dynamic programming algorithm to find the order of traversal after we compute the optimal scores $I_{r,t}$, we can easily obtain $L_f(r, w)$ by counting the frequency that r has been visited in the previous time windows. We also define L_b as the maximum number of time windows we will backtrack from w. This term is used to control the temporal history length we would consider for r. Ideally, a good L_b

should keep the tracking in a reasonable time span (e.g., not too long or too short). Sometimes, r may last less than L_b during the backtracking. In this case, we will consider the traversal history of r's ancestors when computing $L_f(r, w)$. Using the above two terms, we define the normalized traversal frequency $F(r, w)$ of region r for time window w as

$$F(r, w) = \frac{L_f(r, w)}{L_b}. \tag{7.7}$$

7.4.2.4 Dynamic Constraints

With $F(r, w)$, we now define several dynamic constraints:

Traversal frequency constraint (O_ν)**.** If region r has been traversed for multiple time steps in the previous time windows, its score in the current time window w should be decreased so that other regions could have a chance to be selected as the focus. To achieve this, we attempt to minimize the following energy term

$$O_\nu = \Sigma_{r \in \mathbf{R}, t \in \mathbf{W}} \frac{1}{F(r, w)} \|I_{r,t} - 1.0\|^2. \tag{7.8}$$

Temporal correspondence constraint (O_η)**.** This constraint considers the influence of previously focused regions on their temporal corresponding regions. When region r has been visited for multiple time steps, its children which share similar spatial locations with r in neighboring time steps should have lower scores. Contrarily, in order to maintain a smooth traversal order, the siblings of r should get higher scores since they are less likely to be similar to r but have temporal relations with r. For example, they will merge in the later time step or they come from the same parent. Considering these two cases, we define the following energy term

$$\begin{aligned} O_\eta = & \Sigma_{r\in\mathbf{R},t\in\mathbf{W}}\Sigma_{c\in\mathbf{C}(r)}\frac{1}{F(r,w)}\|I_{c,t}-1.0\|^2+ \\ & \Sigma_{r\in\mathbf{R},t\in\mathbf{W}}\Sigma_{s\in\mathbf{S}(r)}F(r,w)\|I_{s,t}-1.0\|^2, \end{aligned} \tag{7.9}$$

where $\mathbf{C}(r)$ and $\mathbf{S}(r)$ are the sets of r's children and siblings, respectively.

Spatial constraint (O_ξ)**.** When region r has been traversed for multiple time steps, its own score will decrease in the subsequent time steps. Meanwhile, we want other regions which have no temporal correspondence with r to have more chance of being selected as the focus. Therefore, we introduce this spatial constraint to increase the scores of such regions based on their spatial distances to r. We formulate the following

energy term

$$O_\xi = \Sigma_{r\in\mathbf{R},t\in\mathbf{W}}\Sigma_{k\in\mathbf{K}(r)}\frac{F(r,w)}{D_{r,k}}\|I_{k,t}-1.0\|^2, \tag{7.10}$$

where $\mathbf{K}(r)$ is the set of regions which have no temporal correspondence with r, $D_{r,k}$ is the distance between the skeletons of r and k at t. In this work, we use the mean of the closest point distances [64] as the distance measure. Intuitively, region k will have a higher score if $D_{r,k}$ is smaller.

7.4.2.5 Energy Function

Based on the above constraints, we formulate the final energy function as follows

$$O = \gamma_\tau O_\tau + \gamma_\sigma O_\sigma + \gamma_\nu O_\nu + \gamma_\eta O_\eta + \gamma_\xi O_\xi, \tag{7.11}$$

where γ_τ, γ_σ, γ_ν, γ_η and γ_ξ are the weights for these constraints, respectively. In this paper, we set γ_η to 2.0 and the rest of weights to 1.0 for all data sets. To find the optimal solution, we convert the energy function into a linear system and leverage a GPU implementation of the concurrent number cruncher (CNC) sparse solver [10] to

solve the system.

7.4.3 Traversal Order Determination

After region scores are computed for a time window w, we utilize a dynamic programming algorithm to find the optimal traversal order $\Omega(w)$ within the time window. $\Omega(w)$ should focus on one region at a time step and the overall region scores along $\Omega(w)$ should be maximized. To achieve this, we define $\breve{I}_{r,t}$ as the maximum traversal score from region r at t to some region at the last time step of w and introduce the following recursive equation

$$\breve{I}_{r,t} = \max_{k \in \mathbf{R}} (I_{r,t} + \breve{I}_{k,t+1}), \tag{7.12}$$

where $\mathbf{R}$ is the set of all regions. This equation indicates that the maximum traversal score from region r at t to some region at the last time step of w is equal to the sum of the score $I_{r,t}$ and the maximum traversal score from region k at $t+1$ to some region at the last time step of w. In this case, region k will be the focal region at $t+1$ and be put in the array $A_{r,t}$, which records the traversal order starting from r at t. For each region, we compute its $\breve{I}_{r,t_0}$ where t_0 is the first time step of w and pick A_{r,t_0} with the largest $\breve{I}_{r,t_0}$ as the optimal traversal order $\Omega(w)$. We apply the same

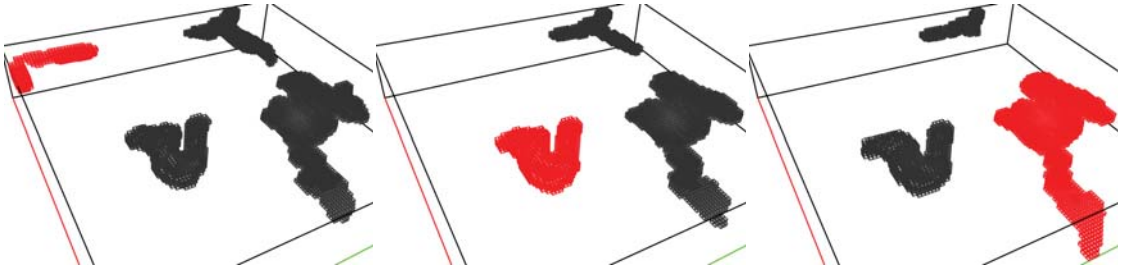
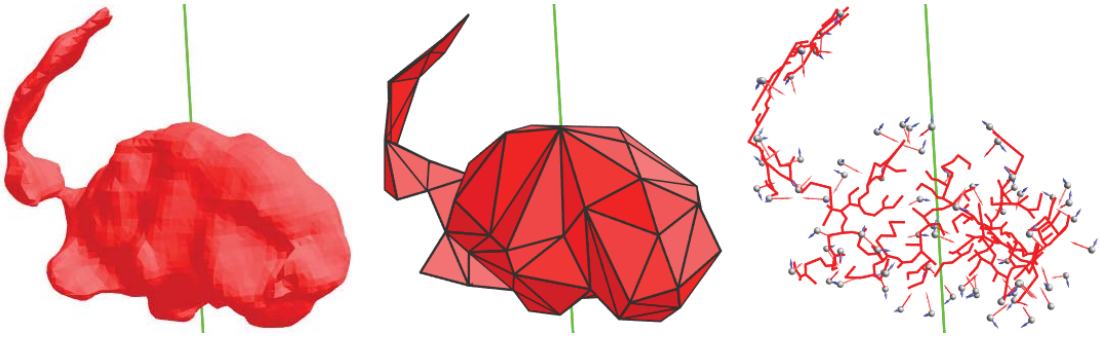


Figure 7.4: The shifting of the focal region (shown in red) for three consecutive time steps of the hurricane data set.

algorithm to find the final traversal order for the entire time sequence when all region scores $I_{r,t}$ are obtained. In Figure 7.4, we show some snapshots of the region traversal order determined using our optimization method where the red region denotes the focal region at each time step.

7.5 Viewpoint Creation and Tour Generation

For the focal region at each time step, we first create a list of viewpoints based on the isosurface generated from the region. Then we select several best viewpoints to provide the user with closeup views of the respective flow pattern for clear observation. If one critical region spans more than one time step, we also consider the location of viewpoints so that the same portion of the region will not be repetitively focused on across multiple time steps. After best viewpoints are picked for all focal regions, a view path traversing all these viewpoints according to the time order is generated. Our goal is to give a smooth and efficient way of exploring the unsteady flow field.



(a) (b) (c)

Figure 7.5: (a) the isosurface constructed from a critical region of the supernova data set. (b) the simplified mesh constructed from the isosurface. (c) all viewpoints generated on the simplified mesh surface.

7.5.1 Isosurface Construction

We leverage the same strategy describe in FlowTour which utilizes isosurfaces to represent critical regions and creates the list of viewpoints. Specifically, for the focal region r_t at time step t, we use the marching cube algorithm [53] to extract the corresponding isosurface $\mathbf{s}_t$ based on the entropy threshold δ_e (Section 7.3). Figure 7.5 (a) shows an example of the constructed isosurface. To obtain the vertex connectivity information of $\mathbf{s}_t$, we convert the isosurface into a triangle mesh $\mathbf{m}_t$. Since our viewpoints are obtained from the vertices on $\mathbf{m}_t$, a mesh decimation algorithm [36] is applied to reduce the number of vertices to a manageable level and a new simplified mesh $\mathbf{m'}_t$ is obtained. A decimation factor δ_s is provided for the user to control the level of simplification. This process is applied to the focal region at each time step. Figure 7.5 (b) shows an example of the simplified mesh.

7.5.2 Viewpoint Creation

Given the simplified mesh $\mathbf{m'}_t$, our algorithm creates a list of viewpoints based on each vertex on $\mathbf{m'}_t$. For each vertex, we first generate a viewpoint v at the vertex's location. The `look-at` center of v is the point on the skeleton of region r_t closest to v. There are two ways to compute the `up` direction. A simple way is to fix the `up` direction to a predefined direction (such as the positive y direction of the volume) while another way is to utilize the skeleton's major direction as the guidance. Specifically, we define the local skeleton direction $\mathbf{d}$ at the `look-at` center as the vector along the skeleton which starts from the `look-at` center and points toward the skeleton's major direction. We then project $\mathbf{d}$ onto a plane perpendicular to the `look-at` direction $\mathbf{l}$ and the final `up` direction is the projected vector on the plane. However, if the flow pattern changes frequently, this method will cause the `up` direction to vary dramatically. In this case, using a fixed `up` direction provides a more stable exploration experience. In Figure 7.5 (c), we show all the viewpoints generated at the vertices on the simplified mesh surface. The corresponding `look-at` directions and `up` directions are also displayed.

In order to consider the zoom level from the viewpoint v to critical region r_t, we generate a set of *offset viewpoints* $\mathbf{V}$ associated with v. The position of each offset viewpoint is pushed away from v along the opposite direction of its `look-at` direction $\mathbf{l}$ for some distance and it shares the same `look-at` center and the `up` direction of v.

In the next step, we will evaluate the quality of each viewpoint in $\mathbf{V}$ and pick one best viewpoint as the representative for $\mathbf{V}$.

7.5.3 Viewpoint Quality Evaluation

Given a viewpoint v associated with critical region r_t, we define its quality based on the entropy value of the 2D projection of the pathlets seeded from r_t. We also consider the foreground occlusion and background noise as penalty to reduce visual clutter and distraction. Specifically, we compute the *viewpoint quality* as follows

$$Q(v) = \lambda_1 S_{\text{focus}} - (\lambda_2 P_{\text{fore}} + \lambda_3 P_{\text{back}}), \tag{7.13}$$

where S_{focus}, P_{fore} and P_{back} are the *focal region score*, *foreground occlusion penalty*, and *background noise penalty*, respectively. λ_1, λ_2, and λ_3 are the corresponding coefficients. By setting different values to these coefficients, we ensure that the viewpoint quality $Q(v)$ is always non-negative. In this paper, we set λ_1 to 1.0 for all data sets and the rest of coefficients to a value less than 1.0.

† *Focal region score* (S_{focus}). This term indicates the information revealed by the pathlets seeded from the focal region's skeleton. To compute this value, we

first perform the viewing frustum culling operation on each pathlet from the focal region and check if it is inside of the viewing frustum. For all the pathlets inside, we compute the entropy value for their 2D projections based on the flow direction and set this value to S_{focus}.

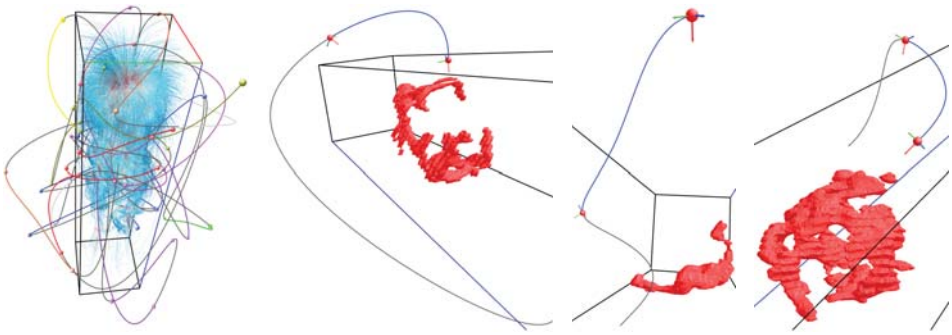
† *Foreground occlusion/background noise penalty* ($P_{\text{fore}}, P_{\text{back}}$). These two terms measure the influence of pathlets seeded from non-focal regions on the current viewpoint. To compute these two values, we first check if the pathlets are inside of the viewing frustum. If true, we then transform the standard OpenGL view projection plane into a predefined $n \times n$ plane and check the depth values of these pathlet projections on the plane to determine whether they are foreground occlusion or background noise pathlets. The value of n is user specified. A larger n will generate a higher resolution projection plane and produce more accurate results, but it is more time consuming than a lower value of n. P_{fore} and P_{back} are obtained as the entropy values of their 2D projections for these two kinds of pathlet, respectively.

For a critical region which spans m time steps ($m > 1$), we assign viewpoints to focus on different portions of the region at each occupied time step so that the user could observe the flow patterns in a more balanced way. To achieve this, we divide the major axis of the focal region's skeleton into m segments where each segment corresponds to one occupied time step. Then we compute the final viewpoint score according to their distance to the segments. Specifically, for a viewpoint v, we find

one segment g which is closest to v and set g as the segment associated with v. Then at each of the m occupied time steps, there will be one *active segment* and all the viewpoints associated with this segment will obtain their final score as $Q(v)$ while other viewpoints get the score of zero. As in the next step, we will pick best viewpoints for each time step based on their score and we can guarantee that only viewpoints within the current active segment would be considered.

7.5.4 Best Viewpoint Selection

By evaluating the viewpoint quality, we first identify the representative viewpoint for each viewpoint set V as the one with the highest score in the set. Then we sort all representatives based on their quality scores and pick the final best viewpoints with the highest scores. In order to avoid similar viewpoints to be selected, any two selected best viewpoints should satisfy either one of the following two criteria: First, the angle between their `look-at` directions is greater than a given threshold δ_α. Second, the distance between their `look-at` centers is greater than a given threshold δ_d.



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

Figure 7.6: (a) the entire tour path along the solar plume data set. (b) to (d): three path segments along three different focal regions.

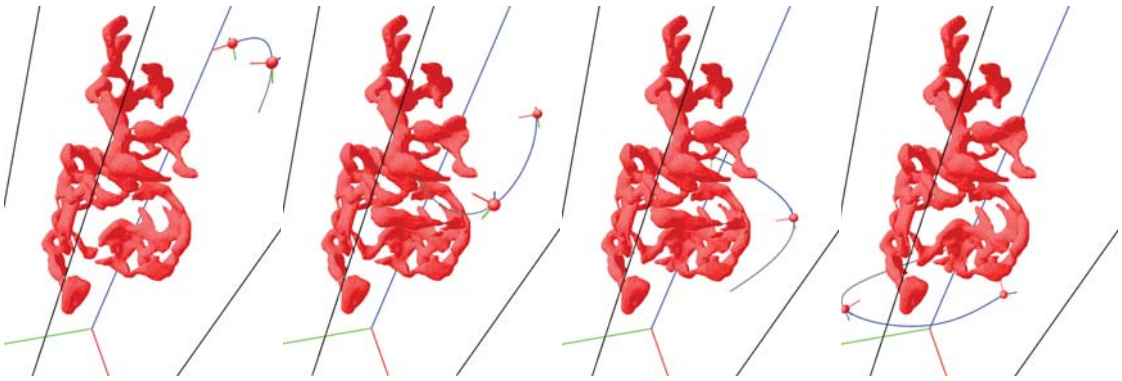


Figure 7.7: Left to right: the tour path segments in order along the same focal region of the solar plume data set.

7.5.5 Tour Path Generation and Animation

After we identify best viewpoints for the focal region at each time step, we need to generate a spatiotemporal view path traversing all these viewpoints in a smooth and efficient manner. To achieve this, we first order the best viewpoints for a focal region r according to the direction of the skeleton's major axis to obtain a local traversal order D_r. Since the temporal order for all focal regions is already determined by the linear system (Section 7.4), we only need to connect the viewpoints between

two temporally adjacent regions to obtain the final traversal order for all the best viewpoints. Specifically, for the focal region r_1 at the first time step, we use its local traversal order D_{r_1} as the final traversal order for this region. Then for the next focal region r_2 along the temporal sequence, we compute the distance between the last viewpoint v_{last} of r_1 and the two end viewpoints of the local order of r_2 and pick the one closer to v_{last} as the starting viewpoint for r_2. So the final order for r_2 will be either D_{r_2} or its reverse order. We repeat this process until we connect all the focal regions and get the global viewpoint traversal order. Finally, we utilize a cubic B-spline curve to connect all these viewpoints to obtain the spatiotemporal view path. In Figures 7.6 and 7.7, we show examples of the entire path generated, the path segments along different focal regions, and the path segments along the same focal region.

