Interactive Visual Clutter Management in Scientific Visualization

Dissertation

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

By

Xin Tong, M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2016

Dissertation Committee:

Han-Wei Shen, Advisor

Huamin Wang

Arnab Nandi

Abstract

Scientists visualize their data and interact with them on computers in order to thoroughly understand them. Nowadays, data become so large and complex that it is impossible to display the entire data on a single image. Scientific visualization often suffers from visual clutter problem because of high spacial resolution/dimension and temporal resolution. Interacting with the visualizations of large data, on the other hand, allows users to dynamically explore different parts of the data and gradually understand all information in the data.

Information congestion and visual clutter exist in visualizations of different kinds of data, such as flow field data, tensor field data, and time-varying data. Occlusion presents a major challenge in visualizing 3D flow and tensor fields using streamlines. Displaying too many streamlines creates a dense visualization filled with occluded structures, but displaying too few streams risks losing important features. Glyph as a powerful multivariate visualization technique is used to visualize data through its visual channels. Placing large number of glyphs over the entire 3D space results in occlusion and visual clutter that make the visualization ineffective. To avoid the occlusion in streamline and glyph visualization, we propose a view-dependent interactive 3D lens that removes the occluding streamlines/glyphs by pulling the them aside through animations. High resolution simulations are capable of generating very large vector fields that are expensive to store and analyze. In addition, the noise and/or uncertainty contained in the data often affects the quality of visualization by producing visual clutter that interferes with both the interpretation and identification of important features. Instead, we can store the distributions of many vector orientations and visualize the distributions with 3D glyphs, which largely reduce visual clutter. Empowered by rapid advance of high performance computer architectures and software, it is now possible for scientists to perform high temporal resolution simulations with unprecedented accuracy. The large number of time steps makes it difficult to perform post analysis and visualization after the computation is completed. Instead of visualize all the time steps, users filter the original data that is too large to be all visualized and interactively pick the interesting parts of the data to display. To achieve this goal, we provide a time-varying data exploration system that allows users to pick the most salient time steps with Dynamic Time Warping (DTW) algorithm and then only visualize the data volumes corresponding to those time steps. We generalize three general strategies to manage visual clutter and demonstrate them using four visualization techniques. In the end of this dissertation, we present possible directions of future works that may inspire the readers to do more researches on visual clutter management for different data and applications.

This is dedicated to my parents Huijuan Yuan and Xiang Tong

Acknowledgments

Many thanks to my parents, Huijuan Yuan and Xiang Tong, for supporting my decision of studying abroad and allowing me to live a comfortable and worry-free life in the United States.

I would like to express my sincere gratitude to my adviser, Prof. Han-Wei Shen, for his patient, careful, and sometimes strict supervision in the past six years. He recognized me in his computer graphics class when I was pursuing the master degree and invited me to join his research team, which made me feel lucky. His talent and attitude towards research inspired and motivated me in every weekly meeting throughout the past five years. Besides, I really appreciate his recommendations for my past summer internships in three national laboratories, which I could never get by myself as a foreign national.

My four summer internships provided me great opportunities to learn from different people. First of all, I sincerely thank my mentor Pak Chung Wong in Pacific Northwest National Laboratory (PNNL) for spending huge amount of time teaching me everything from professional behavior to living a good life when I needed them the most during my first year in the US. His research suggestion on geometry manipulation directly led to my dissertation topic. Many thanks to my mentor Kenneth Moreland in Sandia National Laboratories (SNL) who taught me the mind of parallel computing. My geometry deformation technique wouldn't be so interactive without his inspiration. I greatly appreciate my mentor Patrick McCormick in Los Alamos National Laboratory (LANL) for opening my eyes to open source project develop and management. His instructions inspired me to make my interactive visualization system extensible, which notably boosted my research productivity.

I appreciate the patient works from my candidacy and dissertation committee members, Prof. Huamin Wang and Prof. Arnab Nandi, department representative, Prof. Rephael Wenger, and graduate faculty representative, Prof. Daniel R. Strunk. Prof. Wang offered very constructive suggestion about using physically-based elastic models during my PhD candidacy exam, which inspired my later publication on GlyphLens. During the past two years, Prof. Nandi provided me user interaction ideas and devices that became an important part of the dissertation.

Many thanks to my research collaborators, Prof. Christopher R. Johnson, Prof. John Edward, Teng-Yok Lee, Chris Jacobsen, Cheng Li, Prof. Huijie Zhang, and Chun-Ming Chen for their helps in discussing research ideas, programming, and paper editing. I am grateful for having enjoyed internships and countless lunches with my labmate Kewei Lu. I would like to thank my labmates that are not mentioned above, Xiaotong Liu, Tzu-Hsuan Wei, Ayan Biswas, Abon Chaudhuri, Steven Martin, Boonth Nouanesengsey, Wenbin He, Soumya Dutta, Ko-Chih Wang, Subhashis Hazarika, Junpeng Wang, Xiaoying Ge, Arindam Bhattacharya, David Maung, Zhili Chen, and Xiaofeng Wu, for their encouragements and supports.

I would like to thank my host family, Craig Gladwell and Mary Beth Gladwell, for giving me temporary housing when I just entered the US and helping me settle down in Columbus and practice English. I thank the most amazing roommates during my internships, Anna Shaverdian during PNNL internship in 2011 and Mehmet de Veci during SNL internship in 2013. They helped me fit into American culture and learn to make different kinds of friends. I truly appreciate the companies from my my greatest friends in school and internship cities: Junkun Wan, Dakai Hu, Qichao Que, Sheng Dong, Da Zhang, Linchuan Chen, Fengtao Fan, Yinxiao Chen, Yinxuan Shi, Robin Rupe, Jalyn Ide, Fabio Markus Miranda, Yucong (Chris) Ye, Roman Pyzh, Tianxing Xu, Nick Moss, Jing Lei, Yi Wang, Kai Wang, Qian Fan, Liwen Lin, Haixia Liu, Jin Peng, Ziqi Huang and many others. They made my six years' life in the US full of fun and emotion.

I would like to thank my grand-uncle in Michigan, Chunyang Yuan, and his families for inspiring me to study abroad when they visited China and encouraging me to work hard after I came to the US. I express my greatest thanks to my girlfriend, Wenxi Yang, who bore with my busy schedules of work and life for the most important past two years. She was always very patient to be the model and audio narrator for all the accompanying videos in my paper publications.

Vita

August 29, 1987	Born - Wuhan, China
2003 - 2006	Wuhan No. 2 High School Wuhan, China
2006 - 2010	Bachelor of Engineering Geoinformation Sci. & Tech. Tongji University, Shanghai, China
2010 - 2016	Master of Science Computer Science and Engineering The Ohio State University University
June - August 2011	Research Intern Pacific Northwest National Laboratory
June - August 2012	Research Intern Pacific Northwest National Laboratory
May - August 2013	Research Intern Sandia National Laboratories
May - August 2015	Research Intern Los Alamos National Laboratory
2011 - 2016	Doctoral studies Computer Science and Engineering The Ohio State University University

Publications

Research Publications

X. Tong, C. Li, and H.-W. Shen "GlyphLens: View-dependent Occlusion Management in the Interactive Glyph Visualization". *IEEE Scientific Visualization (SciVis) Conference*, Baltimore, Maryland, Oct. 2016 (accepted for publication).

X. Tong, H. Zhang, C. Jacobsen, H.-W. Shen, and P. McCormick "Crystal Glyph: Visualization of Directional Distributions Based on the Cube Map". *Eurographics Conference on Visualization*, Groningen, Netherlands, Jun. 2016 (short paper).

X. Tong, J. Edwards, C.-M. Chen, H.-W. Shen, C.R. Johnson, and P. C. Wong "View-Dependent Streamline Deformation and Exploration". *IEEE Transactions on Visualization and Computer Graphics*, 2016.

X. Tong, C.-M. Chen, H.-W. Shen, P. C. Wong "Interactive Streamline Exploration and Manipulation Using Deformation". *IEEE Pacific Visualization*, Hangzhou, China, April, 2015.

P. C. Wong, H.-W. Shen, L. Leung, S. Hagos, T.-Y. Lee, X. Tong, K. Lu "Visual Analytics of Large-Scale Climate Model Data". *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization*, Paris, France, Nov. 2014.