For different unsteady flow fields, since the numbers of time steps vary greatly (from tens to hundreds) and the speeds of flow also vary significantly, we decide to keep the current viewpoint stay put for a certain duration when flying through each best viewpoint in the tour animation. This would allow the user to better observe pathlet movements at fixed viewpoints. The transition between two best viewpoints is dynamic, i.e., the viewpoint is changing while pathlets are moving. As the background information, we render pathlines that are alive at the current time step as thin tubes. A larger tube radius is used for pathlines traced from the focal region to differentiate pathlines traced from non-focal regions. We overlay pathlets as arrows to show their

Table 7.1
The parameter settings for the three flow data sets.

data set	dimension	#best viewpoints/ #total viewpoints	entropy δ_e	overlap δ_o	win size L_b	decimator δ_s	angle δ_α	distance δ_d
supernova	$108 \times 108 \times 108 \times 105$	105/4575	4.0	0.1	5	0.330	$\pi/6$	20.0
hurricane	$100 \times 100 \times 20 \times 48$	48/2676	3.5	0.1	4	0.330	$\pi/6$	15.0
solar plume	$63 \times 63 \times 256 \times 28$	56/2615	4.0	0.1	4	0.330	$\pi/6$	20.0

Table 7.2
The timing results for the three unsteady flow data sets (A * denotes out-of-core processing).

data set	entropy comp.	region detection	region corresp.	skeleton extraction	traversal order	viewpoint creation	viewpoint evaluation	tour path generation
supernova	225.7s*	31.8s	124.8s	109.7s	2.6s	0.4s	500.1s	0.1s
hurricane	21.7s	4.2s	3.7s	3.4s	0.7s	0.1s	273.8s	0.02s
solar plume	63.7s	9.6s	37.9s	387.6s	3.7s	0.5s	248.7s	0.1s

movements along the corresponding pathlines. Again, tube radius is larger for focal pathlets than non-focal pathlets. When the camera focus changes from one focal region to another, an animated transition is shown, indicating the changes of pathline and pathlet thickness.

7.6 Results and Discussion

7.6.1 Timing and Parameters

We performed our experiments with the same hardware configuration used in Chapter 4. Entropy field computation, critical region detection, and viewpoint quality evaluation were implemented in the GPU using CUDA. All other computations were performed in the CPU. Since we use pathlet animation to show the movements of the

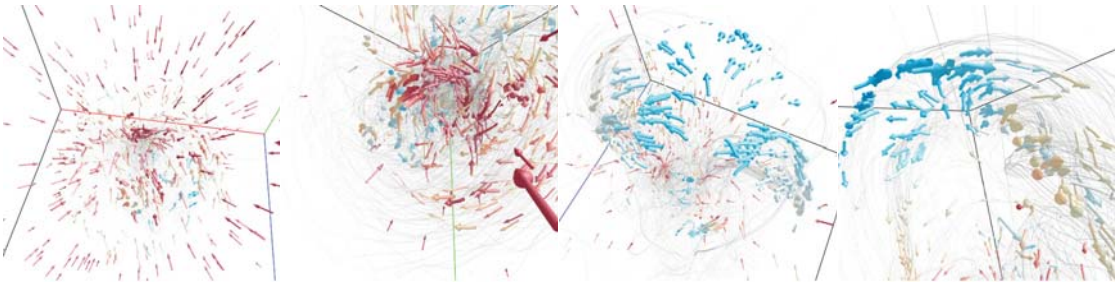
underlying flow patterns, pathlet positions need to be updated in each frame. Furthermore, for better observation of the changes of the focal region over time, we also increase (decrease) the radii of the pathlets and pathlines from the current (previous) focal region. In order to guarantee smooth update of the changes of pathlets and pathlines, we utilized the vertex buffer object (VBO) to render pathlets and pathlines and used the GPU to process their geometry changes on the fly. The timing results and parameter settings used for the three flow data sets are reported in Tables 7.2 and 7.1, respectively.

7.6.2 Case Studies

We presented three case studies on three different unsteady flow data sets to demonstrate the effectiveness of our framework in helping the user explore internal flow features and patterns.

† *Case Study 1 — Supernova*

The simulation produced 105 time steps revealing how the dusts collapsed back into the center of the star after the supernova explosion. Since the flow near the supernova's core is heavily turbulent, it is difficult for the user to observe detail patterns around the core due to the occlusion and clutter among pathlets. Therefore, we designed an automatic view path to explore the data set in the hope that our method can provide the user with more information on the



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

Figure 7.8: (a) to (d): four snapshots of the visualization of the supernova data set along our automatic tour path.

internal flow patterns. In Figure 7.8, we show four snapshots along with our view path. From (a), we see the overall pattern of the supernova at an early time step. It is clear that all pathlets go straightly inward and then become turbulent near the supernova's core. The thicker pathlets indicate that they are from the current focal region. We also display the corresponding pathlines to help the user better follow the flow trajectories. In (b), a sink-like point which absorbs all pathlets is clearly observed. From (c), we observe that some pathlets are repelled from the supernova's core and then form a semi-spherical surface around the core. This feature is difficult to catch since it is not always present throughout the time series. Furthermore, it is also hidden inside of the turbulent flow, and therefore not visible if the viewpoints are placed outside of the volume. Using our method, we not only clearly observe such an interesting flow pattern in a closed-up view but also detect the changes of velocity. In (d), it is clear that the velocity changes from high (blue) to low (yellow) as the flow is repelled from the core.

† *Case Study 2 — Hurricane*

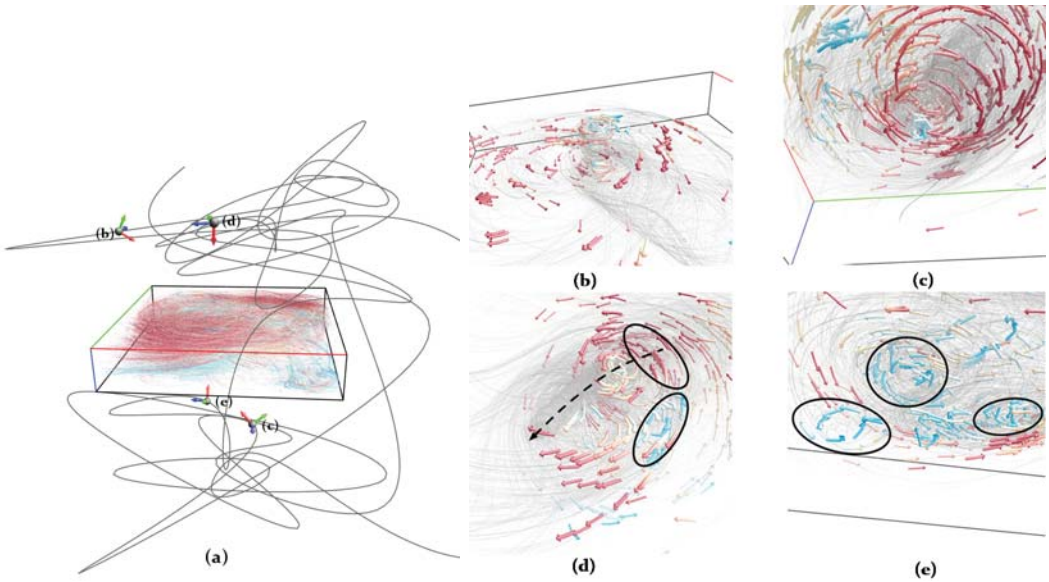
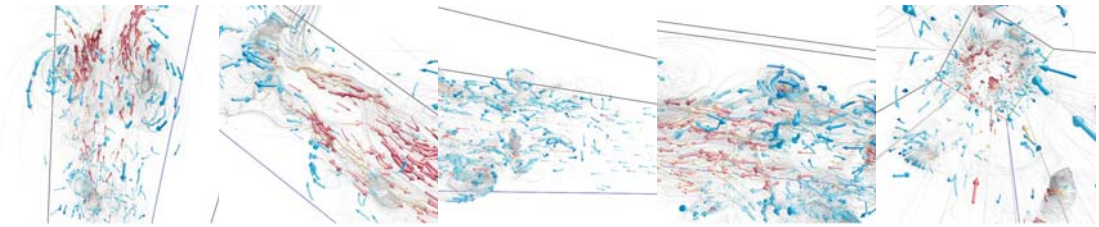


Figure 7.9: (a) the automatic tour path for the hurricane data set. (b) to (e): four snapshots of the visualization along the path.

The simulation produced 48 time steps which demonstrate how the hurricane moved from the Atlantic Ocean to the east coast of Florida. Since the data set is quite flat ($100 \times 100 \times 20$), the user would easily get disoriented if the tour goes across the volume frequently along the z direction. Therefore, we add one more constraint that the final view path should not cross through the volume more than once in the z direction during any given animation time interval. In Figure 7.9 (a), we show the entire tour path. We find that the most portion of the path is on the two sides of the volume and the path only traverses across the volume a few times. In (b), the global pattern of the hurricane is clearly shown. We see that the hurricane's center lies in one corner of the volume with the corresponding pathlets moving out. Figure 7.9 (c) shows the hurricane's center from below. We observe the spiral pattern and the velocity difference



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

Figure 7.10: (a) to (e): five snapshots of the visualization of the solar plume data set along our automatic tour path.

between the center flow (slower) and the surrounding flow (faster). From (d), we find out that the surrounding flow around the hurricane's center is bifurcated into two opposite directions (highlighted in the ellipses). The velocity change could also be discerned via pathline color. Besides, the pathlines depict the moving trajectory of the hurricane's center: from one corner of the volume to another along the diagonal direction (highlighted in the dashed line). In (e), the snapshot shows three small spirals with slower speed (highlighted in the ellipses) at the boundary of the volume which only last for a few time steps. These patterns could be easily missed if the view path does not focus on them at the right time.

† *Case Study 3 — Solar Plume*

This data set consists of 28 time steps which demonstrate the heat flow emitting from the surface of the sun. Figure 7.10 shows five snapshots of the visualization along our automatic tour path. Refer to Figure 7.6 (a) for the overall view path generated by our algorithm. Since the tail of the solar plume only contains straight flow lines and most interesting patterns are around the head of the

solar plume, our view path is almost around the head portion so that the user can gain a clear observation of these important flow features. In (a), an overview of the flow pattern around the head region is captured. Two big swirls with slower speed along with the central flow with higher speed are clearly shown. (b) focuses on the central flow patterns around the head. (c) gives us a glance on how the straight-line flow becomes turbulent in the middle portion of the plume. From (d), we observe a spiral band pattern with slower speed in the middle portion of the plume. Our view tour also provides the user with an expressive traversal experience to observe the internal "kernel" flow patterns by "standing" inside of the volume, which is shown in (e). The straight flow lines moving to the head and the hollow shaft pattern are clearly visible.

7.7 User Study

We conducted a user study to evaluate the effectiveness of our method. We used a design of 2 conditions (our view tour and random view tour) $\times 2$ tasks (answer questions and identify critical regions). To be fair, the random view tour has the same number of viewpoints, and almost the same path length and total view angle change as our view tour. We recruited seven users for each condition. All users are graduate students from different departments (computer science, mechanical engineering, physics) of the same university. For each experimental session, a user was shown

three flow field data sets.

7.7.1 Random View Tour

For the random view tour, we first generate a set of viewpoints whose positions are randomly picked both inside and outside of the volume. For external viewpoints, we keep them not too far away from the volume's center to ensure clear observation of the flow field. The `look-at` center of each viewpoint is also randomly created. However, we constrain their positions to be inside of the volume so that the viewpoints could still focus on the flow field rather than any empty space outside. Next, we connect all these viewpoints in a way such that both the Euclidean distance between viewpoints and the angle change along the path could be minimized. Finally, we interpolate a B-spline curve passing all viewpoints to create the tour path. If the total length or the angle change of the random view tour is much different from our method, we will regenerate the path by replacing some viewpoints.

7.7.2 Experimental Procedure

First, the user was shown an animation of the complete tour of the data set without stop or pause. The speed of the animation could be adjusted if desired. Second, the user was asked to answer several multiple-choice questions about the flow field and

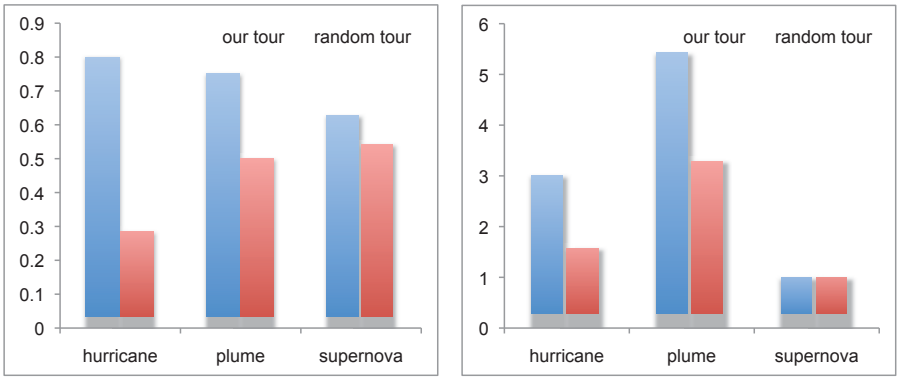
to identify critical regions. They had a limited time to perform these tasks. The user could revisit any part of the tour using a slider or replay the animation. This function was useful for answering questions and was required for identifying critical regions. Finally, the user was asked to answer several post-experimental questions about subjective feedback and suggestions for improvement regarding the experiment and the program's user interface.

User should complete all three data sets in one sitting. The entire experiment took approximately an hour for each user, including the initial paperwork, briefing, and post-experiment questionnaire.

7.7.3 Results and Discussion

We present the results of this study in the following aspects: user correctness on multiple-choice questions, ratings of subjective questions, the proportion of critical regions correctly identified, and the post-experiment feedbacks. We used Student's t-test to analyze statistical significance between the conditions with a standard significance level $\alpha = 0.05$. Though many people oppose to use the t-test on small samples (seven in our case), many articles [12, 20] suggested that there are no principle objections to use a t-test for an small sample size.

Multiple-choice questions. Each data set was analyzed individually by comparing



(a) (b)

Figure 7.11: (a) the average proportion of correct answers of multiple-choice questions for each data set. (b) the average number of critical regions detected for each data set.

users' average proportion of correct answers by the two methods. The average correctness rates of all users for the two methods are given in Figure 7.11 (a). A t-test shows that our method performed better than the random one on the hurricane and solar plume data sets and the difference is statistically significant with $p = 0.000002$ and $p = 0.045$, respectively. For the supernova data set, although our method also received higher average correctness rate than the random method (0.63 vs. 0.54), the difference is not statistically significant.

Subjective questions. There are two subjective questions for each data set asking the effectiveness of finding critical regions and identifying global flow patterns. We quantized the answers by setting 1.0 for "Strongly Agree", 0.75 for "Agree", 0.5 for "Neutral", 0.25 for "Disagree", and 0.0 for "Strongly Disagree". For the hurricane data set, our method gets much higher average ratings (0.93 vs. 0.64 and 0.79 vs. 0.61) than the random one for both questions. But only the first question has significant

difference with $p = 0.03944$. Our method also receives a higher ratings than the random one for the solar plume data set with average ratings 0.75 vs. 0.57 and 0.68 vs. 0.61 though no significant difference is shown. For the supernova data set, both methods received similar scores.

Critical regions identification. The average number of critical regions detected for both methods are given in Figure 7.11 (b). The supernova data set gets the same result (1.0) for both groups. This indicates that every user in both groups identifies one critical region. Actually, this is not surprising since this data set only contains this enormous critical region which is easily identifiable.

The average number of detected critical regions is 3.00 vs. 1.57 for the hurricane data set and 5.43 vs. 3.29 for the solar plume data set. Furthermore, the performance difference for plume is significant with $p = 0.04$ where the same conclusion cannot be made for the hurricane because its p value (0.05005) is only slightly above the significant level (0.05).

Post-experiment feedbacks. We received the following major user comments from the post-experiment questionnaire. First, for both our and random methods, most users suggested that the camera may stay longer at each of the selected viewpoints to allow better observation and less visual jumping. Second, the background pathlines should be thinner for reducing distraction. Third, some users also suggested to provide a global view of the data set before the experiment since the tour path may focus on

the internal detailed patterns rather than the global shape of the flow field. Users also had some different feedbacks for the two methods. For the random method, users gave the neutral rating for detecting flow features. One of them claimed that the `look-at` direction sometimes provided an unreasonable view of sight for observing the flow field. By contrast, most users agreed that the tour generated by our method could easily help them find critical regions. For other questions, such as animation speed and pathlet size, both groups were satisfied with the current configurations.

Discussion: Based on the statistical analysis above, we have the following conclusions. For the multiple-choice questions, users in general performed better with our method than the random method. This indicates that our view tour indeed provides the user with more information about the underlying flow field. For subjective questions, most users agreed that our method better help them detect critical regions and identify the global flow patterns than the random method for the hurricane and solar plume data sets, though the latter one did not have a significant difference. The supernova data set received almost the same rating for both tours due to a simple flow feature which could be easily observed with either method. In terms of identifying critical regions, users performed much better with our method over the random method except for the supernova data set which only contains a single obvious sink-like point at the center. From the post-experiment feedbacks, except for suggestions on animation and interfaces, most users gave positive feedbacks on our method over the random one.

In summary, our view tour indeed helps the user identify and observe internal flow patterns in unsteady flow fields, especially for hidden or occluded features that only exist for a short period of time. Therefore, our view tour could complement the traditional overview tour to provide the user with a more comprehensive exploration experience for large and complex flow fields.

Chapter 8

Pedagogical Visualization Tools for Cryptography Algorithms

8.1 Overview

Though flow field visualization is my major research focus, other topics relevant to visualization also attract my attention. Designing pedagogical tools for cryptography algorithms is just one of these attractions.

Nowadays, data privacy and system security is a major concern in database, computer

[0]The material contained in this chapter has been accepted for publication in *Conference on Innovation & Technology in Computer Science Education 2014*.

network, electronic communication, etc. and *cryptography* has been used to address the security problems for centuries. Many cryptography algorithms have been proposed based on various encryption/decryption schemes. However, since most of such algorithms use a lot of sophisticated mathematics, it becomes a challenge for new learners to gain clear pictures about the overall algorithms. In order to overcome this problem, we developed a set of visualization tools to provide users with an intuitive way to learn and understand these algorithms. I am involved in the development of six visualization tools, namely:DESvisual [78], ECvisual [79], RSAvisual [80], SHAvisual [55], AESvisual, VIGvisual ,which are designed to demonstrate the workflows of the Data Encryption Standard Algorithm, Elliptic Curve based ciphers, the RSA Algorithm, the Secure Hash Algorithm, the Advanced Encryption Standard Algorithm and the Vigenère Cipher, respectively. All of these works except AESvisual which will be submitted soon have been published in top conferences and journals. This Chapter discusses two of these projects, SHAvisual and AESvisual, which are developed by the author in detail. Before this, a brief introduction to other projects is given in this section.

DES and DESvisual: The Data Encryption Standard (*DES*) is a encryption algorithm for electronic data and gained its popularity in early 1970s. It was developed by IBM and published in 1977. DES is now considered to be insecure due to its 56-bit small key size and has been hacked frequently. Though DES is proved insecure and has been abandoned by many applications, the main algorithm is still worth of

studying since it provides a basic framework for many advanced algorithms. In order to provides an intuitive way for the students to learn the DES cipher algorithm, we developed DESvisual: a visualization tool for the DES cipher [78]. Our DESvisual not only helps learners understand the DES but also provides instructors with a visualization tool to teach and demonstrate the algorithm in class. Specifically, our system depicts the primitive operations required by DES with a small-size message (8 or 16 bits) and allows the user to trace through the encryption performed by the system. Furthermore, with a practice mode, the user can study the encryption and decryption step by step and verify the answers in an intuitive way.

Elliptic Curve Cryptography and ECvisual: Elliptic curve cryptography (ECC) is a public-key cryptography approach based on the algebraic structure of elliptic curves [42, 63]. For a public-key cipher, the encryption key is public and differs from the decryption key which is kept secret. We developed our ECvisual: a visualization tool for elliptic curve based ciphers [79] to allow the user to visualize the properties of elliptic curves over the real field and also the operations of elliptic curve based ciphers over a finite field of prime order. Furthermore, various useful functions are provided, such as performing arithmetic operations over a finite field, doing encryption and decryption, and converting plaintext to a point on an elliptic curve. ECvisual also provides a practice mode and allows the user to go over each step and verify their answers. This helps the user understand the primitive operations and how they are used in an elliptic curve cipher.

RSA and RSAvisual: The same as elliptic curve cryptography, RSA is a public-key cryptosystem and being widely used in data transmission. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977 at MIT. The RSAvisual: a visualization tool for the RSA cipher is designed to help learners understand how the RSA algorithm operates, including encryption, decryption, use of the Extended Euclidean algorithm to calculate the private key, and Fermat and Pollard $p-1$ factorization.

Vigenère Cipher and VIGvisual: The Vigenère cipher first appeared in the book *"Traicté des Chiffres"* written by Blaise de Vigenère in 1585. It is a cipher using a series of different Caesar ciphers based on the letters of a keyword. VIGvisual is also designed to visualize the algorithm step by step and facilitate the self-study. It allows the user to animate the Vigenère cipher with cipher tools, all of which are available for the user to practice encryption and decryption with error checking. Furthermore, VIGvisual also helps the user learn how to break the Vigenère cipher. Specifically, VIGvisual uses Kasiski's method and the Index of Coincidence method for keyword length estimation, and the χ^2 method with frequency graphs for keyword recovery, respectively.

8.2 Motivation and Goals

The development of SHAvisual and AESvisual was supported by the National Science Foundation. SHAvisual and AESvisual are designed to be used in classroom and also for self-study for learners to explore the corresponding cryptography algorithms on their own. Specifically, SHAvisual [55] is a visualization tool for demonstrating an advanced SHA algorithm, SHA-512, which is a member of a family of cryptographic hash functions published by the National Institute of Standards and Technology in the early 1990's. This work has been accepted in *Conference on Innovation & Technology in Computer Science Education 2014* as a poster. The Advanced Encryption Standard (AES) is based upon Rijndael, which was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen in 1998 [19]. It has been a federal government standard since 2002 and is now used widely. AESvisual is a visualization tool designed for the AES cipher. It demonstrates all the major steps of AES encryption and decryption along with an overview page to illustrate the global workflow of the algorithm.

SHAvisual and AESvisual have the demo and practice modes. The demo mode visualizes the major algorithm components step by step and the practice mode allows the user to compute the output of each operation and check for the correctness instantly. The demo modes provides instructors with greater flexibility in selecting a lecture

pace for the detailed materials and the study report system helps instructors evaluate the teaching and learning effectiveness. The SHAvisual also provides a full mode to let the user perform full version SHA-512 encryption.

8.3 SHAvisual: A Visualization Tool for SHA-512 Cryptographic Algorithm

8.3.1 System Overview

SHAvisual consists of three major components: Demo mode, Practice mode and Full mode. A separate global view window is available to show the overall algorithm pipeline and also highlight the current procedure in red. Specifically, the Demo mode provides a simplified SHA-512 visualization and is useful for the instructor to demonstrate important operations in the classroom. The Practice mode is designed for students to learn the detailed computations step by step and perform self-study. Further, we also provide a test report system to help instructors verify the learning effectiveness. The Full mode is a full version of the SHA-512 cipher. It takes a plaintext as input and generates the encrypted digest message with major intermediate results shown. Both the Demo mode and Practice mode have multiple subpages and the user may access different subpages by clicking their tab names. Buttons are also provided

to switch between subpages. The Full mode only uses one subpage to demonstrate all the computations. SHAvisual always starts from the Demo mode by default.

8.3.1.1 The Demo mode

The goal of the Demo mode is help the user to understand the SHA-512 algorithm in an intuitive way. To achieve this, it uses shorter length messages and single round so that the user can focus on the essential computations rather than repetitive operations. The Demo mode has five subpages: Message Generation, Workflow Overview, Words Generation, Compression Function and Round Detail. In the following, we will describe each of the subpages in detail:

Message Generation. This subpage demonstrates how to obtain the Augmented Message by expanding the Original Message (plaintext) to the length of a multiple of 256 (1024 originally) bits (Figure 8.1). The user clicks the Random Message button to generate a new random plaintext and the corresponding augmented message will be shown in the bottom. Green, blue and red colors indicate the plaintext, padding field and length field, respectively, of the augmented message. To save space , both messages are shown in hexadecimal. By clicking the Augmented Message, the user will be guided to the Workflow Overview subpage.

Workflow Overview. This subpage offers a general SHA-512 algorithm overview as

Figure 8.1: Message Generation of Demo mode.

shown in Figure 8.2. It illustrates how the final Message Digest is generated using an initial value and the blocks derived from the augmented message. The Initial Value is a 128-bit (512-bit originally) constant defined by the SHA-512 algorithm and used as an input for the first compression function. Each Block is a 256-bit (1024-bit originally) segment of the augmented message and extended to 80 16-bit (64-bit originally) words in the Words Generation stage. The user clicks the Block numbers or the Compression Function button to proceed to the Words Generation or Compression Function subpage.

Words Generation. This subpage demonstrates how the last 64 (in red) out of 80 16-bit (64-bit originally) words are generated from the corresponding block (Figure 8.3), where the first 16 words (in black) are taken from the block. By sliding the row of words horizontally in the upper portion of the page, the user may pick any word (in blue) of the last 64 words to check its generation procedure. Four words used for computation are shown in the middle. The two RotShift buttons are used

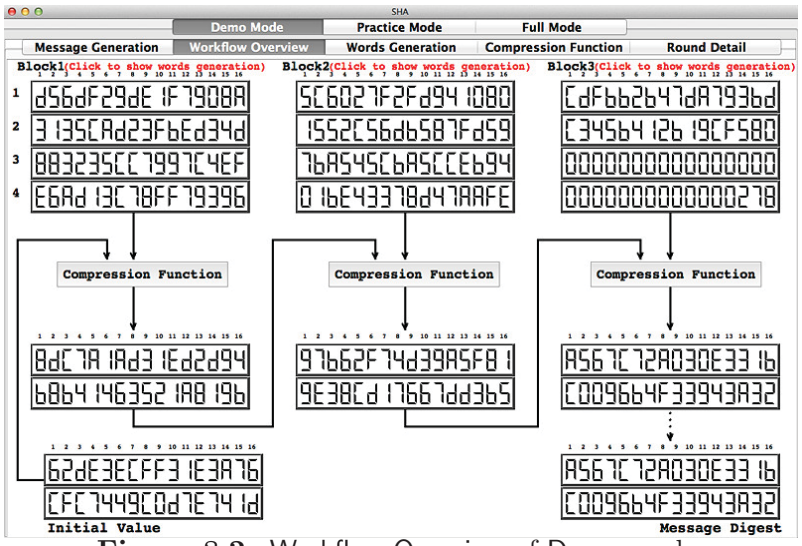


Figure 8.2: Workflow Overview of Demo mode.

to check the detailed operations of the generation. Figure 8.4 shows the snapshot of the corresponding window. Each word is then used in one round (80 totally) of the corresponding Compression Function.

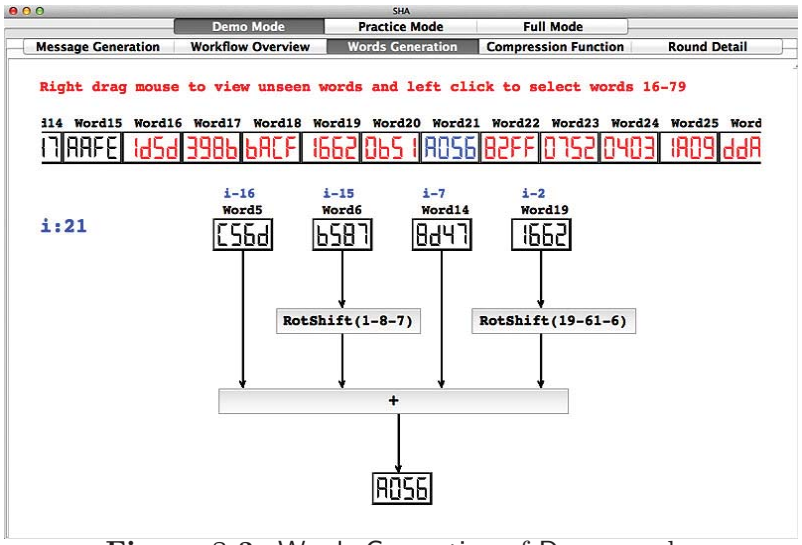


Figure 8.3: Words Generation of Demo mode.

Compression Function. This subpage visualizes the pipeline of the Compression Function as shown in Figure 8.5. The input words A-H are either from the previous

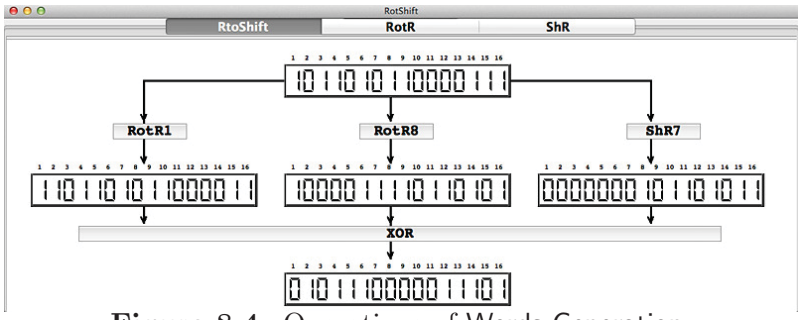


Figure 8.4: Operations of Words Generation.

compression function or from the initial value in the Workflow Overview (for the first compression function). The output is used for the next compression function or forms the final message digest (for the last compression function). Figure 8.2 shows the relationship among different compression functions. Word0 is from the Words Generation while Key0 is one of the 80 16-bit (64-bit originally) constants defined by the algorithm. They will be used in round 0. Eighty similar rounds are needed for the original compression function with one word and one key for each round. For a clear demonstration, we only provide one round in the Demo mode. The user clicks the Round0 button to see round details in the Round Detail subpage.

Round Detail. This subpage shows the computations in a round. A snapshot of the subpage is shown in Figure 8.6. The Round box in the upper half of this page shows the mapping between the input (A-H) and output of the corresponding round. The lower portion demonstrates the computation of the two new words X and Y in the output. Input A-H, Word0 and Key0 are taken from the corresponding Compression Function. The user may click the Majority, Rotation and Condition buttons in the Mixer boxes to check the details of computation.

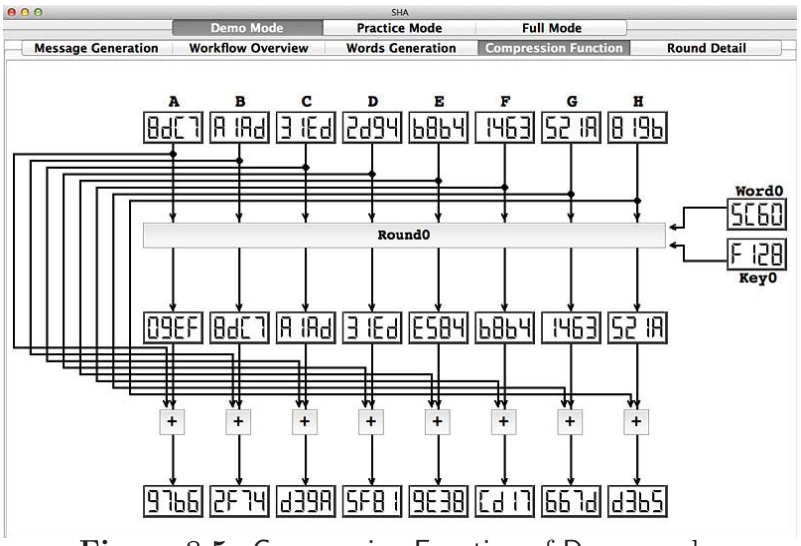


Figure 8.5: Compression Function of Demo mode.

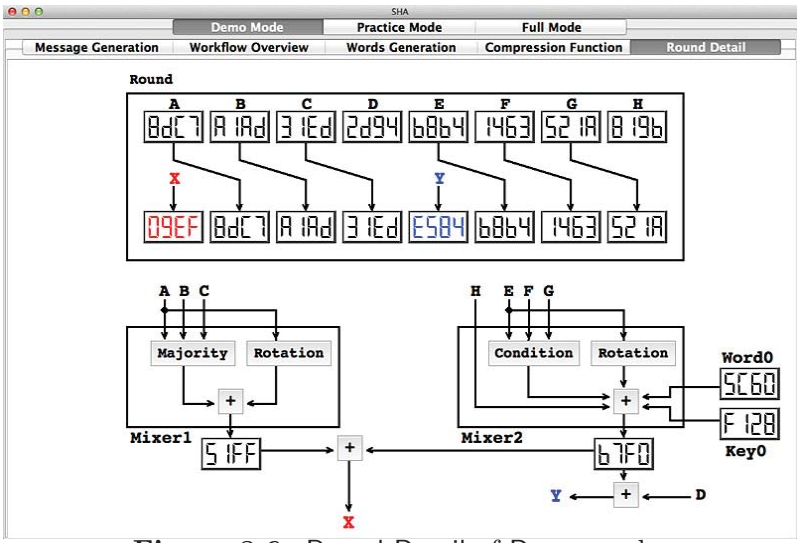


Figure 8.6: Round Detail of Demo mode.

8.3.1.2 The Practice mode

Figure 8.7 shows the snapshot of the Practice mode. We follow the same structure used in the Demo mode so that the user can practice each operation in the same manner demonstrated in the Demo mode. However, all results in this mode are

hidden and a correct answer is required to advance to the next step. To start the process, the user may press the Start Practice button. A dialogue window will pop up to briefly describe the current question. The user should enter an answer and click the Check Ans button. SHAvisual will then verify the input and display "Correct!" if the answer is correct and "Wrong! Try it again!" otherwise. The user should enter a new answer if the current one is wrong. A Show Ans button is provided to show the correct answer and let the user skip the current question. We also provide a simple hexadecimal-binary converter to assist the user's computation. A Completion Report window (Figure 8.8) will be shown after the user finishes all questions. The report records the answer to each question using "Correct", "Wrong" or "Show Ans" according to the user's action. This report may be sent to the instructor to check the student completion rate and evaluate the learning effectiveness.

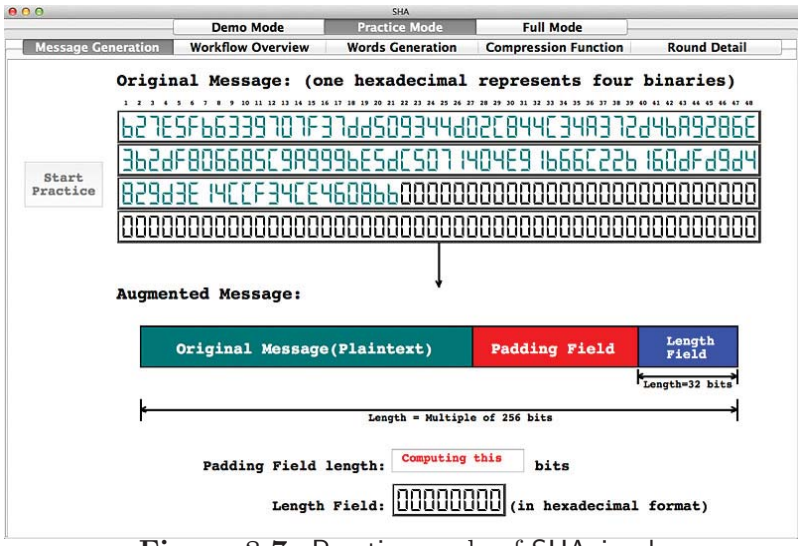


Figure 8.7: Practice mode of SHAvisual.

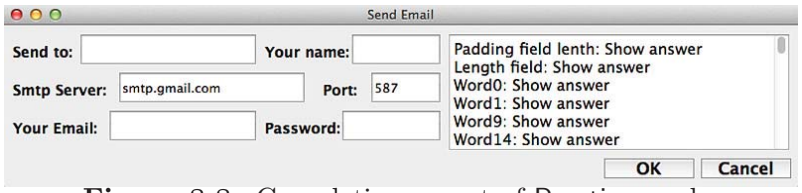


Figure 8.8: Completion report of Practice mode.

8.3.1.3 The Full mode

The Full mode provides a full version SHA-512 cipher (Figure 8.9) to encrypt a given input string. We offer the user the freedom to enter an input string or click the Random button to generate a random one. The Clear button allows the user to clear the input and reset the computations. By clicking the Confirm button, the encryption is performed and the final encrypted message digest along with important intermediate results will be shown when the process is finished.