T.-Y. Lee, X. Tong, H.-W. Shen, P. C. Wong, S. Hagos, and L. Leung "Feature Tracking and Visualization of Madden-Julian Oscillation in Climate Simulation". *IEEE Computer Graphics and Applications*, 33(4): 29–37, 2013.

X. Tong, T.-Y. Lee, H.-W. Shen. "Salient Time Steps Selection from Large Scale Time-Varying Data Sets with Dynamic Time Warping". *IEEE symposium on Large Data Analysis and Visualization*, Seattle, Washington, Oct. 2012.

Fields of Study

Major Field: Computer Science and Engineering

Studies in:

Scientific Visualization
Human Computer Interaction
High Performance Computing

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

Scientific data are mostly generated from measurements or numerical simulations. As the accuracy of measuring instruments and the compute powers of high performance computers increase, the sizes of scientific data rapidly grow. The increases in the spatial and temporal resolutions of data allows the scientists to observe more details of data in the spatial and temporal domains. For example, scientists use numerical weather prediction simulation to predict weather. A high spatial/temporal resolution provides more accurate location/time information. More complex simulation models may produce more realistic results, but also produce a large number of variables as output, which increases the amount of information to be analyzed. In a weather simulation, for example, the variables can include precipitation, pressure, temperature, etc. In ensemble simulations, scientists run a simulation multiple times with different initial conditions and different models and produce results across all the simulation runs or ensemble members. The increasing number of ensemble members also greatly raise the amount of overall data information.

Scientific visualization first converts scientific data using two dimensional (2D) or three dimensional (3D) geometries, such as 2D bar charts and 3D isosurfaces, and

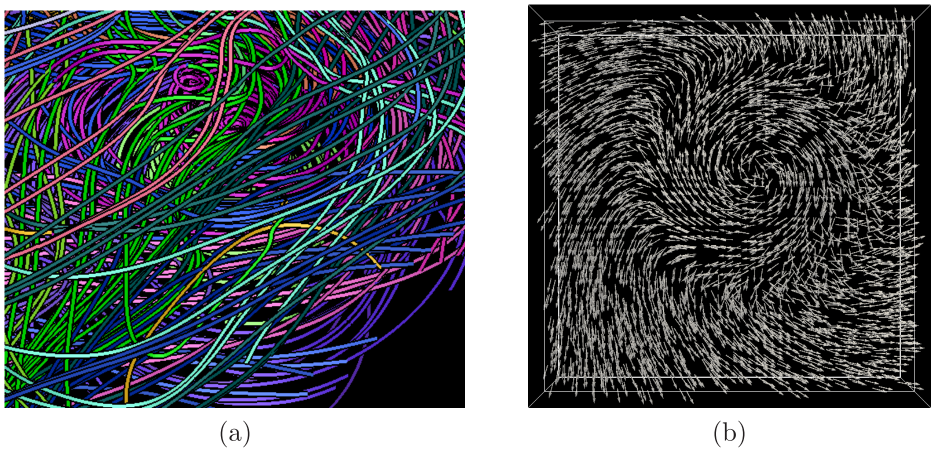


Figure 1.1: Examples of visual clutter in flow visualization. (a) In the streamline visualization of Hurricane Isabel flow field dataset, streamlines occlude each other; (b) Arrow plot visualization of a synthetic tornado dataset forms visual clutter.

then display the geometries on a computer screen. Visualization can more intuitively present information in data to users than simply presenting the numbers. Because of computer screen's limited resolution/size and human's limited visual perception ability, it is not realistic to display all the data on a screen at the same time. Besides, different users may have different interested aspects of the data, so a single image cannot fulfill the needs from different users. If information is overloaded or badly organized in a visualization, they could occlude each other and form visual clutter that distracts users from understanding the visualization.

For example, streamlines and arrow glyphs are well known flow visualization techniques, as shown in Figure 1.1. Both images present severe visual clutter. The streamlines in Figure 1.1a occlude each other, which prevents the viewers from easily tracing a particular streamline. The arrows in Figure 1.1b are densely distributed and overlap with each other, making it difficult to identify the pointing direction of a particular

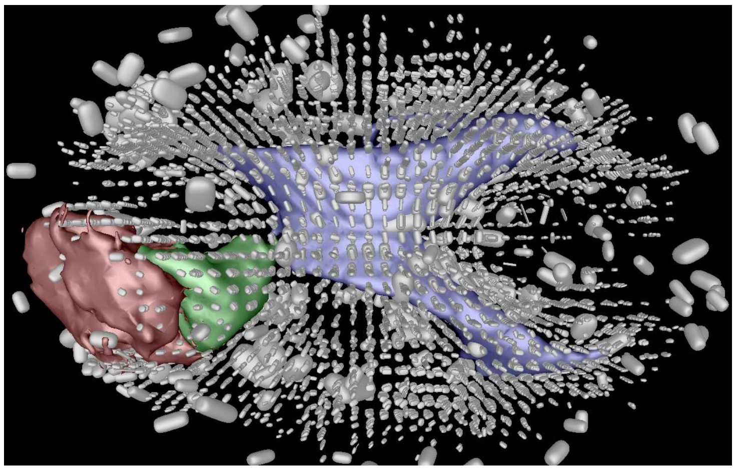


Figure 1.2: Superquadric glyphs visualize the tensor field in a DTI image of a tumor patient's brain. The dense glyphs create severe visual clutter and distraction.

arrow. In another example, superquadric glyphs are used to show the tensor field in diffusion tensor imaging (DTI) data. As shown in Figure 1.2, putting glyphs throughout the brain region generates visual clutter. The glyphs on the surface of the brain block the view of the glyphs inside the brain. Some glyphs in different depths may reside at the same image position, so their depth relationships are difficult to know.

1.2 General Strategies

In this dissertation, we want users to use the following three general strategies to reduce or remove visual clutters in scientific visualization.

Strategy 1: Directly manipulate geometries in visualization. The most intuitive way of removing visual clutter is directly taking the cluttered geometries away. We can allow users to pick their interested portion of data, such as a sub-region in the 3D spatial domain, and suppress the distraction from other parts of the data.

Instead of completely removing the occluding or distracting geometries, we smoothly transform their positions to the side to become contexts. In this way, both the user interested information and the occluding geometries are visible to the users. For example, in Figure 1.1a and Figure 1.2, users can interactively pull away the streamlines or superquadric glyphs on the surface of the volumes to see the inside geometries. In immersive environments, such as using virtual reality headsets, traditional computer input devices, such as mouse and keyboard, are difficult to use, because they are invisible in the view and their interaction space, the desktop, is different from the visualization space. A better interaction space is the 3D areas in front of users where their hands are captured by tracking systems with devices such as hand motion camera.

Strategy 2: Statistically summarize and visualize the data. If showing individual data item is not possible, the statistical summarization of the data, such as data's distribution, can also help users understand data. Distributions can be visualized in various forms, such as pie charts, box plots, histogram and etc. For example, the vector directions in Figure 1.1b can be summarized as histograms of 3D directions in individual local regions. Placing the visualization of the histogram in each local region visualizes the vector direction distribution in that region without introducing any visual clutter.

Strategy 3: Automatically suggest interesting parts of data to display. When data can be easily separated into multiple parts, we want the computer to automatically suggest the most representative parts to give an overview of the data. For example, when visualizing time-varying datasets, we can subdivide the screen into multiple regions and display the visualization of different time steps as small multiples in order to compare them. If visualizing all the time steps generates visual clutter, then only visualizing the data in computer suggested key time steps can make the information on the screen less cluttered.

1.3 Contributions

We summarize the contributions of this dissertation as follows:

Occlusion Management Using Deformation We propose a new streamline and glyph exploration approach by visually manipulating the cluttered streamlines or glyphs by pulling visible layers apart and revealing the hidden structures underneath. We present a customized view-dependent deformation algorithm to minimize visual clutter. The algorithm is able to maintain the overall integrity of the fields and expose previously hidden structures. We provide two space deformation models to displace the glyphs based on their spatial distributions. After the displacement, the streamlines or glyphs around the user-interested region are still visible as the context information, and their spatial structures are preserved.

Interactive Lens By using a lens metaphor of different shapes to select the transition zone of the targeted area for deformation interactively, the users can move their focus around and examine the vector or tensor data freely.