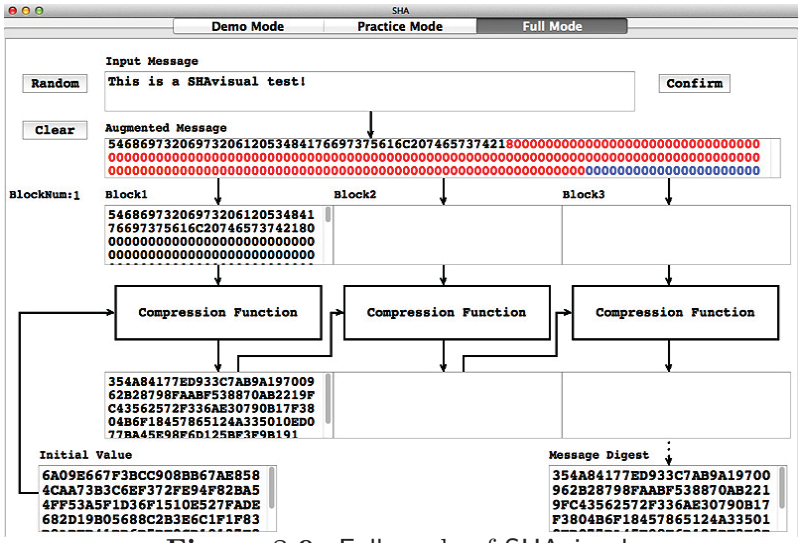


Figure 8.9: Full mode of SHAvisual.

8.3.2 Evaluation

We conducted a survey to evaluate the effectiveness of SHAvisual. The tool was released to students one week before the survey. The survey consists of two major components, 12 questions listed in Table 8.1 and 11 write-in comments. The first nine questions (Q1-Q9) evaluate the effectiveness of SHAvisual (EEQ) and the other three (Q10-Q12) investigate the use of SHAvisual (UIQ). The EEQ questions all have the same set of choices: 1:strongly disagree, 2:disagree, 3:neutral, 4:agree, and 5:strongly agree. The choices for Q10 are 1:less than 5 mins, 2:5-10 mins, 3:10-15 mins, 4:15-30 mins, 5:over 30 mins. The choices for Q11 are 1:only once, 2:1-3 times, 3:3-5 times, 4:5-10 times, 5:over 10 times. The choices for Q12 are 1:less than 5 mins, 2:5-15 mins, 3:15-30 mins, 4:30-60 mins, 5:over 1 hour. We collected 24 valid survey forms from two disciplines: 19 in computer science and software engineering (CS) and five in computer engineering (CpE).

8.3.2.1 General Discussion

Table 8.2 shows the mean (μ) and standard deviation (σ) for each question. For EEQ (effectiveness evaluation questions), the highest score of 4.2 was given to Q8, which indicates that students highly agreed that SHAvisual helped them understand the SHA

Table 8.1
Survey Questions.

ID	Question
Q1	Demo mode helped me better understand the work flow of the SHA cipher
Q2	Demo mode was helpful for my self-study
Q3	Practice mode helped me remember SHA encryption
Q4	Full mode helped me understand how the SHA cipher encrypts a full-length message
Q5	Full mode provided me a simple tool to do SHA encryption
Q6	Global view helped me locate the current demonstrated operation
Q7	Using SHAvisual I was able to identify the parts of the SHA cipher that I did not understand before
Q8	I was able to better understand the SHA algorithm with SHAvisual
Q9	The SHA software enhanced the course
Q10	How long did it take to understand SHA Algorithm with SHAvisual
Q11	How often did you use SHAvisual
Q12	How long did you use SHAvisual totally

algorithm better. Q1, Q3 and Q7 all received the same high score of 4.0, suggesting that both the Demo mode and Practice mode had positive impact on student learning. Except for Q6, other questions were rated in the range from 3.5 to 3.9, which is still above the neutral rating (3.0). Q6 received the lowest score of 3.3, suggesting that the global view sightly helped students identify the relation between the current operation and the overall algorithm. Therefore, we conclude that although the rating of EEQ varied among questions with standard deviations in a small range from 0.7 to 0.9, the general trend was positive.

In terms of UIQ (usage investigation questions), Q12 got a very high average (4.6), indicating that students used the software for nearly an hour. The average of Q10

Table 8.2
Mean μ and Standard Deviation σ.

	Q1	Q2	Q3	Q4	Q5	Q6
μ	4.0	3.8	4.0	3.5	3.6	3.3
σ	0.8	0.8	0.9	0.8	0.8	0.8
	Q7	Q8	Q9	Q10	Q11	Q12
μ	4.0	4.2	3.9	3.7	2.3	4.6
σ	0.8	0.8	0.7	1.1	1.1	0.7

was 3.7, which means that the majority of the students took less than 15 minutes to understand the SHA algorithm. However, most students only used the tool a few times as indicated by the average 2.3 of Q11. Both the standard deviations of Q10 and Q11 were slightly larger than 1.1. Table 8.3 lists the distributions of answers for UIQ. For Q10, around 33% of all students took 10 to 15 minutes to understand the algorithm with the tool while another 33% took more than 30 minutes. The distribution of Q11 indicates that more than 90% of all students used the tool less than five times. Q12 suggests that more than two-third of the students (67%) used the tool for over an hour while a quarter of them used it 30 minutes to one hour. We also applied the Spearman rank test to further investigate the correlations among UIQ. The null hypothesis is that there is no correlation under the level of significance $\alpha = 0.05$. Based on the results, only Q10 and Q12 had a significant positive correlation with the p-value being 0.002. This is not surprising since the students who took more time to understand the SHA algorithm with the tool may also spend longer time on the tool.

Table 8.3
Usage Answer Distributions.

	Choice1	Choice2	Choice3	Choice4	Choice5
Q10	0.00	0.17	0.33	0.17	0.33
Q11	0.21	0.42	0.29	0.00	0.08
Q12	0.00	0.00	0.08	0.25	0.67

8.3.2.2 Further Statistical Analysis

We also investigated whether the use of SHAvisual (Q10-Q12) would have an impact on student evaluation for the EEQ questions (Q1-Q9). To this end, student reactions are divided into two groups based on Table 8.4based on questions Q10 to Q12. The null hypothesis for this study is: the time spent on understanding the SHA algorithm (Q10), the number of times using this tool (Q11) and the total time spent on this tool (Q12) have no impact on student reactions on the EEQ questions. The level of significance is $\alpha = 0.05$.

Table 8.4
Student Reactions Grouping.

	Group1	Group2
Q10	$\leq$ 15 mins (12, 50%)	$>$ 15 mins (12, 50%)
Q11	$\leq$ 3 times (15, 62%)	$>$ 3 times (9, 38%)
Q12	$\leq$ 1 hour (8, 33%)	$>$ 1 hour (16, 67%)

Table 8.5 shows the p-values of our ANOVA (ANalysis Of VAriance) study. The two smallest p-values are 0.054 from the Q6-Q10 pair and 0.006 from the Q5 -Q12 pair. Since all other p-values are larger than the chosen $\alpha = 0.05$, the null hypothesis

cannot be rejected. Therefore, we have strong evidence showing that the student reactions, except for the indicated two cases, were not affected by the time spent on understanding the SHA algorithm (Q10), the number of times they used the tool (Q11), and the time spent on using the tool (Q12).

Table 8.5
ANOVA p-values for Three Groupings.

Grouping Question	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
Q10	0.807	0.628	0.488	0.611	1.000	0.054	1.000	0.603	0.219
Q11	0.485	0.803	0.859	0.794	0.353	0.839	0.600	0.789	0.639
Q12	0.676	0.864	0.871	0.097	0.006	0.348	0.096	0.191	0.831

Q6 asked students if the global view helped them locate the current demonstrated operation. The p-value of the two groups based on Q10 for Q6 is 0.054, which is barely larger than the chosen level of significance 0.05. Therefore, statistically we cannot reject the null hypothesis. In other words, the two groups which took no more than and over 15 minutes to understand the SHA algorithm ($\mu = 3.6$ and $\mu = 3.0$), respectively, had no significant difference response to Q6. Q5 asked students if the full mode offered a simple tool to perform SHA encryption. Grouping based on Q12 showed a p-value 0.006, which is smaller than the chosen level of significance $\alpha = 0.05$ and, hence, the null hypothesis is rejected. This means that the group of using the tool for more than one hour ($\mu = 3.9$) had a significant different response to this question from the group of using the tool for no more than one hour ($\mu = 3.0$).

This ANOVA analysis treated each question individually. To address the possible

dependence among the nine EEQ questions, we also conducted a MANONA (Multivariate ANOVA) analysis by considering all nine questions simultaneously. Wilk's lambda test suggested that we cannot reject the null hypothesis (i.e., no group differences) under the chosen level of significance $\alpha = 0.05$. Based on these findings, we conclude that student reactions are generally independent of the time they spent on understanding the SHA algorithm, and the time (and the number of times) they spent on using the software.

8.3.2.3 Student Comments

We also gathered the student comments of the 11 write-in questions regarding the future improvement. Particularly, we focused on the following issues: whether the restrictions of small-size messages and the single round demonstration had an impact, whether the Words Generation module is useful, whether the Compression Function module needs improvement, whether the Round/Mixer module is good enough, the evaluation of Demo, Practice and Full modes, and software installation issues.

Based on the student comments, we found that the restrictions of the small-size messages and single round demonstration did not affect their learning of the SHA algorithm. Students said "*It gave good insight to blocks*", "*A larger message size would have make it more confusing*", "*Concepts are the same even if the size is small*", and "*It was focused and made the inner workings of the round clearer*". One

student also suggested that defining a minimum message size would be useful while another student thought that a second round may be helpful although the idea was already clear after the first round.

The feedbacks of the Word Generation module were nearly all positive with comments like: "*Word pattern was pretty well shown, which enhanced my learning*" and "*The fact that it was visually laid out was helpful*". We also received some suggestions, such as using a bubble dialog to show detailed explanation. The Compression Function and Round/Mixer modules followed the same trend. The majority of students felt these modules were helpful by pointing out that "*The mode was illustrative*" and "*That would have been extremely confusing as just a formula and the visual aspect helped*". One student mentioned that the Compression Function module was a little confusing due to a "deep nesting" structure; however, this student also agreed that this was the nature of the algorithm.

For the Practice mode, students agreed that it was effective and indicated "*I liked that you are able to step through the process*", "*Clearly marked + Straightforward*", "*Well laid out and easy to follow*", "*Most effective component*", and "*It helped me understand SHA*". We also received some improvement suggestions, such as adding a "hint" button to provide a brief reminder and building more connections between input and output windows.

The Full mode received fewer comments since most students never used it. This is not

surprising because SHAvisual was released right before the evaluation and students were required to use only the Demo and Practice modes as homework. Thus, there was limited time for students to have a more comprehensive use of the tool. However, we still saw some positive comments, for example: "*It was neat to be able to use the 'real deal'*".

Compared with blackboard work, students nearly all agreed that the Demo mode was more useful for them to learn the SHA algorithm. Typical comments were "*I was able to use it on my own later to reinforce what I learned in class*", "*I like the more dynamic nature*", "*It definitely helped following a program rather than using the blackboard. Visualization would make it even better*" , "*It was easier to see how items connected with each other*", "*The best part was that it really enforced where the data came from and what was done to it*", and "*The structure of the algorithm is very nested and the demo gave a good overview and let you see details of each piece*". Therefore, we believe that SHAvisual indeed provided students with an effective way to learn the algorithm.

Students also gave some general comments for improvement. For example, they suggested adding a "help" button to explain each step, integrating the "Dec-Hex" conversion into the current converter, and giving a brief explanation when the answer is wrong in the Pactice mode.

In summary, we conclude that our SHAvisual has fulfilled its initial purpose, helping

students learn and instructors teach the SHA algorithm effectively. Furthermore, with the above suggestions we will be able to modify SHAvisual accordingly and improve its efficiency in the near future.

8.4 AESvisual: A Visualization Tool for the AES Cipher

8.4.1 System Overview

AESvisual supports Windows, MacOS and Linux. It consists of two major components: Demo mode and Practice mode. The Demo mode displays both the encryption and decryption operations of the AES algorithm and each operation contains multiple pages to demonstrate the major steps. The Practice mode allows students to learn the detailed computations step by step and perform self-study. Only the encryption is available in this mode since the decryption follows the same workflow in a reversed order. The tool also provides a test report system to help the instructor verify the learning effectiveness.

8.4.1.1 The Demo mode

AESvisual always starts from the Demo mode. It has four subpages: Overview, Encryption, Decryption, and Key Expansion. The overview subpage provides a global overview for the algorithm and also illustrates the relationship between encryption and decryption. Figure 8.10 shows the snapshot of this page. Both Encryption and decryption involve ten rounds and only the first round (highlighted in the red) is shown. By clicking the Go buttons in the Round 1 frames, the user will proceed to the Encryption or Decryption subpage. A Expand Key button is also provided to allow the user to advance to the Key Expansion subpage.

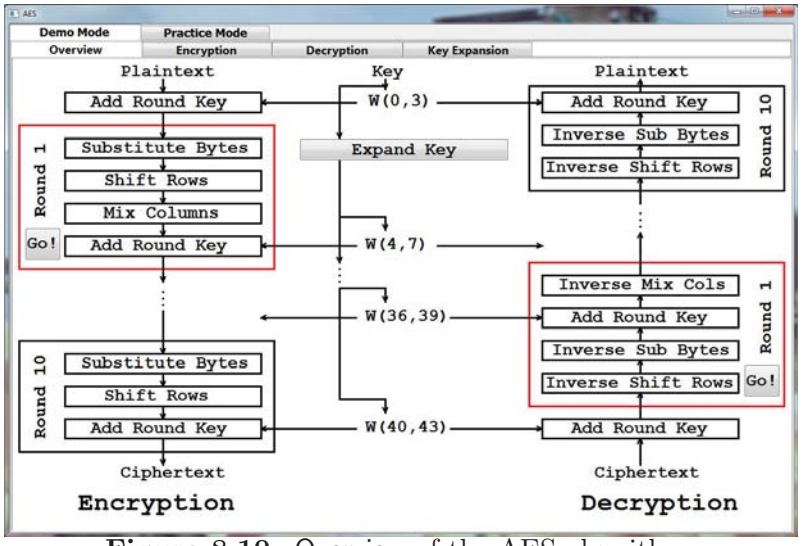


Figure 8.10: Overview of the AES algorithm.

ENCRYPTION: This subpage demonstrates the four major steps of the Round 1 for the encryption operation: Substitute Bytes, Shift Rows, Mix Columns and Add Round Key. Each of these steps has its own subpage.

Substitute Bytes. This subpage visualizes how the 128-bit original Plaintext is processed using Add Round Key and S-box transformation, as shown in Figure 8.11. By clicking the Random button, the user may generate a new random plaintext-key pair. The generated key is then expanded in the Key Expansion subpage to create 44 32-bit words. To check the key expansion procedure, a Expand Key button is provided to allow the user to proceed to the corresponding subpage. The user may also click the Add Round Key to observe how the plaintext is added with the first four words W(0, 3). The output is then transformed with the S-box transformation. The user may select one element (in red) in the output matrix of the Add Round Key and then click the Check S-box button to see the details of the transformation (Figure 8.12). The corresponding element in the result is highlighted (in green) and the selected row and column are also displayed above the Check S-box button. The result from the transformation is then used as input matrix to the Shift Rows subpage.

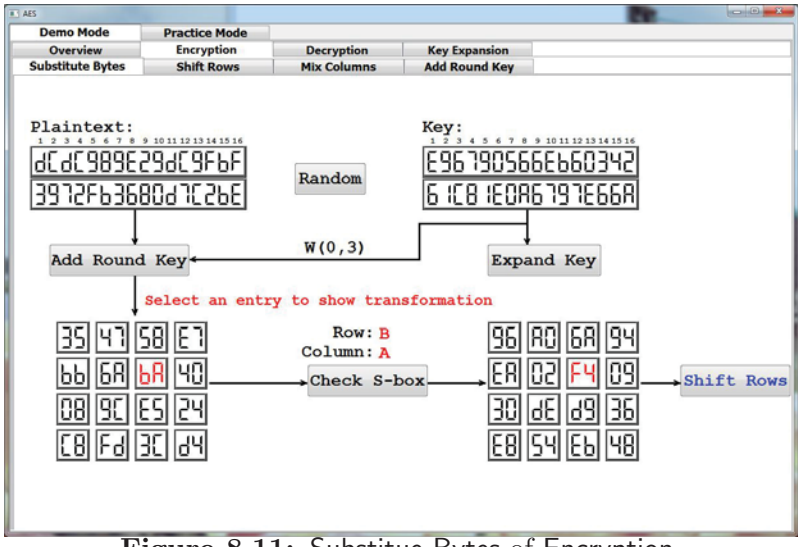


Figure 8.11: Substitue Bytes of Encryption.

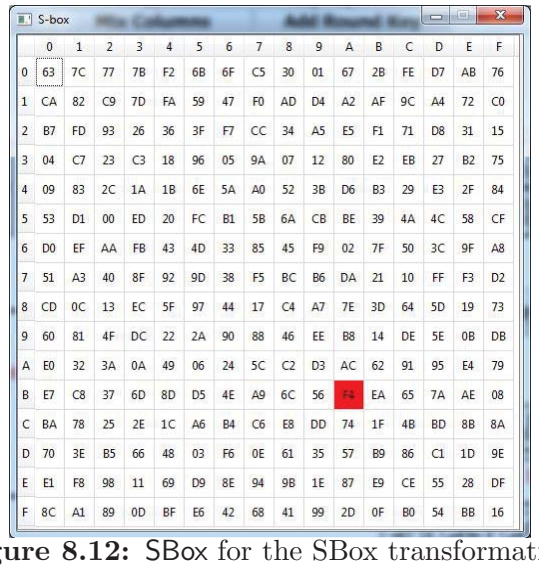


Figure 8.12: SBox for the SBox transformation.

Shift Rows. This subpage demonstrates how the input matrix is transformed by performing row-based byte rotation (Figure 8.13). The result goes to the Mix Columns subpage.

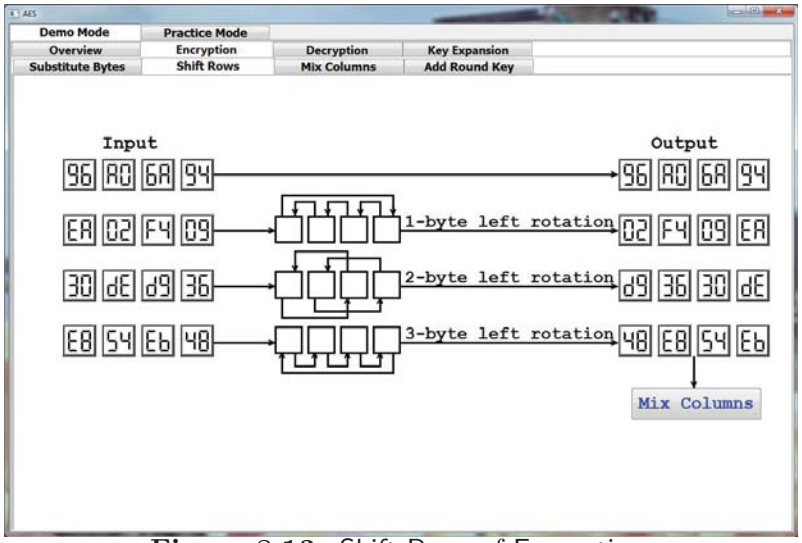


Figure 8.13: Shift Rows of Encryption.

Mix Columns. This subpage shows how the output matrix is obtained by multiplying

the input matrix with a given matrix in GF(2^8) (Figure 8.14). If the user selects a column (in red) of the input matrix, the corresponding column of the output matrix will be highlighted (in green). The lower half of the page shows the details of the matrix multiplication for the selected column. The user may click the $\times$ and $+$ buttons to further explore the corresponding GF(2^8) multiplication and addition operations in detail, respectively. Figure 8.15 (a) and (b) show the snapshots of the two operation windows. For an intuitive illustration, we use binary presentation in these windows. The output matrix is then feed as input for the Add Round Key subpage.

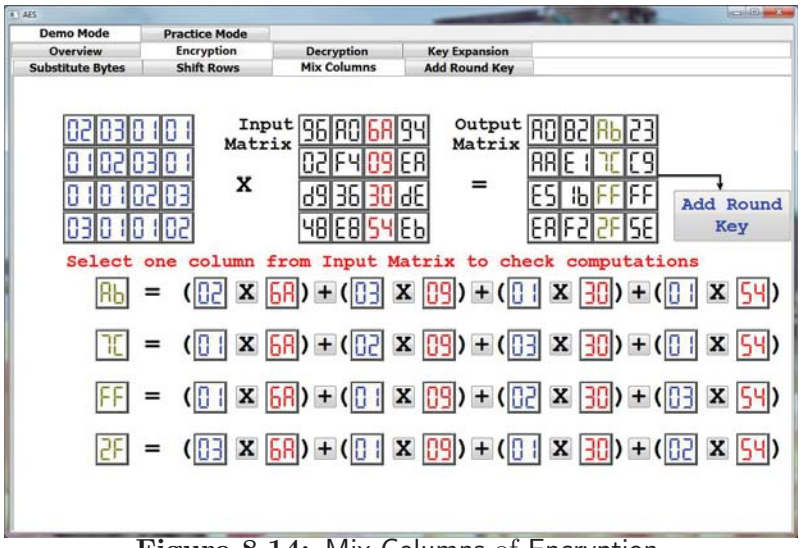
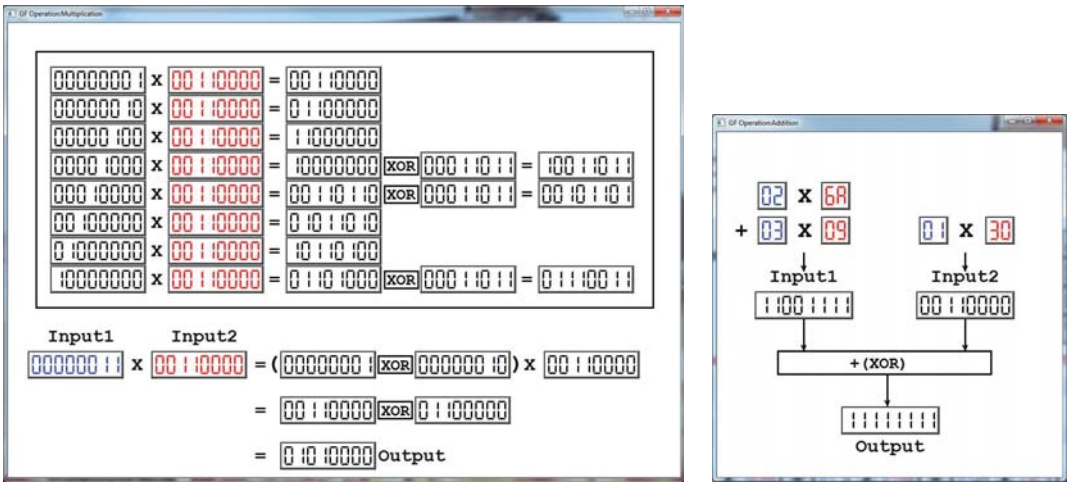


Figure 8.14: Mix Columns of Encryption.

Add Round Key. This subpage demonstrates how the input matrix is XORed ($\oplus$) with the word matrix element-by-element. Figure 8.16 shows the corresponding page. The user may select one element from the input matrix (in red) or the word matrix (in blue) and the corresponding element in the output matrix will be highlighted (in green). The lower half of this subpage shows the corresponding exclusive disjunction



(a) (b)

Figure 8.15: (a) G(2^8) Multiplication. (b) G(2^8) Add.

operation in binary format. The final ciphertext after the ten rounds of the encryption process is also shown in the lower right corner of this page.

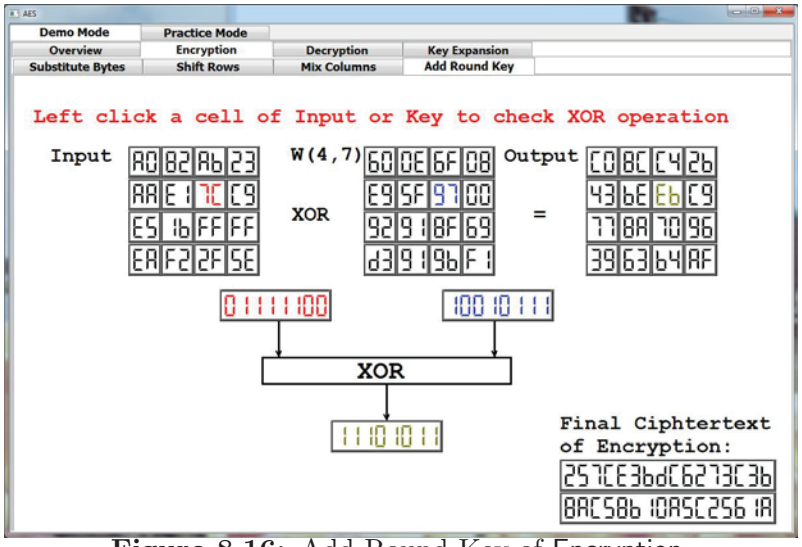


Figure 8.16: Add Round Key of Encryption.

DECRYPTION: The Decryption subpage also consists of four subpages showing the four major steps of the first round of the decryption. It starts with the Shift

Rows subpage shown in Figure 8.17, followed by Substitute Bytes, Add Round Key, and Mix Columns. The ciphertext in the Shift Rows is taken from the encryption and the user may click the Add Round Key and Substitute Bytes buttons to advance to the corresponding subpages. The decrypted plaintext after ten rounds is shown in the lower right corner of the Mix Columns subpage (Figure 8.18). The Substitute Bytes and Add Round Key subpages are the same as in the Encryption subpage.

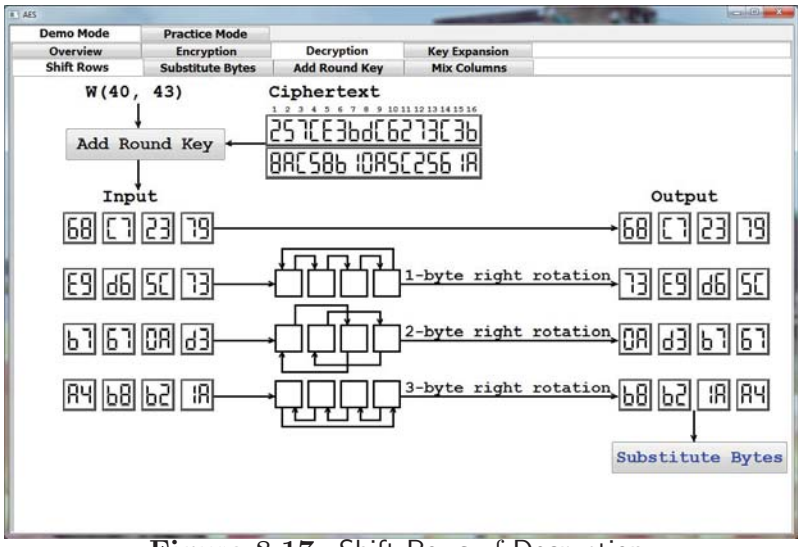


Figure 8.17: Shift Rows of Decryption.

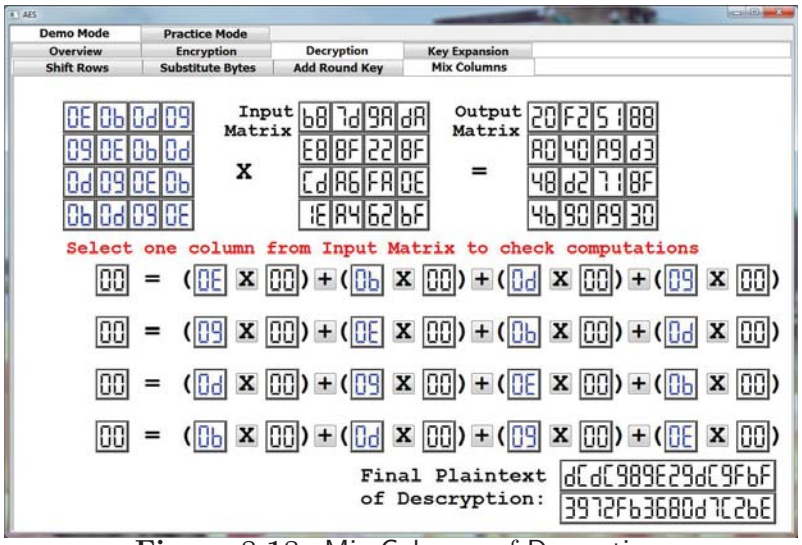


Figure 8.18: Mix Columns of Decryption.

KEY EXPANSION: This subpage demonstrates how the original 128-bit key is expanded to 44 words (Figure 8.19). These words are used in the ten rounds (four words per round) and one initial step for both encryption and decryption. Figure 8.10 provides an intuitive illustration of the workflow. The first four words (in black) are directly derived from the input and all other words (in red) are generated from them. The user may right drag the mouse to move words back and forth horizontally and click a single word (in blue) to check the word generation procedure. The lower portion of this subpage demonstrates how the four output words with the selected word in blue are obtained using the four input words. The user may click the G button to proceed to the "OperationG" window. Figure 8.20 shows the snapshot of the corresponding window. We also allow the user to check the XOR ($\oplus$) operations (Figure 8.21) by clicking the XOR buttons.

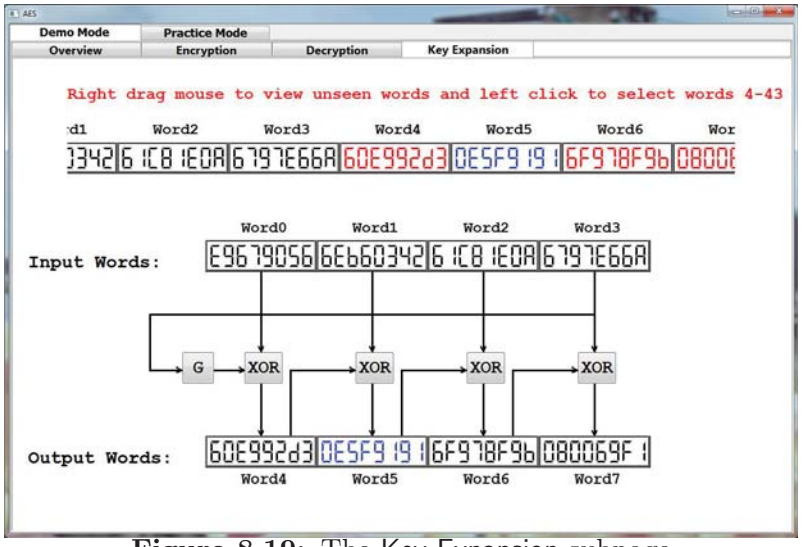


Figure 8.19: The Key Expansion subpage

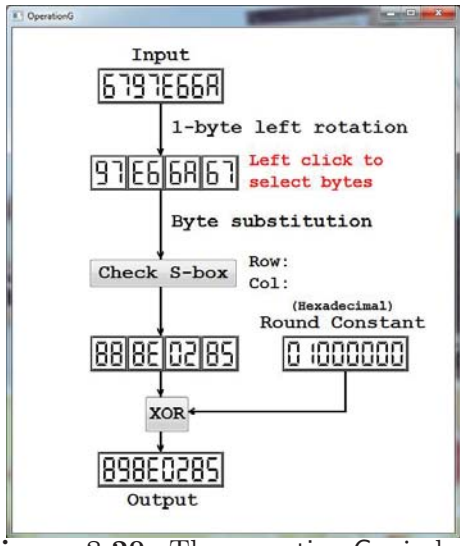


Figure 8.20: The operation G window.

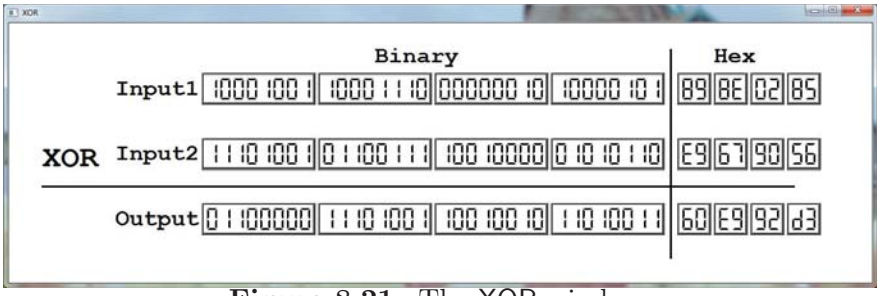


Figure 8.21: The XOR window.

8.4.1.2 The Practice mode

The Practice mode follows the same structure of the Demo mode with only encryption supported. A snapshot of this mode is shown in Figure 8.22. The user may step through each computation; however, all results are hidden and a correct answer is required to advance to the next step. Clicking the Start button allows the user to start a new session by generating a new plaintext-key pair. Then a dialogue window will pop up to briefly describe the current question and ask the user to enter the

answer. A Check Ans button is provided to let the user verify the answer correctness and the user may enter a new one if the current answer is wrong. We also provide a Show Ans button to allow the user to skip the current question by showing the correct answer. To assist the computation, a simple hexadecimal-binary converter is also provided. After the user finishes all the questions, a Completion Report window recording the user's answer to each question will pop up. The words Correct, Wrong or Show Ans are used to indicate the corresponding answer was correct, incorrect or skipped, respectively. The instructor may use this report to check the student completion rate and evaluate the learning effectiveness.

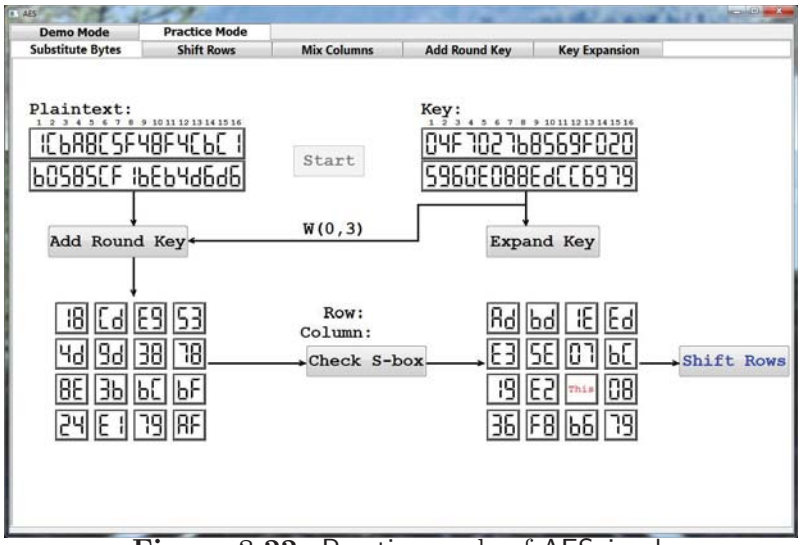


Figure 8.22: Practice mode of AESvisual.

8.4.2 Evaluation

We conducted a survey which took one week to evaluate the effectiveness of AESvisual. Our survey consists of two parts: a set of 12 questions and 11 write-in comments. Choices available are 5:strongly agree, 4:agree, 3:neutral, 2:disagree, and 1:strongly disagree. We collected 23 valid forms. The distribution of majors was as follows: 1 in computer network and system administration (CNSA), 8 in electrical and computer engineering (EECE), 9 in computer science, 2 in mathematics, 1 in chemical engineering, and 2 undeclared.

8.4.2.1 General Discussion

We set $\alpha = 0.05$ as the level of significance for all statistical decisions. We found that the students used AESvisual 2.6 times on average during the period of evaluation and the average time they spent on the software was 34.3 minutes with standard deviation and confidence interval 18.9 and (26.4, 41.0).

A summary of the remaining questions is listed in Table 8.6. The first three questions Q1, Q2 and Q3 received means 4.04, 4.09 and 3.83, standard deviations 0.64, 0.67 and 0.98, and confidence intervals (3.79, 4.30), (3.82, 4.35) and (3.45, 4.20), respectively. This suggested that AESvisual indeed helped students better learn the encryption

Table 8.6
Survey Questions.