Visualization of Directional Distribution We present the cube map histogram, a new data structure for storing the distribution of three-dimensional vector directions. We also present the crystal glyph that effectively visualizes the directional distribution using OpenGL cube map textures. By placing crystal glyphs in the 3D data space, users can identify the directional distribution of the regional vector field from the shape and color of the glyph without visual clutter.

Salient Time Steps Selection We present a novel technique that can retrieve the most salient time steps, or key time steps, from large scale time-varying data sets. Visualizing only a small number of salient time steps reduces visual clutter. To achieve this goal, we develop a new time warping technique with an efficient dynamic programming scheme to map the whole sequence into an arbitrary number of time steps specified by the user. A novel contribution of our dynamic programming scheme is that the mapping between the whole time sequence and the key time steps is globally optimal, and hence the information loss is minimum. We propose a high performance algorithm to solve the dynamic programming problem that makes the selection of key times run in real time.

Interactive Visualization Systems We developed interactive visualization systems to explore glyph-based visualization and streamline visualization. In the systems, we provide a few lens utilities that allows users to pick a feature and look at it from different view directions. We compare different display/interaction techniques to visualize/manipulate our lens and glyphs. Our visualization system supports both mouse, direct-touch, and hand motion gesture interactions to manipulate the viewing perspectives and visualize the streamlines and glyphs in depth. We also create a visualization system that allows the user to browse time varying data at arbitrary levels of temporal detail. Because of the low computational complexity of this algorithm, the tool can help the user explore time varying data interactively and hierarchically.

Case Studies and Expert Feedback Case studies are provided for the proposed four techniques to demonstrate the utility for solving real world visualization problems with real scientific data. Domain experts, such as neurosurgeons, neurologists and researcher in material science, provided useful feedback on our streamline deformation and GlyphLens techniques. Their feedback not only validate the practicality of our techniques, but also provide valuable suggestions for further improvements.

1.4 Accompanying Videos

We provide three accompanying videos for the Chapter 3 - 5. Watching the videos can make the proposed techniques easier to be understood. The videos can be found on the YouTube website through the following links.

- Chapter 3: https://youtu.be/9RJ7OMm33x4
- Chapter 4: https://youtu.be/acsFQvv4BOQ
- Chapter 5: https://youtu.be/Mx-GsOxoO18

Chapter 2: Background

2.1 Visual Clutter Management

Overcoming visual clutter or occlusion is an important but challenging task in 3D visualization. Several approaches have been proposed in the past to avoid or remove occlusion. Li *et al.* [76] argued that the use of transparency fails to convey enough depth information for the transparent layers. They use cutaways to remove occlusion and expose important internal features. McGuffin *et al.* [83] used a deformation approach to allow users to cut into, open up, spread apart, or peel away parts of the volumetric data in real time, which makes the interior of the volume visible while preserving the surrounding contexts. An occlusion-free route is visualized by scaling the buildings that occlude the route. Hurter *et al.* [56] used an interactive dig tool to deform the volumetric data by simply pushing the data points, to reduces occlusion.

In 3D streamline visualization, many streamline selection or placement approaches have been proposed with a goal to minimize occlusion or visual clutter. Mattausch *et al.* [82] applied magic volume, region-of-interest-driven streamline placement, and spotlights to alleviate the occlusion problem. Li and Shen [76] proposed an image-based streamline generation approach that places seeds on the 2D image plane, and then unprojects the seeds back to 3D object space to generate streamlines. Occlusion is avoided by spreading out streamlines in image space. Marchesin *et al.* [81] defined the overlap value, the average number of overlapping streamlines for each pixel in the image space, to quantify the level of cluttering and then remove the streamlines that have high overlap values on their projected pixels. Lee *et al.* [73] proposed a view-dependent streamline placement method. In their method, streamlines will not be generated if they occlude regions that are deemed more important, characterized by Shannon's entropy. Another method to alleviate streamline occlusion is to reduce the opacity of the occluding streamlines. Park *et al.* [91] applied multi-dimensional transfer functions (MDTFs) based on physical flow properties to change the color and opacity of streamlines. Xu *et al.* [127] proposed to make the streamlines in lower entropy regions more transparent to reduce occlusion. Günther *et al.* [49] provided a global optimization approach to render streamlines with varying opacity in order to achieve a balance between presenting information and avoiding occlusion. Brambilla [15] measures the degree of occlusion for stream surface and split the surface along a cutting curve to reduce the degree of occlusion. The above occlusion-aware streamline placement methods and transparency modulation methods have their downsides. The problem for the streamline removal methods is that some interesting streamlines may not be shown when they occlude many other streamlines. On the other hand, for the transparency modulation methods, it is difficult to judge the relative depths among the semi-transparent streamlines, and those streamlines can become a distraction. Our F+C streamline deformation method can solve the occlusion problem with better user control while making all the input streamlines easier to see.

There are various types of interactive occlusion removal techniques, such as volumetric probes, virtual X-ray, and projection distorter [32]. The volumetric probe can transform the visualization near the probe in object space. The space near the probe can be enlarged [18, 104] or deformed [83] to make the geometries near the probe less occluded. In addition, the geometries near the explosion probe [101] can be exploded and separated from each other to reduce visual clutter. Deformation techniques have also been used to visualize driving routes and remove the occlusion from their environments [103]. However, the conventional volumetric probes cannot completely remove occlusion or well preserve features in glyph visualization. Virtual X-ray techniques remove occluding objects mostly in image space, using dynamic transparency, cutaway, and 3D magic lens. Dynamic transparency techniques change the transparency of occluding objects [29] to reduce occlusion. Cutaway techniques directly remove the occluding part of the model [35, 30, 76]. 3D magic lens techniques [10, 110, 79, 8] visually transform the contents beneath the lens. However, the virtual X-ray techniques cannot preserve the occluding objects as contextual information. The projection distorter bends the camera ray, instead of distorting the space, to avoid going through occluding objects [95, 26, 126]. It is powerful when being used to discover targets, but it can make the relative positions of the displayed objects confusing. Compared with the existing techniques, our technique not only can provide view-dependent occlusion-free images, but also can better preserve the spatial structures of context features.

2.2 Focus+Context Technique

F+C techniques have been used by different applications that magnify the focus objects while preserving the surrounding context. The techniques include fisheye views [39, 97, 40] and magnification lens [70, 19, 115, 130]. Magic lens [10] changes the presentation of objects to unveil hidden information. In flow visualization, 3D lenses have been applied to show the focus region with greater details [38, 82]. Gasteiger *et al.* [42] use a magic lens to attenuate the focus attribute while showing the context attribute within the lens. Krüger *et al.* [66] use 2D lens to control the visibility of features. Van der Zwan *et al.* [109] blend several levels of detail with a halo-like shading technique to simultaneously show multiple abstractions. NPR lens [88] and the Edgelens [123] interactively distort the features within a 2D lens to emphasize effects and reduce edge congestion, respectively. Among those referenced F+C works, some of them do not solve the occlusion problem. Some methods [38, 82] can reduce occlusion in 3D but do not completely keep the focus objects out of occlusion. Some other methods [66, 42, 109] can completely remove occlusion, but they remove the context information (the occluding objects) at the same time. Only the Edgelens [123] work solves the cluttering problem without removing context.

Some of our proposed techniques are related to the following works with F+C flow visualization using spatial deformation. Correa *et al.* [23] proposed an illustrative deformation system for F+C visualization of discrete datasets. Deformation is used to expose the internal focus region, and an optical transformation is applied to mark up the context region. Because the deformation is performed in data space, the focus can be occlusion-free for only certain view directions. Tao *et al.* [104] devised a deformation framework specifically for streamlines. This method deforms the data grid, and generates streamlines based on the deformed grid. It magnifies the streamlines in the focus while compressing the context region. In their method, because deforming space cannot move individual streamlines according to their specific locations, it is more difficult to avoid occlusion from certain view angles. Both Correa *et al.*'s and

Tao *et al.*'s methods are view-independent, which means the deformation can fail to remove occlusion completely. Besides, both methods require user input of 3D locations on a 2D screen, which makes direct user manipulation difficult. In contrast to these two deformation approaches, our new approach displaces the streamline vertices in 2D screen space and hence can achieve efficient occlusion-free rendering and easy user control for an arbitrary view.