ID	Question
Q1	The Demo mode helped better understand encryption workflow
Q2	Demo mode was helpful for my self-study
Q3	The Demo mode was helpful for self-study
Q4	The "Mix Columns" module helped understand multiplication and addition in GF(2^8)
Q5	The Practice mode helped remember how to encrypt and decrypt
Q6	AESvisual helped identify the parts of AES that I did not understand
Q7	AESvisual helped better understand AES
Q8	AESvisual enhanced the course
Q9	Is AESvisual easy to use

and decryption flow and for self-study. On the other hand, the Practice mode (Q5) was rated slightly lower with mean, standard deviation and confidence interval 3.70, 0.97 and (3.31, 4.09), respectively. We also found that students gave Mix Columns module (Q4) low rate with mean, standard deviation and confidence interval 3.26, 1.32 and (2.73, 3.79). This may be caused by the fact that the Mix Columns component requires students to have a deeper understanding of GF(2^8) arithmetic to completely comprehend the workflow, which may not be very easy for some students. On the other hand, the low rating of Q4 may also indicate that our design of AESvisual and the way we present the materials require some improvement to be more effective. For example, we found that a few students were not satisfied with the diagram-based design and preferred to have an algorithmic view. The next three questions Q6, Q7 and Q8 received good ratings with means 3.87, 3.91 and 3.78, standard deviations 0.97, 0.79 and 0.90, and confidence intervals (3.48, 4.26), (3.60, 4.23) and (3.42, 4.23), respectively. This indicated that AESvisual helped students better understand

the AES algorithm and that AESvisual did enhance the course. Finally, the easy to use question Q9 was rated relatively low with mean 3.48, standard deviation 0.95 and confidence interval (3.10, 3.86). In our opinion, this may be due to the complexity of the GF(2^8) arithmetic and too many subpage/step switching.

8.4.2.2 Further Statistical Analysis

We first investigated the relation among questions and found that the ratings of questions are loosely related to each other. The correlation between every pair of questions was positive. The lowest correlation was 0.18 between Q8 and Q9, which indicated "whether AESvisual enhanced the course" is mostly independent of "whether AESvisual is easy to use". The highest correlation was 0.77 between Q3 and Q7, which suggested that the helpfulness of the Demo mode for self-study and the helpfulness of AESvisual to better understand AES were closely related. The correlation between Q1 and Q2 was 0.63, indicating the ratings for the Demo mode to better understand encryption workflow and decryption workflow were moderately related to each other.

We also investigated the reaction from different disciplines. Students were grouped into three groups: computer science (CS), electrical and computer engineering (EECE), and students from other departments (non-CS). Since the questions may correlate with each other, the questions were also grouped into three groups: (1) Q1, Q2, Q3: the Demo mode was helpful, (2) Q6, Q7, Q8: AESvisual was helpful, and (3)

all other questions in a single group. We applied MANOVA (Multivariate ANOVA) to study the differences among the three student groups on each of the three questions groups. We also applied ANOVA to investigate the difference among all three student groups on each single question.

We used the general linear model (GLM) of R to perform all tests. The p-values for the three groups were 0.72, 0.75 and 0.87, respectively. This indicated that the ratings from students in different groups did not vary significantly. The ANOVA result on each single question did not suggest any significant difference either, with the smallest p-value being 0.45 for Q7. In addition, we investigated the difference between CS and EECE using MANOVA on the same question groups and ANOVA on each question. The p-values for the three groups were 0.49, 0.31 and 0.78, indicating that the ratings from CS and EECE did not vary significantly. We did not find significant difference on any single question either using ANOVA, with the smallest p-value being 0.21 for Q7.

8.4.2.3 A Test Score Comparison

To evaluate the effectiveness of our AESvisual, we preformed a test score comparison. First, a quiz of six problems that address all aspect of the AES cipher was given after the classroom lecture. Then, we introduced AESvisual to students and made the software available. One week later a second quiz was given. The quiz problems were

similar to those of the first, which covered Substitute Bytes, Shift Rows, Mix Columns, Add Round Key and Key Expansion. Both quizzes had a full score of 6 points (i.e., one point per problem). We collected 37 and 36 papers from the first quiz and second quiz, respectively. The results for the two quizzes are shown in Table 8.7. The t-values of comparing the means obtained in various t-tests were all larger than 3 with p-values around 0.003, and Cohen's d is 0.73. Thus, the difference between the means is significant and the effect size is reasonably large. As a result, we concluded that our AESvisual did have a significant impact on student learning of the AES algorithm.

Table 8.7
Test Scores.

	Quiz1	Quiz2
Mean	3.32	4.17
St. Dev	1.23	1.13
CI	(2.93, 3.72)	(3.80, 4.54)

8.4.2.4 Student Comments

We also collected results of the 11 write-in questions asking students to make suggestions for further development. Specifically, we focused on the following issues: whether only doing the first round of the AES algorithm would be sufficient, whether the Substitute Bytes, Shift Rows, Mix Columns, Add Round Key and Key Expansion modules are helpful, the usefulness of the Practice mode, whether the Demo mode is more useful than black-board work, whether new features should be added, and

software installation issues. We found that students uniformly agreed that only doing the first round of the AES algorithm is sufficient. For the module evaluation, only the Mix Columns module received negative comments. Students indicated that the Substitute Bytes, Shift Rows, Add Round Key and Key Expansion modules were straightforward. Typical comments were "It (Substitute Bytes) was explanatory and did enhance my learning", "The diagrams (of Shift Rows) made it very easy to learn", "It (Add Round Key) did not enhance my learning as much as other modules but it was still helpful", "This part was hard for me to figure out until I used the simulation", and "This section greatly enhanced my learning by visually showing the full key expansion procedure and operation".

The Mix Column module was rated the lowest at 3.26. Thus, student comments may provide more information of the possible problems. In general, students felt that the Mix Columns component is the most difficult part of the AES algorithm. Reactions were mixed. Typical positive comments were: "Helped me understand what I was doing wrong the first time I did the assignment", "It made matrix multiplication easier to grasp", and "The actual process is hard to understand but the tool helped break down the steps and was very helpful to learning". Typical negative comments were "The multiplication steps are still complicated" and "This is really the only hard part of AES, and the program did not help. (Neither the book nor the program explain multiplication in GF(2^8) field.)". In general, those who provided negative comments indicated that AESvisual did not help step through and did not explain

the multiplication and addition over GF(2^8) well. The textbook [84] explains GF(2^8) arithmetic with polynomials and provides several examples step-by-step. However, this is not the focus of AESvisual.

Some students believed that the Demo mode would be sufficient and they did not use the Practice mode. The following has some typical comments: "I think it is a useful way for some people to visualize it, but I don't learn that way" and "Pretty great. It has a nice step-by-step implementation".

As for the question "if the Demo version helped the students follow the AES algorithm better than the use of the blackboard", most students believed it is useful with typical comments like "I think it did because I learn better visually, which is what this tool provided. Watching values change instantaneously helped", and "The Demo mode version did help me more than the use of the blackboard". On the other hand, a few students suggested that the use of blackboard would help them take notes: "It helped, but being told about how it works and writing it out helped equally", and "I feel you couldn't have one without the other. A basic intro is needed before demoing the software".

Based on the student comments, we conclude that our AESvisual indeed helped students learn the AES algorithm, especially for understanding the Substitute Bytes, Shift Rows, Add Round Key and Key Expansion modules.

Chapter 9

Distributed System and Tiled Display Wall

9.1 Overview

Since the size of many visualization data is extremely large, big data processing becomes a new research trend in visualization recently. Therefore, designing algorithms to process such data with least space and time cost by utilizing parallel computing and distributed systems forms my another minor research interest. In order to well coordinate multiple processors to work efficiently, smart resource scheduling strategies

[0]The material contained in this chapter has been accepted for publication in *IS&T/SPIE Conference on Visualization and Data Analysis 2015*.

should be carefully designed. Furthermore, since workload balance among processors will significantly effect the system performance, how to partition the original data becomes a crucial problem. As an attempt, we develop a framework to visualize big visualization data, e.g., climate and astronomy data on the fly and enable user interaction for the iGraph: A Graph-Based Technique for Visual Analytics of Image and Text Collections [31]. This work has been published in *IS&T/SPIE Conference on Visualization and Data Analysis 2015*. Section 9.2 briefly introduces the iGraph on a single machine and Section 9.3 discusses how we extend this work to a distributed system.

9.2 iGraph Introduction

With the booming of digital cameras, image archiving and photo sharing websites, browsing and searching through large online image collections has become a notable trend. Consequently, viewing images separately as individuals is no longer enough. In many cases, we now need the capability to explore these images together as collections to enable effective understanding of large image data. Another notable trend is that images are now often tagged with names, keywords, hyperlinks and so on. Therefore, solutions that can nicely integrate images and texts together to improve collective visual comprehension by users are highly desirable.

Therefore, Yi et al.[31] developed iGraph, a visual representation and interaction framework to address the increasing needs of browsing and understanding large image and text collections. These needs include the following. First, when relationships among images and texts are extracted and built in the general form of a graph, effective navigation through such a large graph representation becomes critically important. A good solution must allow collection overview and detail exploration. This demands a flexible graph layout that dynamically and smoothly displays relevant information content at various levels of detail. Second, visual guidance should be given so that users can easily explore the collection with meaningful directions. Besides interactive filtering, the capability to compare nodes of interest for deep comprehension is necessary. Third, automatic recommendation that provides the suggestions for further exploration is also desirable. Such a capability allows users to browse through the graph in a progressive manner.

iGraph consists of tens of thousands of nodes and hundreds of millions of edges. To enable effective exploration, it incorporates progressive graph drawing in conjunction with animated transition and interactive filtering. Node comparison is enabled by visually arranging selected nodes and their most related ones for detailed analysis. iGraph also provides various means for image and keyword input so that users can conveniently select nodes of interest for purposeful comparisons. To provide effective guidance, automatic visual recommendation is realized by providing the suggestions for future exploration based on the analysis of image popularity, text frequency, and

user exploration history.

Since iGraph on the single machine is not the focus of this Chapter, we skip the technique part of this work and recommend readers to read the paper [31] for details.

9.3 iGraph on Distributed System

iGraph is designed for a single machine and the user interface is suitable to display on a desktop monitor. To make it capable for dealing with big data and providing high-resolution rendering, we extend this work by using a large tiled display at Michigan Technological University's Immersive Visualization Studio (*IVS*).

9.4 IVS Cluster and Tiled Display Wall

IVS cluster is a distributed system built with Rocks Cluster Distribution 5.4.2 (with CentOS 5.5) at Michigan Technological University. There are eight tile nodes with four CPU cores, 32 GB RAM, and two NVIDIA GeForce GTX 680 GPUs in the cluster for computation and visualization. The cluster also provides a tiled display wall which consists of 6×4 thin-bezel 46-inch Samsung monitors, each with 1920×1080 pixels. These 24 monitors are driven by the eight tile nodes and each node corresponds

to three monitors. In total, the tiled display can display nearly 50 million pixels simultaneously.

9.4.1 Chromium

As an initial attempt, we first leveraged the open-source libraries Chromium [1], a system for interactive rendering on clusters of workstations to forward iGraph from the local desktop monitor to the display wall. One key feature of Chromium is that it is transparent to the programmers and allows many OpenGL programs to run without modification. By specifying the hardware configuration into the Chromium config file, users can easily set up Chromium and fit it into their own tiled display architecture. However, Chromium could only support pure OpenGL program. This will not be the case when a OpenGL program has some GUI components designed by some third-party libraries, such as QT. Furthermore, the performance of Chromium is poor when a program has a lot of animation and transitions due to some network bandwidth issues. In order to overcome such problems, we built a distributed display framework which was initially designed by James Walk and Dr. Kuhl.

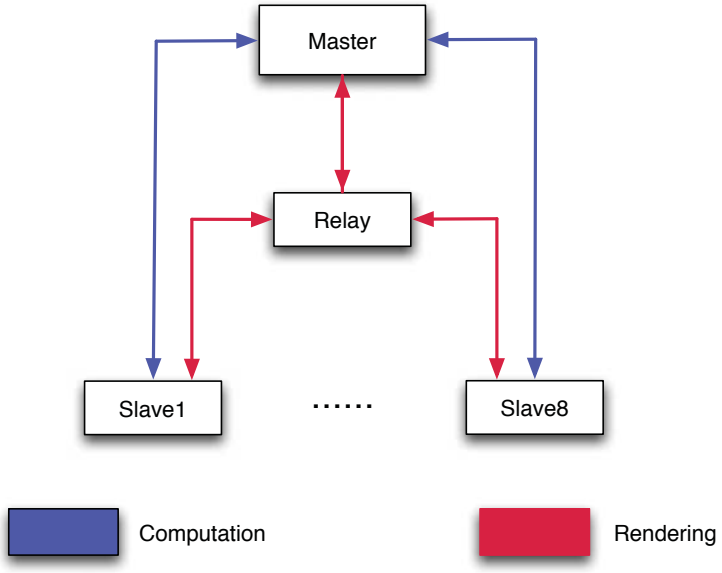


Figure 9.1: Computation and rendering workflows of the distributed system.

9.4.2 Distributed Display Framework

The iGraph on the distributed system has two major modules: computation module and rendering module. The computation module consists of a master node and eight slave computation nodes. The master program not only has a user interface to accept all the user interactions but also displays iGraph. It runs on a local computer which is located in the same room as the display wall. This computer captures user interactions, sends instructions to the eight slave computation nodes for parallel computing. Specifically, each computation node uses a CUDA based program to process extensive computations in parallel locally and the eight computation nodes work simultaneously to achieve parallel computing globally. The communication between

nodes are achieved by using Message Passing Interface (*MPI*). An uniform work assignment strategy is used to balance work load among nodes. When the computation nodes complete the computation, the master node gathers all the results and then executes the CUDA-based graph layout program to generate final layout of iGraph. To render the iGraph on the display wall, the master then sends the rendering data to the rendering module.

For the rendering module, an OpenGL program will be installed and running on all eight slave rendering nodes simultaneously with different viewport specifications. Basically, we partition the whole iGraph layout into eight blocks based on the tiled display configuration and each rendering node is responsible for one block. A relay node is used to communicate between the master node and all rendering nodes. It receives the rendering data from the master and then broadcasts it to all the rendering nodes. The UDP packages are used for fast data transmission. However, since the size of one UDP package is too small to hold all the data, there will be hundreds and even thousands of packages for each transmission. To guarantee the correctness, we order each package before sending and let the rendering nodes sort them after receiving. Furthermore, the rendering nodes will send a flag to the relay after each receive and the relay node will send next package right after it gets all the flags from the rendering nodes. After the rendering nodes receive the data from the relay node, they decode the data and render the visualization results to the block that it is responsible for. Since the time gaps between neighboring receiving of a rendering node may be less

than that of decoding so that the data may change undesirably during decoding, we create a large buffer, use one thread to receive the data and save it to the buffer so that the new coming data will not override the previous data. This also prevents receiving from interrupting decoding and rendering.

Figure 9.1 shows the workflows for the computation and rendering modules of the distributed system. The blue and red arrows indicate the data transmission for the computation and rendering modules, respectively.

9.5 Results

We experiment with two well-known collections: the APOD collection and the MIR Flickr collection. The Astronomy Picture of the Day (APOD) [65] is an online astronomy image collection maintained by NASA and Michigan Technological University. Everyday APOD features a picture of our universe, along with a brief explanation written by a professional astronomer. Since its debut in June 1995, APOD has archived thousands of handpicked pictures, which makes it the largest collection of annotated astronomy images on the Internet. The MIR Flickr collection [37] is offered by the LIACS Media lab at Leiden University. The collection was introduced by the ACM MIR Committee in 2008 as an ACM sponsored image retrieval evaluation. We use the MIRFLICKR-25000 collection which consists of 25,000 annotated images

downloaded from the social photography site Flickr through its public API.

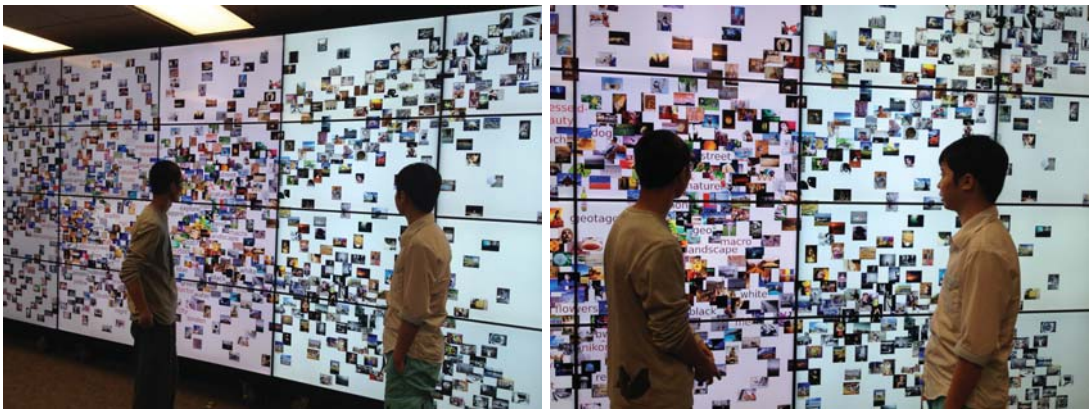


Figure 9.2: Photos showing iGraph of the MIR Flickr data set using the tiled display wall.

Figure 9.2 shows three iGraph photos of the MIR Flickr data set rendered on the display wall (1000 images and 50 keywords are displayed). The figure is from the original publication [31]. With the display wall, we are able to display thousands of images and keywords simultaneously for comfortable viewing. Currently, we are using this display wall for showing iGraph demos to visitors, including university alumni, visitors, and summer students. Initial feedback from several groups of visitor is fairly positive as they comment that running iGraph on this life-size tiled display is much more expressive and fun to watch compared with on a regular desktop display. The advantage of using the display wall is that it allows more than a dozen of people to comfortably view and discuss the results together in such a collaborative environment. Nevertheless, with the dramatic expanding of display area, it takes more effort for a viewer to correlate and compare images that are on the opposite sides of the display wall, especially for those images close to the wall's boundary.

References

[1] Chromium. http://chromium.sourceforge.net/.

[2] OpenMesh. http://www.openmesh.org/.

[3] Tutorial: Introduction to 2D Flow Field Visualization. http://www.cs.mtu.edu/~chaoliw/2dflowvis.html/.