2.3 Glyph-Based Visualization

Glyph-based visualization is a powerful technique for visualizing multivariate datasets. The data attributes are encoded on the glyph's visual channels, such as shape, color, texture, size, orientation and etc. To visualize three-dimensional(3D) volumetric datasets, glyphs can be placed in 3D space to visualize data attributes at local positions or regions. There are three strategies for placing glyphs in 3D space [12]: data driven, feature-driven, and user-driven. Placing too few glyphs in 3D space leads to information loss or missing contextual information. Placing dense glyphs over the entire 3D space, on the other hand, can show more information but also introduces severe visual clutter or occlusion, making the visualization ineffective. Thus, occlusion is a major challenge when visualizing glyphs in 3D space [12].

Glyph-based methods have been used for visualizing multivariate data to preserve the spatial relationship of data. Particle simulations are widely used in fluid dynamics [67], cosmology [116] and molecular dynamics [7]. Spherical glyphs are commonly used to visualize particle datasets with the sphere's color representing a scalar value of the particle [46, 45, 65]. To visualize the local vector fields, one can either use simple arrow glyphs [94] to visualize the 3D directions or to use a probe to visualize local change of velocity [28]. Even though the integration-based methods such as streamlines or stream surfaces can also visualize the vector field, they cannot completely replace the glyphs when the data is noisy or uncertain. To visualize diffusion imaging dataset, superquadric glyphs [62, 98] are used to visualize tensor field dataset, while HARDI glyphs [92, 99] are used to visualize high angular resolution diffusion imaging (HARDI) dataset.

Glyphs are very powerful in visualizing two-dimensional flow fields because there are less concerns of visual clutter or occlusion on 2D images. Kirby *et al.* provide a 2D flow visualization that uses arrow glyphs to represent velocity and ellipse glyphs to show the rates of strain tensors [63]. Peng and Laramee use 2D glyphs to visualize the flow on the surfaces of an unstructured adaptive resolution boundary mesh [94]. In 3D flow visualization, the vector glyph can be used as a simple and direct rendering of the local vector field [24, 31]. De Leeuw and van Wijk design a probe/glyph to show characteristics of the flow such as velocity and velocity gradient tensor [28]. The avoidance of visual clutter and the occlusion of important details become important factors in determining the effectiveness of these glyph-based techniques. The 2D projections of 3D glyphs can overlap if they are not well placed in the 3D space. Boring and Pang [13] apply different lighting conditions to a three-dimensional hedgehog glyph and other geometry-based forms of the flow to emphasize different vector directions. By only highlighting the glyphs corresponding to user-defined directions, they reduce the displayed data and hence alleviate the clutter problem. Laramee [71] addresses visual clutter by resampling the vector field and generating a smaller number of summary vectors that leads to a sparser glyph placement. Our work is also related to the angular distribution in the high angular resolution diffusion imaging

(HARDI). The angular distributions are described by certain models or functions, which can be visualized by 3D glyphs [92, 61, 99].

2.4 Vector Field Histogram Visualization

Vector Field Histogram Two-dimensional vector field histograms have been widely used in computer vision to analyze the statistics of directions or orientations. The applications include real-time obstacle avoidance in mobile robots [11], crowd flow abnormality detection [58], and human detection using HOG [27]. There are very few modern techniques of visualizing the vector field histogram directly. Neuroth *et al.* [89] visualize local 2D velocity histogram with low visual clutter. Their binning strategy based on Cartisian coordinates also allows showing the distribution of velocity magnitudes, but it is not intuitive to express the distribution of vector orientations.

Glyph-based Uncertainty Vector Field Visualization Part of our work is closely related to vector field uncertainty glyphs that show the uncertainty information of local regions within a vector field. Lodha *et al.* place uncertainty glyphs along particle traces to show the magnitude of the deviation between two streamlines [78]. Wittenbrink *et al.* present a technique that expresses the variation of both vector direction and magnitude using different glyph shapes [121]. Zuk *et al.* discuss the interactive rendering of glyphs for visualizing uncertainty in a bidirectional vector field [131]. Hlawatsch *et al.* extend the vector field uncertainty glyph to visualize unsteady flows [53]. This technique shows the possible ranges of flow directions by angles and the time by the glyph radius. Jarema *et al.* use 2D glyph to visualize the Gaussian Mixture Models (GMM) of the ensemble vector field [60]. The last four techniques [121, 131, 53, 60] only visualize the measures of distribution (e.g. mean and variance) or the models of distribution (e.g. GMM). Besides, these glyphs with 2D geometries do not effectively visualize 3D directions. The use of our crystal glyph is a three-dimensional geometry that directly visualizes the histogram of 3D vector directions.

2.5 Time Series Browsing

As the speed of processors and degree of parallelism continue to increase, the high cost of storage and slow speed of data movement have now become the major limiting factors for applications to take full advantage of the computation power in large scale parallel machines. For scientific applications such as climate modeling, nuclear reactor simulation, and combustion engine design, the overwhelming amount of data generated by time-varying simulations forces the scientists to cut down the amount of data that can be stored to disk. Since it is not possible to analyze results from all time steps, scientists often can only look at a small fraction of data at some pre-selected time interval decided with simple heuristics. When the phenomena modeled by the simulations have complex patterns and occur at unknown frequency, the question of what time steps to use for analysis and visualization is still illusive. Researchers have focused on various aspects of time-varying data visualization in the past two decades. Examples are feature tracking, compression, high-dimensional rendering, and interaction.

One strategy to overview time-varying data is to depict the information from all time steps simultaneously. As histograms are frequently used to display data distributions, Akiba *et al.* designed a visualization interface based on *Time Histogram* [5]. Time Histogram is a 2D image that concatenates data distributions over time, allowing the user to design effective transfer function across multiple time steps. Later Time Histogram was combined with parallel coordinates for multivariate volume exploration [6]. Besides, other statistics over time can be utilized too. One example is the importance curves presented by Wang *et al.* [113]. For each spatial block, its importance curve is computed from the mutual information between adjacent time steps over time. As mutual information represents the similarity between local time steps, importance curves can indicate when the distribution within a data block has changed.

In addition to considering data distributions, direct visualization of value changes over time can be beneficial too. Given a fixed spatial location, the data values over time form a time series, or called Time Activity Curve (TAC) in medical applications [122, 50]. In the user interface designed by Fang *et al.*, data in the spatial domain can be classified based on their TAC patterns, and the region of interest can be detected based on user-specified TACs [33] . Woodring and Shen [124] used k-means clustering to detect the representative TACs from all data points, and designed a spreadsheet-like interface to overview all representative TAC and its distribution over the spatial domain. Later Woodring and Shen use the TAC-based representation to detect the evolution of features, and create a transfer function that can be automatically adapted to capture the change of data ranges[125]. To track the propagation of feature, Lee and Shen presented an idea called TAC-based distance field [74]. Given a user-specified feature, their approach is to first model the feature as a TAC, and then estimate when and where the feature TAC appears in all data TACs. Based on the estimated locations and time steps, the propagation of the feature can be depicted in a 2D plot.

Because of the limited screen space, it is also beneficial to only display a subset of time steps that can represent the change of importance within the data. Lu and Shen presented a storyboard-like interface [80] where the key time steps are distributed on 2D screen space based on similarity measures of the data. By linking these key time steps following the order of time, the user can see how the data evolve. Lee and Shen modeled the change of TAC pattern as transition of states, thus reducing the TAC at each location into a state machine that has fewer states [112]. The concept of state-transition graph is also used by Wang *et al.* [112] and the *TransGraph* presented by Gu and Wang [48]. Along this direction, our goal is to select salient time steps that can be the most representative for the whole sequence. We also design effective user interfaces to allow effective browsing of time-varying data.

2.6 Interaction Devices

Traditionally, users interact with 3D scientific visualization using the mouse. The mouse, as a 2D interaction tool, also has limited abilities to perform 3D operations, such as rotation [20, 100]. Using direct-touch interaction [129, 64, 128] on the touch screens, users can directly interact with the 3D visualization on the display space. However, none of the 2D space interaction techniques, such as mouse or touch screen, provide intuitive 3D operations such as rotating an object or specifying a 3D slicing plane. On the other hand, 3D interaction devices such as the 3D mouse by 3Dconnexion and motion cameras such as Microsoft Kinect, Leap Motion and Intel RealSense, can easily specify 3D position and orientation with six degrees of freedom (DOF). Jackson *et al.* allow users to specify fiber orientations using a depth sensing camera and interact with 3D thin fiber data [59]. Besides the 2D display devices, immersive techniques such as cave automatic virtual environment (CAVE) [25] and virtual reality headset [69], can display stereoscopic views with head tracking. In this work, we also discuss how these 2D/3D interaction techniques can be used to manipulate our lens and visualization in 3D space with the assistance of different display techniques.

Chapter 3: View-Dependent Streamline Deformation and Exploration

3.1 Introduction

Streamlines are commonly used for visualizing three-dimensional (3D) vector and tensor fields. When too many streamlines are shown, however, getting a clear view of important flow features without occlusion is difficult. Even though a single streamline does not cause much occlusion compared with higher-dimensional geometry, mixing many streamlines of different depths together can generate a very confusing image.

Although through interactive seeding of streamlines one can control the amount of occlusion and visual cluttering, it makes finding specific flow features, and hence understanding their surrounding context, more difficult. To reach a balance between displaying too much information and too little, focus+context (F+C) techniques provide a nice solution. In Chapter 3, we present a streamline deformation technique to achieve an F+C view of 3D streamlines. Earlier streamline deformation approaches [23, 104] deform the 3D space of the flow field. The main drawback of the space deformation approaches is that it is not easy for users to control the deformation of continuous 3D space to remove occlusion completely.

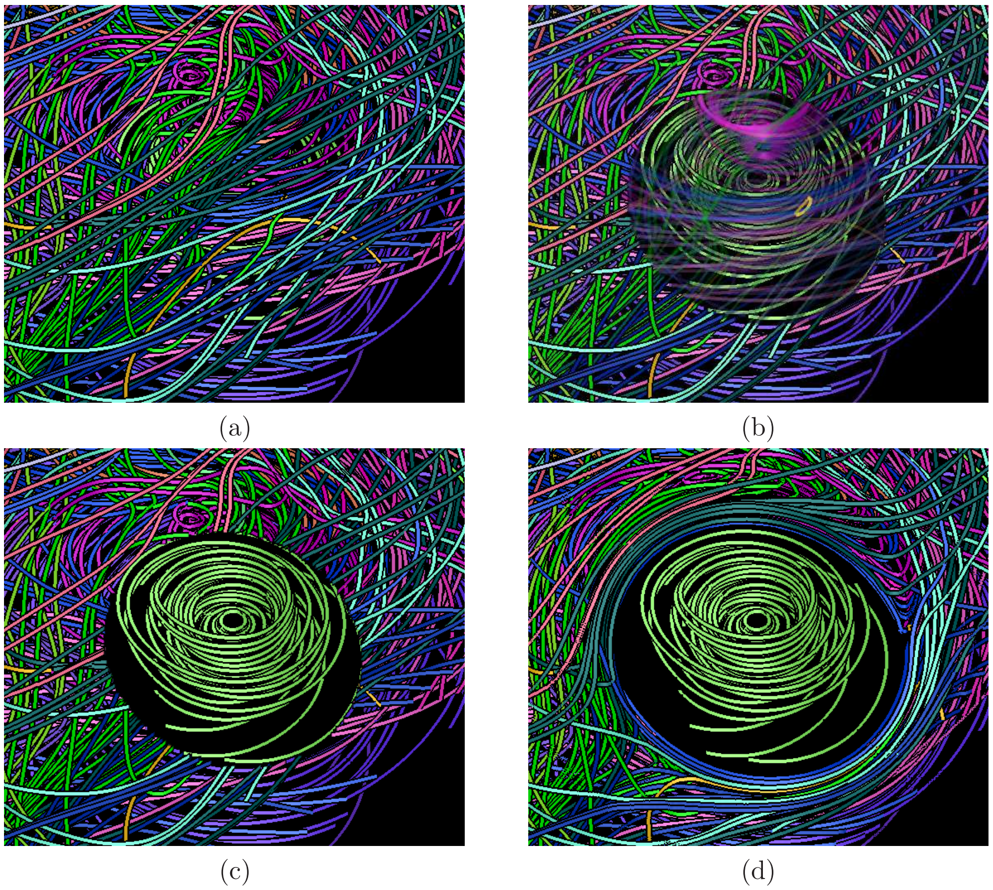


Figure 3.1: Three methods are compared to reveal the feature in Hurricane Isabel dataset, a group of vortex-shaped streamlines, originally occluded by other streamlines. (a) original rendering with the features occluded; (b) transparency method is applied; (c) cutaway method that removes both the occluding factors and the contexts; (d) our deformation method is applied, which completely removes the occlusions with context information preserved.

Adjusting transparency is another common method used to expose occluded features [91, 127]. But it is difficult to set the transparency value and define the semi-transparent region so that a clear view of both the F+C objects can be obtained. For example when visualizing a vortex in the Hurricane Isabel dataset, as shown in Fig. 3.1a and 3.1b, the vortex-shaped streamlines are originally occluded in (a) and then become visible after making some streamlines more transparent in (b). However, the context of the flow around the vortex is mostly lost if the occluding streamlines are too transparent, and with transparency, the depth relationships among streamlines are more difficult to discern. Furthermore, depth sorting is required to render semi-translucent objects correctly, but it is difficult to sort a large number of line segments in real-time. Cutaway method is another common technique to deal with occlusion. It not only removes all the objects occluding the focus features but also removes the context features in that region, as show in the example of Fig. 3.1c.

We propose a view-dependent deformation model for interactive streamline exploration. After users defining a *focus region* in screen space, streamlines occluding the focus region are deformed and gradually moved away based on two deformation models, a point model and a line model. The point model moves streamlines away from the center of the focus region, while the line model cuts the streamlines along the principal axis of the focus region and moves the streamlines to both its sides. Because occlusion has a view-dependent nature and our deformation is performed in screen space, occlusion can be more effectively removed. Compared to the other methods mentioned above, our deformation technique can better preserve the context streamlines in the vicinity of the focus feature and through the graduate deformation transition, as shown in 3.1d and the accompanying video. As an application of our deformation model, an interactive 3D lens was presented to allow users to freely move streamlines away from selected areas on the screen using both mouse and direct-touch interaction. To explore more complex features, we also introducing two more complex lenses, *layered lenses* and *polyline lens*. The layered lenses deform different layers differently, so we can show the features and contexts of different layers more clearly.

The polyline lens can cut the streamlines with a series of connected line segments and deform the surrounding streamlines smoothly to the side. Two real vector field datasets (Hurricane Isabel, Solar Plume) and a tensor field dataset (brain images from a brain tumor patient), were used to demonstrate our deformation framework. Additionally, neurosurgeons and neurologists were invited to evaluate our system and provide valuable suggestions on further studies.

3.2 Algorithm

The goal of our algorithm is to expose interesting features in the *focus* area by deforming the occluding streamlines. In our deformation model, depending on the approximate shapes of the focus area, two shape models, the point model and the line model, are used. These two shape models can generate effective deformation for different shapes of focus area, and is easy for users to control interactively. For each of the two shape models, we design a screen-space deformation algorithm that displaces vertices of the occluding streamlines.

3.2.1 Algorithm Overview

The input to our algorithm is a set of densely distributed streamlines, which can be roughly divided into *focus streamlines* and *context streamlines*. Focus streamlines are what the user is interested in visualizing without any occlusion, e.g. a cluster of streamlines with a similar shape, or a group of streamlines passing through a user-specified region. The remainder are context streamlines. Any streamlines that block the focus streamlines will be deformed and moved to the side. To perform the deformation, our F+C deformation model divides the screen space into three regions: *focus region*, *transition region*, and *context region*, as shown in Fig. 3.2. The focus region is a user-specified region in screen space that contains the features of interest; the transition region is the area that is immediately adjacent to the focus region used to contain the deformed streamlines; and the context region is the rest of screen space that contains undeformed streamlines. Although streamlines are defined in 3D space, our deformation takes place in 2D screen space, i.e., streamlines are deformed without changing their original depth. For this reason, our shape model described below in Section 3.2.2 is defined in 2D space.