[4] T. Arbel and F. P. Ferrie. Viewpoint Selection by Navigation through Entropy Maps. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 248–254. IEEE, 1999.

[5] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for Drawing Graphs: An Annotated Bibliography. *Computational Geometry*, 4(5):235–282, 1994.

[6] B. Beckman. Theory of Spectral Graph Layout. *vertex*, 1:2, 1994.

[7] U. D. Bordoloi and H.-W. Shen. View Selection for Volume Rendering. In *Proceedings of IEEE Visualization Conference*, pages 487–494. IEEE, 2005.

[8] A. Brambilla, R. Carnecky, R. Peikert, I. Viola, and H. Hauser. Illustrative Flow Visualization: State of the Art, Trends and Challenges. In *Eurographics State-of-the-Art Reports*, pages 75–94, 2012.

[9] L. Brillouin. *Science and Information Theory*. Courier Dover Publications, 2004.

[10] L. Buatois, G. Caumon, and B. Lévy. Concurrent Number Cruncher - A GPU Implementation of a General Sparse Linear Solver. *International Journal of Parallel, Emergent and Distributed Systems*, 24(3):205–223, 2009.

[11] B. Cabral and C. Leedom. Imaging Vector Fields Using Line Integral Convolution. In *Proceedings of ACM SIGGRAPH Conference*, pages 263–270. ACM, 1993.

[12] M. Campbell, S. Julious, and D. Altman. Estimating Sample Sizes For Binary, Ordered Categorical, and Continuous Outcomes in Two Group Comparisons. *BMJ: British Medical Journal*, pages 1145–1148, 1995.

[13] C.-M. Chen, L. Xu, T.-Y. Lee, and H.-W. Shen. A Flow-Guided File Layout for out-of-Core Streamline Computation. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 145–152, organization=IEEE, 2012.

[14] M. Chen and H. Jänicke. An Information-Theoretic Framework for Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1206–1215, 2010.

[15] Y. Chen, J. D. Cohen, and J. H. Krolik. Similarity-Guided Streamline Placement with Error Evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1448–1455, 2007.

[16] C. D. Correa, D. Silver, and M. Chen. Illustrative Deformation for Data Exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1320–1327, 2007.

[17] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-interscience, 2012.

[18] W. Cui, Y. Wu, S. Liu, F. Wei, M. X. Zhou, and H. Qu. Context Preserving Dynamic Word Cloud Visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 121–128, 2010.

[19] J. Daemen and V. Rijmen. AES Proposal: Rijndael. 1998.

[20] J. De Winter. Using the Student's t-test with extremely small sample sizes. *Practical Assessment, Research & Evaluation*, 18(10):2, 2013.

[21] U. Doğrusöz, B. Madden, and P. Madden. Circular Layout in the Graph Layout Toolkit. In *Graph Drawing*, pages 92–100. Springer, 1997.

[22] M. Feixas, M. Sbert, and F. González. A Unified Information-Theoretic Framework for Viewpoint Selection and Mesh Saliency. *ACM Transactions on Applied Perception*, 6(1), 2009.

[23] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic Camera Placement for Image-Based Modeling. In *Computer Graphics Forum*, volume 19, pages 101–110. Wiley Online Library, 2000.

[24] C. O. Fritz, P. E. Morris, and J. J. Richler. Effect Size Estimates: Current Use, Calculations, and Interpretation. *Journal of Experimental Psychology: General*, 141(1):2, 2012.

[25] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-Directed Placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[26] A. L. Fuhrmann and M. E. Gröller. Real-Time Techniques for 3D Flow Visualization. In *Proceedings of IEEE Visualization Conference*, pages 305–312. IEEE, 1998.

[27] S. Furuya and T. Itoh. A Streamline Selection Technique for Integrated Scalar and Vector Visualization. In *IEEE Visualization Conference Poster Compendium*. IEEE, 2008.

[28] N. Gagvani and D. Silver. Parameter-Controlled Volume Thinning. *Graphical Models and Image Processing*, 61(3):149–164, 1999.

[29] R. Gasteiger, M. Neugebauer, O. Beuing, and B. Preim. The FLOWLENS: A Focus-and-Context Visualization Approach for Exploration of Blood Flow in Cerebral Aneurysms. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2183–2192, 2011.

[30] Y. Gu and C. Wang. Transgraph: Hierarchical Exploration of Transition Relationships in Time-Varying Volumetric Data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2015–2024, 2011.

[31] Y. Gu, C. Wang, J. Ma, J. R. Nemiroff, and L. D. Kao. iGraph: A Graph-Based Technique for Visual Analytics of Image and Text Collections. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*. International Society for Optics and Photonics, 2015.

[32] L.-W. He, M. F. Cohen, and D. H. Salesin. The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing. In *Proceedings of ACM SIGGRAPH Conference*, pages 217–224. ACM, 1996.

[33] B. Heckel, G. H. Weber, B. Hamann, and K. I. Joy. Construction of Vector Field Hierarchies. In *Proceedings of IEEE Visualization Conference*, pages 19–25. IEEE, 1999.

[34] E. Heiberg, T. Ebbers, L. Wigström, and M. Karlsson. Three-Dimensional Flow Characterization Using Vector Pattern Matching. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):313–319, 2003.

[35] I. Herman, G. Melançon, and M. S. Marshall. Graph Visualization and Navigation in Information Visualization: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):24–43, 2000.

[36] H. Hoppe. Progressive Meshes. In *Proceedings of ACM SIGGRAPH Conference*, pages 99–108, 1996.

[37] M. J. Huiskes and M. S. Lew. The MIR Flickr Retrieval Evaluation. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 39–43. ACM, 2008.

[38] H. Jänicke, T. Weidner, D. Chung, R. S. Laramee, P. Townsend, and M. Chen. Visual Reconstructability as a Quality Metric for Flow Visualization. *Computer Graphics Forum*, 30(3):781–790, 2011.

[39] S. Janušonis. Comparing Two Small Samples with An Unstable, Treatment-Independent Baseline. *Journal of neuroscience methods*, 179(2):173–178, 2009.

[40] G. Ji and H.-W. Shen. Dynamic View Selection for Time-Varying Volumes. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

[41] B. Jobard and W. Lefer. Creating Evenly-Spaced Streamlines of Arbitrary Density. *Visualization in Scientific Computing*, 97:43–56, 1997.

[42] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[43] S. Kullback and R. A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[44] R. S. Laramee, C. Garth, H. Doleisch, J. Schneider, H. Hauser, and H. Hagen. Visual Analysis and Exploration of Fluid Flow in a Cooling Jacket. In *Proceedings of IEEE Visualization Conference*, pages 623–630. IEEE, 2005.

[45] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. 23(2):203–222, 2004.

[46] T.-Y. Lee, O. Mishchenko, H.-W. Shen, and R. Crawfis. View Point Evaluation and Streamline Filtering for Flow Visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 83–90. IEEE, 2011.

[47] P. Leopardi. A Partition of the Unit Sphere into Regions of Equal Area and Small Diameter. *Electronic Transactions on Numerical Analysis*, 25(12):309–327, 2006.

[48] L. Li, H.-H. Hsieh, and H.-W. Shen. Illustrative Streamline Placement and Visualization. In *Proceedings of IEEE VGTC Pacific Visualization Symposium*, pages 79–86. IEEE, 2008.

[49] L. Li and H.-W. Shen. Image-Based Streamline Generation and Rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 13(3):630–640, 2007.

[50] A. Light and P. J. Bartlein. The End of the Rainbow? Color Schemes for Improved Data Graphics. *EOS Transactions of the American Geophysical Union*, 85(40), 2004.

[51] Z. Liu and R. J. Moorhead. Interactive View-Driven Evenly Spaced Streamline Placement. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*. International Society for Optics and Photonics, 2006.

[52] Z. Liu, R. J. Moorhead, and J. Groner. An Advanced Evenly-Spaced Streamline Placement Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):965–972, 2006.

[53] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of ACM SIGGRAPH Conference*, pages 163–169, 1987.

[54] J. Ma, W. James, C. Wang, A. K. Scott, and C.-K. Shene. FlowTour: An Automatic Guide for Exploring Internal Flow Features. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 25–32. IEEE, 2014. to appear.

[55] J. Ma, J. Tao, M. Keranen, J. Mayo, C.-K. Shene, and C. Wang. SHAvisual: A Secure Hash Algorithm Visualization Tool (poster). In *Proceedings of Conference on Innovation & Technology in Computer Science Education*, pages 338–338. ACM, 2014.

[56] J. Ma, C. Wang, and C.-K. Shene. Coherent View-Dependent Streamline Selection for Importance-Driven Flow Visualization. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*. International Society for Optics and Photonics, 2013.

[57] J. Ma, C. Wang, and C.-K. Shene. FlowGraph: A Compound Hierarchical Graph for Flow Field Exploration. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 233–240. IEEE, 2013.

[58] J. Ma, C. Wang, C.-K. Shene, and J. Jiang. A Graph-Based Interface for Visual Analytics of 3D Streamlines and Pathlines. *IEEE Transactions on Visualization and Computer Graphics*.

[59] S. Marchesin, C.-K. Chen, C. Ho, and K.-L. Ma. View-Dependent Streamlines for 3D Vector Fields. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1578–1586, 2010.

[60] O. Mattausch, T. Theußl, H. Hauser, and M. E. Gröller. Strategies for Interactive Exploration of 3D Flow Using Evenly-Spaced Illuminated Streamlines. In *Proceedings of Spring Conference on Computer graphics*, pages 213–222. ACM, 2003.

[61] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over Two Decades of Integration-Based, Geometric Flow Visualization. *Computer Graphics Forum*, 29(6):1807–1829, 2010.

[62] A. Mebarki, P. Alliez, and O. Devillers. Farthest Point Seeding for Efficient Placement of Streamlines. In *Proceedings of IEEE Visualization Conference*, pages 479–486. IEEE, 2005.

[63] V. S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in CryptologyCRYPTO85 Proceedings*, pages 417–426. Springer, 1986.

[64] B. Moberts, A. Vilanova, and J. J. van Wijk. Evaluation of Fiber Clustering Methods for Diffusion Tensor Imaging. In *Proceedings of IEEE Visualization Conference*, pages 65–72. IEEE, 2005.

[65] R. J. Nemiroff and J. T. Bonnell. Astronomy Picture of The Day: http://antwrp.gsfc.nasa.gov/apod/astropix.html. In *Bulletin of the American Astronomical Society*, volume 27, page 1291, 1995.

[66] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-Balanced Parallel Streamline Generation on Large Scale Vector Fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1785–1794, 2011.

[67] H. C. Purchase, N. Andrienko, T. Jankun-Kelly, and M. Ward. Theoretical foundations of information visualization. In *Information Visualization*, pages 46–64. Springer, 2008.

[68] O. Rosanwo, C. Petz, S. Prohaska, H.-C. Hege, and I. Hotz. Dual Streamline Seeding. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 9–16. IEEE, 2009.

[69] C. Rössl and H. Theisel. Streamline Embedding for 3D Vector Field Exploration. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):407–420, 2012.

[70] M. Ruiz, I. Boada, M. Feixas, and M. Sbert. Viewpoint Information Channel for Illustrative Volume Rendering. *Computers & Graphics*, 34(4):351–360, 2010.

[71] M. Schlemmer, I. Hotz, B. Hamann, F. Morr, and H. Hagen. Priority Streamlines: A Context-Based Visualization of Flow Fields. In *Proceedings of Eurographics/IEEE VGTC Symposium on Visualization*, pages 227–234. Eurographics Association, 2007.

[72] D. Schroeder, D. Coffey, and D. F. Keefe. Drawing with the Flow: A Sketch-Based Interface for Illustrative Visualization of 2D Vector Fields. In *Proceedings of ACM SIGGRAPH/Eurographics Sketch-Based Interfaces and Modeling*, pages 49–56. ACM, 2010.

[73] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Proceedings of ACM Workshop on Applied Computational Geometry*, pages 203–222, 1996.

[74] D. Silver and X. Wang. Tracking and Visualizing Turbulent 3D Features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):129–141, 1997.

[75] D. Silver and X. Wang. Tracking Scalar Features in Unstructured Data Sets. In *Proceedings of IEEE Visualization Conference*, pages 79–86, 1998.

[76] B. Spencer, R. S. Laramee, G. Chen, and E. Zhang. Evenly Spaced Streamlines for Surfaces: An Image-Based Approach. *Computer Graphics Forum*, 28(6):1618–1631, 2009.

[77] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. A Feature-Driven Approach to Locating Optimal Viewpoints for Volume Visualization. In *Proceedings of IEEE Visualization Conference*, pages 495–502. IEEE, 2005.

[78] J. Tao, J. Ma, M. Keranen, J. Mayo, and C.-K. Shene. DESvisual: A Visualization Tool for the DES Cipher. *Journal of Computing Sciences in Colleges*, 27(1):81–89, 2011.

[79] J. Tao, J. Ma, M. Keranen, J. Mayo, and C.-K. Shene. ECvisual: A Visualization Tool for Elliptic Curve Based Ciphers. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 571–576. ACM, 2012.

[80] J. Tao, J. Ma, M. Keranen, J. Mayo, C.-K. Shene, and C. Wang. RSAvisual: A Visualization Tool for the RSA Cipher. In *Proceedings of ACM Technical Symposium on Computer Science Education*, 2014. (to appear).

[81] J. Tao, J. Ma, C. Wang, and C.-K. Shene. A Unified Approach to Streamline Selection and Viewpoint Selection for 3D Flow Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):393–406, 2013.

[82] A. Telea and J. J. van Wijk. Simplified Representation of Vector Fields. In *Proceedings of IEEE Visualization Conference*, pages 35–42. IEEE, 1999.

[83] I. G. Tollis. Graph Drawing and Information Visualization. *ACM Computing Surveys (CSUR)*, 28(4es):19, 1996.

[84] W. Trappe and L. C. Washington. *Introduction to Cryptography with Coding Theory*. Pearson Education India, 2006.

[85] R. J. Trumpler and H. F. Weaver. *Statistical Astronomy*. Univ of California Press, 1953.

[86] G. Turk and D. Banks. Image-Guided Streamline Placement. In *Proceedings of ACM SIGGRAPH Conference*, pages 453–460. ACM, 1996.

[87] J. J. van Wijk. Spot Noise Texture Synthesis for Data Visualization. In *Proceedings of ACM SIGGRAPH Conference*, pages 309–318. ACM, 1991.

[88] J. J. van Wijk. Image Based Flow Visualization. In *Proceedings of ACM SIGGRAPH Conference*, pages 745–754. ACM, 2002.

[89] J. J. van Wijk and W. A. A. Nuij. Smooth and Efficient Zooming and Panning. In *Proceedings of IEEE Information Visualization Symposium*, pages 15–22. IEEE, 2003.

[90] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Viewpoint Selection using Viewpoint Entropy. In *Proceedings of Vision, Modeling, and Visualization Conference*, pages 273–280, 2001.

[91] V. Verma, D. Kao, and A. Pang. A Flow-Guided Streamline Seeding Strategy. In *Proceedings of IEEE Visualization Conference*, pages 163–170. IEEE Computer Society Press, 2000.

[92] I. Viola, M. Feixas, M. Sbert., and M. E. Gröller. Importance-Driven Focus of Attention. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):933–940, 2006.

[93] C. Wang and H.-W. Shen. Information Theory in Scientific Visualization. *Entropy*, 13(1):254–273, 2011.

[94] J. Wei, C. Wang, H. Yu, and K.-L. Ma. A Sketch-Based Interface for Classifying and Visualizing Vector Fields. In *Proceedings of IEEE VGTC Pacific Visualization Symposium*, pages 129–136. IEEE, 2010.

[95] L. Wong, C. Dumont, and M. Abidi. Next Best View System in a 3-D Modeling Task. In *Proceedings of International Symposium on Computational Intelligence in Robotics and Automation*, pages 306–311. IEEE, 1999.

[96] K. Wu, Z. Liu, S. Zhang, and R. J. Moorhead. Topology-Aware Evenly Spaced Streamline Placement. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):791–801, 2010.

[97] C. Xu and J. L. Prince. Gradient Vector Flow: A New External Force for Snakes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 66–71, 1997.

[98] L. Xu, T.-Y. Lee, and H.-W. Shen. An Information-Theoretic Framework for Flow Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216–1224, 2010.

[99] L. Xu and H.-W. Shen. Flow Web: A Graph Based User Interface for 3D Flow Field Exploration. In *SPIE Proceedings of Visualization and Data Analysis*. International Society for Optics and Photonics, 2010.

[100] X. Ye, D. Kao, and A. Pang. Strategy for Seeding 3D Streamlines. In *Proceedings of IEEE Visualization Conference*, pages 471–478. IEEE, 2005.

[101] H. Yu, C. Wang, and K.-L. Ma. Parallel Hierarchical Visualization of Large Time-Varying 3D Vector Fields. In *Proceedings of ACM/IEEE Supercomputing Conference*. IEEE, 2007.

[102] H. Yu, C. Wang, and K.-L. Ma. Parallel Hierarchical Visualization of Large Time-Varying 3D Vector Fields. In *Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on*, pages 1–12. IEEE, 2007.

[103] H. Yu, C. Wang, C.-K. Shene, and J. H. Chen. Hierarchical Streamline Bundles. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1353–1367, 2012.

Appendix A

Information Theory Background

In this Chapter, we give detailed explanations for several key concepts in information theory which are extensively used in our work. The original definitions are provided first followed by their application in our projects.

A.1 Terminology

A.1.1 Marginal, Joint, and Conditional Probabilities

The *marginal probability* refers to the occurrence probability of a single event for a given random variable which is irrelevant to other events. Marginal probability is

usually computed by summing values in a table along rows or columns, and writing the summation in the margins of the table [85].

Given a random variable A and all its n sample events, the marginal probability of a specific independent event x can be computed as m/n, where m is the times event x occurs.

A *joint probability* is the probability that two events will occur simultaneously. Given two independent events x and y, their joint probability $p(x, y)$ is computed as $p(x) \times p(y)$ where $p(x)$ and $p(y)$ are marginal probabilities for events x and y, respectively.