The goal of the deformation is to compress and move the occluding streamline segments from the focus region to the transition region. To make space for these streamlines, streamline segments that were originally in the transition region will also be compressed and moved towards the outer boundary of the transition region. Essentially, the deformation makes sure that the features of interest in the focus region occlusion-free to the view. The occluding streamlines are deformed but not removed, providing the context to the focus area. Any other streamlines outside of these two regions remain unchanged.

We have two design goals for our deformation model:

1. The deformed streamline should preserve its shape as much as possible, even though the shape is compressed. In other words, the relative positions of the streamlines and their vertices should be preserved.

2. After the deformation, the vertices should be distributed on the streamline as uniformly as possible. In other words, any two connected vertices on a streamline should not be placed too far from or too close from each other, compared to other pairs of connected vertices. Otherwise, the streamlines will be jagged and a long edge between two connected vertices may cut across the focus region.

In our algorithm, the deformation of a streamline is achieved by displacing its projected vertices in screen space. During deformation, we displace the deformed vertices away from the center of the focus region. The amount of the displacement is determined by each streamline vertex's distance to the center of the focus. We design a *displacement function* to place the deformed streamlines in the transition region, preserving their shapes as much as possible, in order to satisfy our first design goal. In addition, an adjustment is applied to the vertex displacement to make the deformed streamlines satisfy our second design goal.

3.2.2 Shape Models

We designed two shape models, a point model and a line model, to represent the shape of the focus area. Fig. 3.2 illustrates these two models. The point model is designed for focus areas that have a circular shape, while the line model is for focus areas that have a linear shape. Both shapes are typical for streamlines. The first section of the accompanying video demonstrates and compares the two shape models. We note that besides the simple regular shapes (point and line), a more complex irregular shape could be used, e.g. a skeleton or principal curve of the streamline cluster and their surrounding curved tube-shaped regions. However, the resulting context streamlines would be distorted, making it hard for users to mentally recover their original shape. Therefore, they are not considered in this work.

Point Model

As shown in Fig. 3.2a, the point model is composed of a 2D focus area (the inner black ellipse in the figure), a transition area (the area between the inner black ellipse and the outer green ellipse), and its center O. The inner focus area can also be

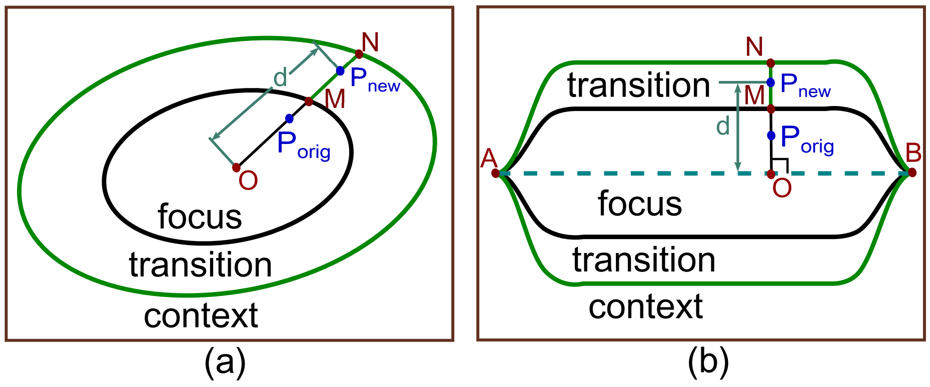


Figure 3.2: Sketches of the two deformation models: (a) the point model and (b) the line model. The region inside the black boundary is the focus region. The region between the black and green boundaries is the transition region. The region outside the green boundary is the context region. During deformation, the vertex moves from P_{orig} to P_{new}. In (a) O is the center of the black ellipse, while in (b) O is the intersection point between the line AB and the perpendicular line of AB passing through P_{orig}. M and N are the intersection points between the line OP_{orig} and the two boundaries.

represented by a convex polygon if desired. The convex polygon and the ellipse-based focus areas have their own advantages. The convex polygon model can more tightly cover the focus streamlines, while the ellipse focus has a relatively smoother and more regular shape and can be represented analytically. The center of the focus area in the point model is the reference point from which the streamlines are moved away.

Line Model

Fig. 3.2b shows our line model. The line model is composed of a principal axis line (line AB), a linear bounding area immediately outside of the principal axis that represents the focus region, and the transition area that is an expanded area outside of the inner focus. We call this bounding shape *open blinds*, because it looks like two window blinds that are open. Around the two end points A and B, the shape of the open blinds is represented by the *tanh* function. During the deformation, we cut the streamlines in the open blinds region by the axis line and move the streamlines to both sides of the axis line along the direction normal to AB, until the focus area is clear.

The point model and the line model are selected based on the shape of streamlines. If we apply the point model to straight streamlines distributed in an area as in Fig. 3.2b, for example, some vertices will have to travel a long distance to the region around the point A or B, resulting in a large distortion. On the other hand, the point model is good for circular focus regions to avoid unnecessary cutting of the streamlines happening in the line model. The overall shape of the focus streamlines determines which model to use. To measure the overall shape, we use the roundness of the focus area, which is a minimum enclosing ellipse of the focus streamlines. If the major radius of the ellipse is much larger than the minor radius, i.e. low roundness, then we use the line model; otherwise, we use the point model.

3.2.3 Deformation Model

Our method achieves the deformation by displacing its vertices iteratively to preserve the smoothness of the streamlines throughout the whole process, as required by our second design goal. This means the relative positions of the vertices on a streamline need to be updated constantly; otherwise, the distance between two adjacent vertices on a streamline can become too large, and consequently, the line segment between them can cut across the focus region and still cause occlusion. To achieve this, the force-directed algorithm [37], which considers the relative positions of the points by moving them iteratively to generate the final layout, inspires the design of our approach. In our method, a vertex does not just follow a linear path and move towards a single predetermined direction. Instead, the deformation is computed through multiple iterations and generates an animation sequence. In each iteration, the vertex adjusts its moving direction so that its relative position is preserved throughout the animation. This animation sequence provides the context, and the final layout of the streamlines provides an occlusion-free view of the focus area, as shown in Fig. 3.1d.

When a streamline is deformed, in each iteration the position of a vertex on the streamline is modified based on two considerations. First, the vertex should gradually move out of the focus area. Second, the vertex should not be placed too far away from its neighboring vertices. Based on these considerations, we control the displacement of a vertex using two subcomponents, each of which is represented by a velocity and a moving direction. Mathematically, the vertex movement can be written as:

$$P' = P + v \cdot \vec{w} + v_c \cdot \vec{u} \tag{3.1}$$

where P' is the new position of the vertex, P is the old position, $v \cdot \vec{w}$ represents the movement that moves the point out the focus area, and $v_c \cdot \vec{u}$ makes sure that the new point position is not too far from its neighbors. Hereafter we refer to $v \cdot \vec{w}$ as the *major displacement*, and $v_c \cdot \vec{u}$ as the *minor adjustment*. The two directions $\vec{u}$ and $\vec{w}$ are shown in Fig. 3.3a. Below we explain each of the terms in detail.

Major Displacement

At each iteration, the streamline vertex moves away from the focus area along the direction $\vec{w}$ at a speed of v. The moving direction $\vec{w}$ is related to the underlying shape model in use. For the point model, as shown in Fig. 3.2a, $\vec{w}$ is from the centroid O to the current vertex position P, i.e. $\vec{w} \parallel OP$, which is also shown in Fig. 3.3a. If the line model is used, shown in in Fig. 3.2b, $\vec{w}$ is the normal direction of the line AB. If we draw a line through point P and perpendicular to AB, it intersects with AB at O, and we have $\vec{w} \parallel \vec{OP} \perp \vec{AB}$. For both shape models, we generalize the definition of $\vec{w}$ as:

$$\vec{w} = normalize\left(\vec{OP}\right) \tag{3.2}$$

The moving speed v determines the amount of major displacement in one iteration. Assuming P_{orig} is the vertex's original position, and d is the distance between O and the vertex's final position P_{new} after deformation, i.e. $d = |\vec{OP_{new}}|$. For the vertex to reach a distance of d from O, the speed of the movement for P is determined by how much the point has yet to travel, that is:

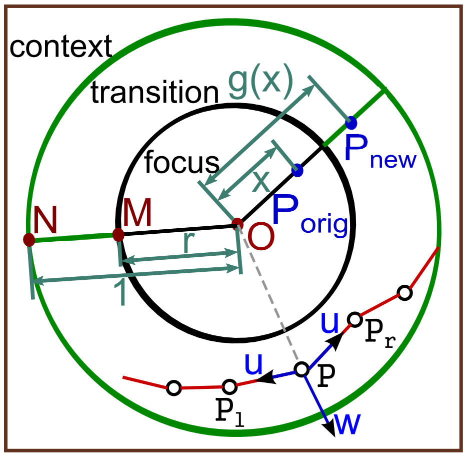
$$v = (d - |\vec{OP}|) \cdot \alpha \tag{3.3}$$

where α is a constant that has a value in $(0, 1)$. It controls the magnitude of the moving speed. An empirical value of α is 0.01. Because as P moves away from O, $|\vec{OP}|$ keeps increasing, and thus the speed of v keeps decreasing, until the vertex P stops moving and arrives at its final position P_{new}.

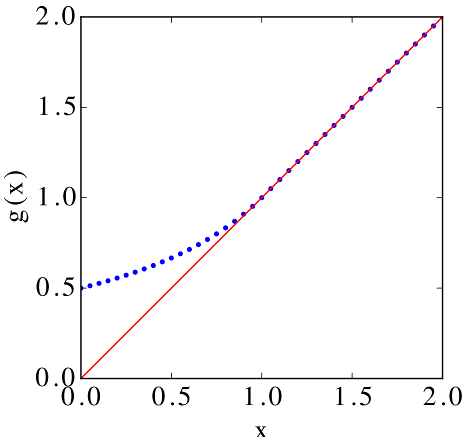
From Equation 3.3, we know that d determines the final position of each streamline vertex after the deformation. Because we want to compress and preserve the shape of the streamline, we control the value of d using a monotonically increasing function to transform the original distance $|\vec{OP_{orig}}|$ to a larger value d. This function, denoted as g, takes a normalized value of $|\vec{OP_{orig}}|$ as its input and has the general form:

$$d = g\left(\frac{|\vec{OP}_{orig}|}{|\vec{ON}|}\right) \cdot |\vec{ON}| \tag{3.4}$$

where $|\vec{ON}|$ is the distance between O and the outer boundary of the transition region along the moving direction of $\vec{w}$, as shown in Fig. 3.2. $|\vec{OP}_{orig}|$ is normalized to be between 0 and 1 by dividing its value by $|\vec{ON}|$.



(a)



(b)

Figure 3.3: (a) Illustration of the point model in the normalized space. $\vec{w}$ and $\vec{u}$ are the two displacement directions for the point at P. P_l and P_r are the two vertices connected to P on this streamline. (b) Blue dotted line: normalized displacement function g in Equation 3.10 when $r = 0.5$. Red line: a reference displacement function $F(x) = x$, which gives no displacement for the entire domain.

Fig. 3.3a is an illustration of the point model similar to Fig. 3.2a, marked with several normalized distances to illustrate the displacement function $g(x)$. The normalized value of $|\vec{ON}|$ is 1. We define r as the normalized value of $|OM|$, i.e. $r = \frac{|\vec{OM}|}{|\vec{ON}|}$. Before deformation, vertices on the non-deformed streamline segments distribute over both the focus and transition regions, so $|\vec{OP}_{orig}|$ varies in the range $[0, |\vec{ON}|]$, i.e.

$\frac{|\vec{OP_{orig}}|}{|\vec{ON}|}$ varies in the range $[0, 1]$. After deformation, the transformed vertices will all go to the transition region, so d varies in the range $[|\vec{OM}|, |\vec{ON}|]$, i.e. $\frac{d}{|\vec{ON}|}$ varies in the range $[r, 1]$. Essentially, g monotonically transforms a value in $[0, 1]$ to a larger value in $[r, 1]$.

Displacement Function

Instead of designing the displacement function g as a linear function , we apply the transformation function of fisheye lens [97] to design a non-linear displacement function g to control the speed of streamline vertices as discussed in the previous section and produce a smoother deformation across the region boundary. Here we assume a point model with a circular boundary shown in Fig. 3.3a to explain the idea. Below, we first give the design goal of the function g, and then solve g.

Our first criterion is that, as shown in Fig. 3.3a, a vertex located at O should be moved to the inner boundary of the transition region at M, and a vertex located at the outer boundary of the transition region at N should remain on the outer boundary. So we have:

$$g\left(0\right) = r \tag{3.5}$$

$$g\left(1\right) = 1 \tag{3.6}$$

Secondly, the function g must be a monotonically increasing function to make sure that the deformed streamlines and their vertices have the same relative positions to O after the deformation. So we have:

$$\frac{dg(x)}{dx} > 0 \tag{3.7}$$

$\frac{dg(x)}{dx}$ describes the amount of distortion in the deformed space at a point whose distance to O is x. We know that $\frac{dg(x)}{dx}$ is 1 for any points in the context region because they will not move. To ensure that the amount of distortion smoothly changes from the transition region to the context region at their boundary point N in Fig. 3.3a, $\frac{dg(x)}{dx}$ should be continuous at that point. So we know

$$\left.\frac{dg(x)}{dx}\right|_{x=1} = 1 \tag{3.8}$$

Finally, our last design criterion is that, from a position near O to a position farther away from O, the amount of distortion should also change monotonically. The value of $\frac{dg(x)}{dx}$ should monotonically increase from a value less than 1 to the value 1 when x changes from 0 to 1. The change of the distortion amount is the second derivative of displacement function $\frac{d^2g(x)}{dx^2}$. So we get:

$$\frac{d^2g(x)}{dx^2} > 0 \tag{3.9}$$

Combining the four criteria from Equation 3.5 - 3.9, we can solve the displacement function g. There is more than one solution for g, and we use the simplest one:

$$g(x) = \frac{(r-1)^2}{-r^2x+r} - \frac{1}{r} + 2 \qquad x \in [0,1] \tag{3.10}$$

To show that the above function satisfies the four criteria, we plot Equation 3.10 in Fig. 3.3b. The figure clearly shows that Equations 3.5 - 3.9 are satisfied.

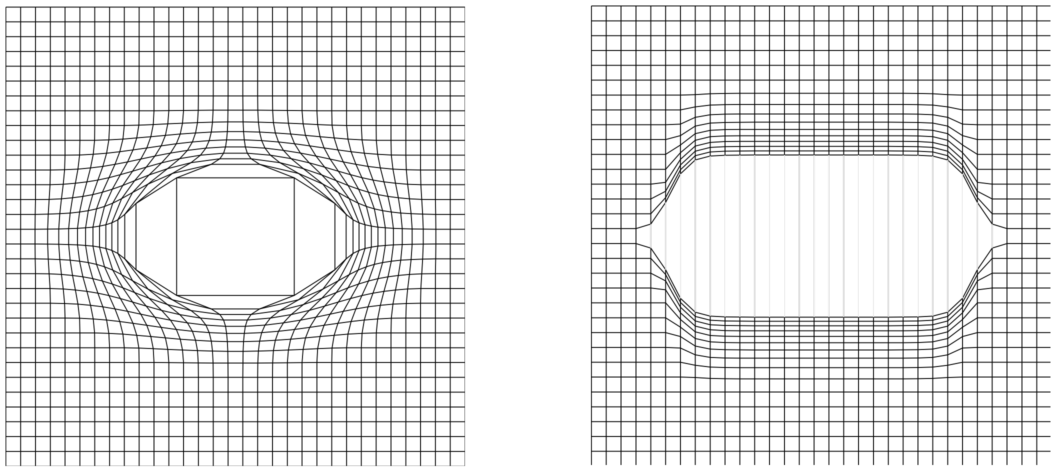

(a) point model (b) line model

Figure 3.4: Deformed grids using two shape models with $r = 0.5$.

If we use the displacement function g to move a regular grid with our two shape models, we get the deformed grids shown in Fig. 3.4. Note that we create a void focus region at the center of the grid, because $g(x) \geq g(0) = r$ for $x \in [0, \infty)$; all the streamline vertices will be cleared out of the focus region. We also notice that for the same amount of distortion (same value of r) in the space we can create a larger void space with the line model shown in Fig. 3.4b than the point model shown in Fig. 3.4a. Besides, the deformed grid in the point model shows more stretching but less compression than the deformed grid in the line model.