We use the *conditional probability* to compute the occurrence probability of an event on the premise that another event has already occurred, assuming x and y are independent to each other. Given two events x and y, the conditional probability of x given y is denoted as $p(x|y)$ and can be computed as follows:

$$p(x|y) = \frac{p(x, y)}{p(y)}, \tag{A.1}$$

where $p(y)$ is the marginal probability of y and $p(x, y)$ is the joint probability of x and y.

A.1.2 Shannon Entropy

The concept of *entropy* was first introduced by Shannon in 1948 as a quantitive measurement for the expected information value contained in a message or the uncertainty of a random variable represented by a distribution. Entropy is typically measured in bits, nats, or bans [9].

Given a discrete random variable X with alphabet $\mathcal{X}$ and its marginal probability $p(x)$, we define its entropy as following:

$$H(X) = -\sum_{x} p(x) \log p(x), \tag{A.2}$$

where $p(x) \in [0, 1]$ and $\sum p(x) = 1.0$. The logarithm is taken in base 2 or e. The zero probability contributes nothing to the entropy as we define $0 \log 0 = 0$. Since entropy indicates the number of bits required to measure the uncertainty of the variable X, its value will never be negative, which could also be verified from Equation A.2. The higher the entropy is, the more information the variable contains. One important property of the entropy is that $H(X)$ is a concave function and reaches its maximum of $\log |X|$ if and only if $p(x)$ is equal for all x, i.e., when the probability distribution is uniform. So in most visualization applications including ours, the notion of "equal probability, maximum entropy" [93] is at the heart of probability function design.

A.1.3 Mutual Information

In information theory, the *mutual information* of two random variables is used to quantify the mutual dependence of the two variables. In other words, it measures the amount of information shared by two variables. Given two discrete random variable X and Y, we define the mutual information between them as:

$$I(X;Y) = \sum_{x}\sum_{y} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}, \tag{A.3}$$

where $p(x)$ and $p(y)$ are marginal probabilities of variable X and Y, and $p(x,y)$ is their joint probability. Mutual information is the reduction in the uncertainty of one random variable due to the knowledge of the other [17]. If X and Y are independent to each other, then $p(x,y) = p(x)p(y)$, which means knowing X does not provide any information about knowing Y and vice versa. Therefore, $I(X;Y) = 0$. On the other hand, if X and Y are exactly identical, then all information from one variable will also be shared by the other. In this case, $I(X;Y)$ is equal to the information contained in X or Y alone, which is the $H(X)$ or $H(Y)$.

A.1.4 Information Channel

In data communication, transmitting a message X from the source point, the sender, through a noisy communication channel to the destination, the receiver, is a major task. However, due to the noisy nature of the channel, information loss is inevitable and the final received message X' will be different from the original message X. Therefore, one obvious goal for data communication is to quantify the uncertainty noise embedded in the transmitted message so that the signal interference could be eliminated as much as possible in the noisy channel [93].

Borrowing the idea from data communication, we treat the visualization process as a special communication channel, which is called *information channel*. This channel conveys the information in the source data, e.g., a 2D image or a 3D iso-surface, to the destination, the viewer. As described in [93], the source data in a visualization pipeline need to be transformed by a sequence of steps such as denoising, filtering, visual mapping, and projection. Each of the transformation steps can be thought as an encoding process whose goal is to preserve the maximum amount of information from the source data and generate output for the next stage. However, information loss is usually inevitable during the transformation, e.g. projecting 3D objects into a 2D image. Therefore, most visualization applications pay attention to reduce the distortion and preserve as much information of the original data as possible.

A.2 Application in our work

In this section, we explain how information theory is applied to our work by introducing the way we define the streamline importance based on entropy and mutual information. Furthermore, the entropy field computation for a corresponding flow field is also described. Finally, we also introduce how we leverage information channel to build two dual channels between a set of streamlines and a set of viewpoints in order to solve streamline selection and viewpoint selection in a unified framework.

A.2.1 Streamline Entropy

We evaluate the streamline importance based on its entropy value. For each streamline, we employ a sliding window technique along each point of the streamline and evaluate its entropy within the local window region. To better evaluate the entropy, we assume that each streamline has been reparameterized by the arc length and we use newly created sample points along the reparameterized streamline.

Based on Equation A.2, we need to compute $p(x)$ for every point on the streamline to obtain the entropy value. To achieve this, we interpolate the point's vector from the original vector field and evaluate the vector variation within the sliding window

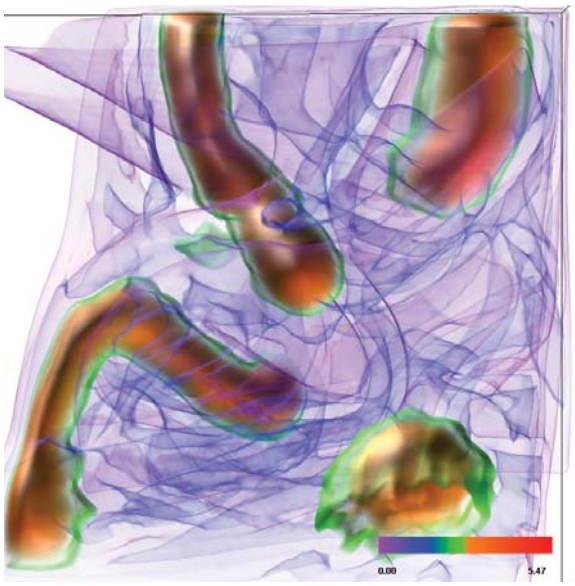


Figure A.1: The entropy field of the five critical points data set (© 2014 IEEE).

centered at the point. Specifically, the sliding window covers several consecutive sample points on the streamline (e.g. 5 or 7) and the current computed point is at the center of the window. We consider both the *direction* and the *magnitude* of the vectors. For vector direction, we decompose a unit sphere into a certain number of patches of equal area with small diameter following the algorithm proposed by Leopardi [47]. All vectors falling into the same patch will be quantized into the same bin of vector direction. For vector magnitude, we quantize it into a certain number of levels and each level corresponds to a counting bin. A 2D histogram consisting of vector direction and magnitude is created for each sliding window. $p(x)$ is computed as the normalized bin count of the 2D histogram.

A.2.2 Entropy Field Computation

Given an input 3D flow field, we first compute its corresponding scalar entropy field. We employ a $9 \times 9 \times 9$ cube centered at each voxel and evaluate its entropy within this local region. The entropy of a discrete random variable X with alphabet $\mathcal{X}$ fulfills the following equation: $H(X) = -\sum_x p(x) \log p(x)$, where $p(x) \in [0, 1]$ is the marginal probability of x. In our case, we compute $p(x)$ by evaluating the variation of both *direction* and *magnitude* of vectors for all voxels in the cube and creating a 2D histogram consisting of these two components. In terms of vector direction, we decompose a unit sphere into a certain number of patches with equal area following the work of Leopardi [47]. Each sphere patch indicates one bin of vector direction. By locating the sphere patch each vector falls into, we quantize all vectors into different bins of vector direction. For vector magnitude, we first define several magnitude levels and then quantize vector magnitudes accordingly. The final $p(x)$ is obtained by normalizing each bin count of the 2D histogram. By applying this process to each voxel in the flow field, we generate an entropy field for the original flow field. Figure A.1 gives one such example. Colors are mapped to different entropy values as shown. The figure is from the work FlowTour [54] (). We implement entropy computation in the GPU using CUDA. For a data set which cannot be loaded into graphics memory once, we divide it into blocks and compute the entropy field in an out-of-core manner.

A.2.3 Streamline Mutual Information

We utilize the mutual information $I(X;Y)$ between the 3D streamline X and its 2D projection Y as one factor to quantify the view-dependent streamline importance. If $I(X;Y)$ is high, then the 3D streamline has a high entropy and its 2D projection preserves much of the 3D information. Conversely, if the 3D streamline has a low entropy, or its 2D projection loses much of the 3D information (even though the 3D streamline has a high entropy), then $I(X;Y)$ is low. Therefore, we favor streamlines that have high information content while their 2D projections reveal their characteristics well.

To compute the marginal probability $p(x)$ in Equation A.3, we use a similar solution presented in entropy evaluation and consider both vector direction and magnitude for the points along each streamline. The only difference is that we do not use the sliding window here and $p(x)$ is taken over the entire streamline. To compute the marginal probability $p(y)$, we use the projections of all vectors along all points of the streamline. To quantize projected 2D vector directions, we evenly partition a unit circle into a certain number of angle ranges. All vectors falling into the same range will be quantized into the same bin of vector direction. For projected vector magnitude, we quantize it into a certain number of levels as well. To compute the joint probability $p(x,y)$, we create a 2D joint histogram where the two axes are for all vector direction and magnitude combinations for variables X (3D streamline) and Y

(streamline 2D projection), respectively. In the joint histogram, the normalized bin count corresponds to $p(x, y)$.

A.2.4 Shape Characteristics

Shape characteristics is one critical term for computing the *streamline conditional probability* which will be introduced in the next subsection. It indicates how stereoscopic the shape of streamline s is reflected under viewpoint v. Since the number of points along each streamline could be fairly large (e.g., in the order of hundreds or thousands of points), we opt to approximate a streamline using its *skeleton* for fast shape characteristics analysis. The "skeleton" of a streamline is obtained using a uniform subsampling scheme along the integration points of the streamline to reduce the number of points to a smaller scale (e.g., in the order of tens of points). Let us denote the skeleton of streamline s as $\tilde{s} = \{\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_k\}$, the viewing vector as $\overrightarrow{v}$, and the angle between $\overrightarrow{v}$ and $\overrightarrow{\tilde{p}_i\tilde{p}_{i+1}}$ as θ. We define the shape characteristics of $\overrightarrow{\tilde{p}_i\tilde{p}_{i+1}}$ as

$$\alpha_{\tilde{p}_i\tilde{p}_{i+1};v} = \alpha_{\min} + (1.0 - \alpha_{\min})\Big(1.0 - \frac{|\pi/4 - \theta'|}{\pi/4}\Big), \tag{A.4}$$

where $\alpha_{\min}$ is the minimum value for the shape characteristics (we set $\alpha_{\min} = 0.1$ in our work) and

$$\theta' = \begin{cases} \pi - \theta, & \theta > \pi/2 \\ \theta, & \theta \leq \pi/2 \end{cases} \tag{A.5}$$

The intuition is that $\alpha_{\tilde{p}_i\tilde{p}_{i+1};v}$ gets its maximum (minimum) value of 1.0 ($\alpha_{\min}$) when $\overrightarrow{v}$ and $\overrightarrow{\tilde{p}_i\tilde{p}_{i+1}}$ form a 45° or 135° (0°, 90°, or 180°) angle. The shape characteristics of streamline skeleton $\tilde{s}$ is defined as

$$\alpha_{\tilde{s};v} = \frac{\sum_{i=1}^{k-1} \alpha_{\tilde{p}_i\tilde{p}_{i+1};v} \|\tilde{p}_i\tilde{p}_{i+1}\|}{\sum_{i=1}^{k-1} \|\tilde{p}_i\tilde{p}_{i+1}\|}. \tag{A.6}$$

A.2.5 Streamline Conditional Probability

With mutual information and shape characteristics defined for streamline s under viewpoint v, we define conditional probability $p(s|v)$ as

$$p(s|v) = \frac{\alpha_{\tilde{s};v} I(s; s_v)}{\sum_{s \in S} \alpha_{\tilde{s};v} I(s; s_v)}. \tag{A.7}$$

With $p(s|v)$ defined, besides simply assuming $p(v) = 1/m$, we can also obtain $p(v)$ from the normalization of the summation of all streamlines' conditional probabilities under v over all viewpoints V. That is, $p(v) = p(S|v)/p(S|V)$, where $p(S|v) = \sum_{s \in S} p(s|v)$ and $p(S|V) = \sum_{v \in V} p(S|v)$. We use this nonuniform specification of $p(v)$ in our work. Figure A.2 summarizes the order of computing the probabilities for the two channels.

Figure A.3 shows a comparison of selecting the best viewpoint based on $p(v)$ with considering mutual information only (left), shape characteristics only (middle), and

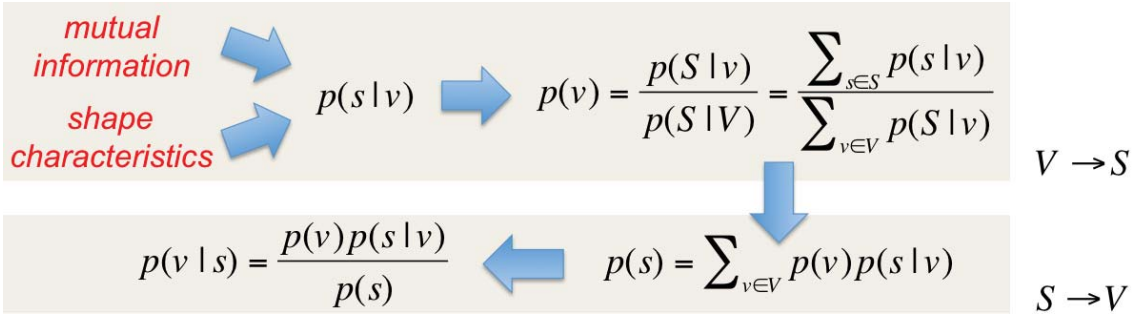


Figure A.2: The order of computing the probabilities for the two channels (© 2013 IEEE).

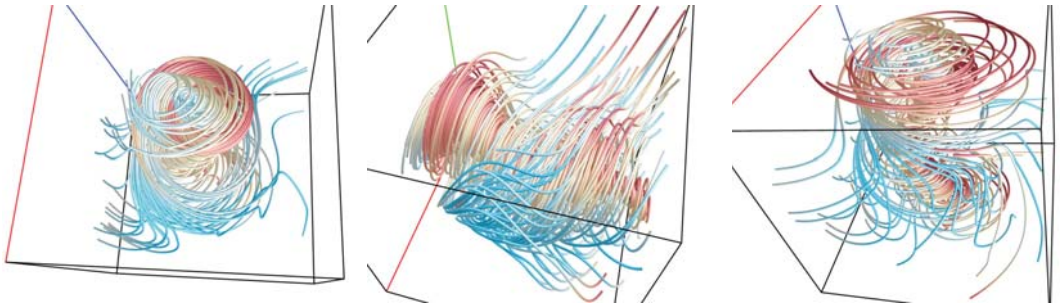


Figure A.3: The best viewpoint selection results (© 2013 IEEE).

both (right). When only mutual information is considered, the main axis of the tornado is almost parallel to the viewing vector, making $\overrightarrow{v}$ and $\overrightarrow{\tilde{p}_i\tilde{p}_{i+1}}$ form an almost 0° or 180° angle. $p(s|v)$ achieves its maximum for almost every streamline, letting $p(v)$ get its highest value. When only shape characteristics is considered, $\overrightarrow{v}$ and $\overrightarrow{\tilde{p}_i\tilde{p}_{i+1}}$ now form an almost 45° or 135° angle. The best viewpoint selected is still not desirable. When considering both mutual information and shape characteristics into $p(v)$ evaluation, we can select the more desirable best viewpoint as the overall structure of the tornado is best perceived.

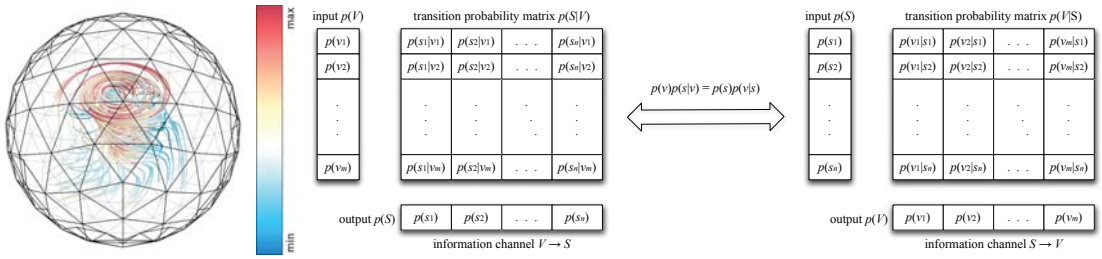
A.2.6 Information Channel Between A Streamline Set and A Viewpoint Set

We solve the problems of streamline selection and viewpoint selection in a single, unified framework. We consider a set of streamlines $S = \{s_1, s_2, \ldots, s_n\}$ and a set of viewpoints $V = \{v_1, v_2, \ldots, v_m\}$ as discrete random variables and build two interrelated information channels between them: $V \rightarrow S$ and $S \rightarrow V$. Our assumptions for viewpoints are (1) the flow field is centered in a sphere of sample viewpoints constructed from the recursive discretization of an icosahedron; and (2) the camera at a sample viewpoint is looking at the center of the sphere. Figure A.4 (a) shows sample viewpoints along the sphere. We use modified spectral colors [50] for streamline coloring based on the velocity magnitude.

The main components in the information channel $V \rightarrow S$ are the following:

- The *transition probability matrix* $p(S|V)$ where conditional probability $p(s|v)$ represents the probability of "seeing" streamline s from viewpoint v (i.e., the importance of s with respect to v).

- The *input probability distribution* $p(V)$ where $p(v)$ represents the probability of selecting viewpoint v. If we assume $p(v)$ to be evenly distributed, then $p(v) = 1/m$ where m is the number of sample viewpoints.



(a) (b)

Figure A.4: (a) Sample viewpoints constructed along a view sphere (b) The information channel $V \rightarrow S$ (left) and the inverted channel $S \rightarrow V$ (right) (© 2014 IEEE).

- The *output probability distribution* $p(S)$ where $p(s)$ represents the average probability that streamline s is seen from all viewpoints V. That is, $p(s) = \sum_{v \in V} p(v)p(s|v)$.

Similarly, we can construct the inverted information channel $S \rightarrow V$, where the input and output probability distributions are swapped: $p(S)$ becomes the input and $p(V)$ becomes the output. In this inverted channel, the new transition probability matrix is $p(V|S)$, where $p(v|s)$ represents the probability of selecting viewpoint v given streamline s. As shown in Figure A.4 (b), these two channels are connected via the Bayes theorem, i.e., $p(v)p(s|v) = p(v, s) = p(s, v) = p(s)p(v|s)$, which provides us a means to compute $p(v|s)$ given $p(v)$, $p(s)$, and $p(s|v)$. Figure A.2, A.3 and A.4 are from the work [81] (© 2013 IEEE) discussed in Chapter 3.