Minor Adjustment

The minor adjustment plays an important role in making the vertices uniformly distributed on the deformed streamline and thus satisfies the second design goal of our deformation model. During our deformation process, some edges can be stretched more, which makes those portions of the streamline jagged. Furthermore, the long edge can cut across and hence still occlude the focus region, which is undesired.

As shown in Fig. 3.3a, the vertex P is connected to two vertices at point P_l and point P_r with two edges. A local approach to uniformly distribute the vertices over the streamline is that each vertex moves towards the farther one of the two neighboring vertices through multiple iterations so that the relative positions among the vertices are preserved. This shortens the longest edge in each iteration, and eventually no edge is much longer than the other edges.

The minor adjustment $v_c \cdot \vec{u}$ is a product of a constant adjustment speed v_c and an adjustment direction $\vec{u}$. $\vec{u}$ is a normalized direction parallel to the longer connected edge, which is defined as:

$$\vec{u} = \begin{cases} normalize(\vec{PP_l}), & \text{if } |\vec{PP_l}| > |\vec{PP_r}| \\ normalize(\vec{PP_r}), & \text{if } |\vec{PP_l}| < |\vec{PP_r}| \\ \vec{0}, & \text{if } |\vec{PP_l}| = |\vec{PP_r}| \end{cases}$$

Note that the minor adjustment may move the vertex in a direction other than the direction of major displacement $\vec{OP}$, which ends up changing $\vec{w}$ in the next iteration. This change is not recoverable for the later iterations. Therefore, when the viewpoint or the focus region is changed during the deformation, if we continue the deformation with the new value of $\vec{w}$ or $|\vec{ON}|$, then we can still keep the focus region occlusion-free, but we may not be able to preserve the shape of deformed streamlines in the transition region. The solution is to recover the streamlines' original positions and redo the deformation from the first iteration.

3.3 Interactive Deformation

In this section we introduce a streamline exploration tool, interactive 3D lens, based on the deformation algorithm described above. To overcome the occlusion problem, the lens can be placed anywhere in the image space with an adjustable depth to peel away the occluding streamlines layer by layer. To enhance the experience of user interaction, we use a direct-touch technique to allow users to control the lens directly on the screen with multi-touch gestures.

3.3.1 Interactive 3D Lens

The interactive lens is useful when users want to freely explore the computed streamlines. The lens defines a focus region with a certain depth range. Any streamline that is entirely or partly under the lens and closer to the viewer than the far side of the lens is treated as a context streamline and will be moved out of the focus region in the screen space. The other streamlines are all treated as focus streamlines that will not be deformed, even when they are not the interesting features.

We design our interactive 3D lens as a 3D cylindrical object, shown as the black cylinder in Fig. 3.5. The lens resides in screen space with depth defined, shown as the blue cube in the figure. The axis of the cylinder is perpendicular to the screen, i.e., parallel to the z axis, and the top surface of the lens is parallel to the XY plane. The length of the cylinder is used as the *lens depth*. For the point model, the lens has an elliptical surface; while for the line model, the lens surface has an open blind shape. As the example shows in Fig. 3.5, there are three streamlines (one red and two green), but only the red streamline that intersects with the lens will be deformed. In the deformation, all the vertices on the red streamlines will be moved out of the

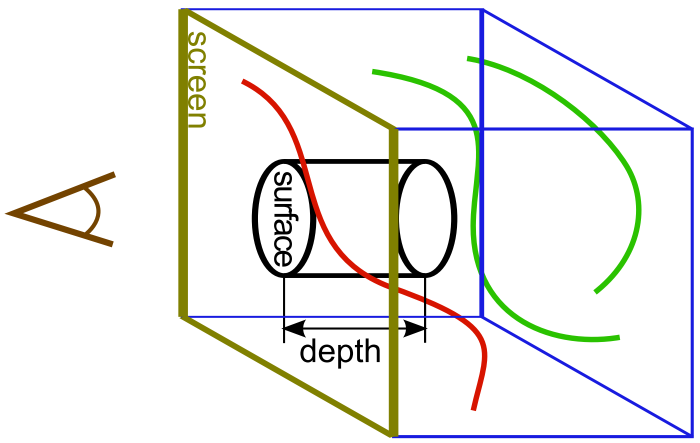
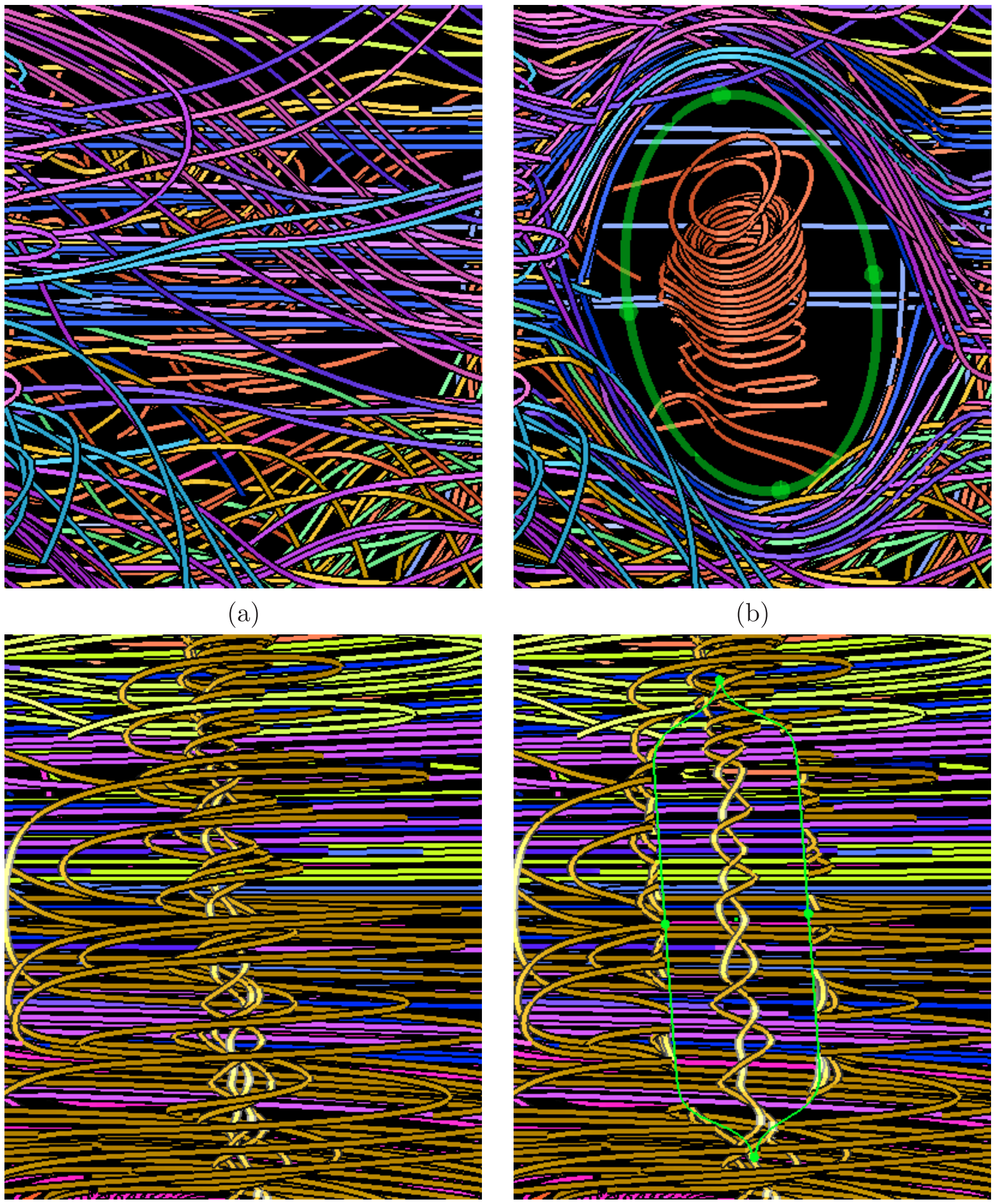


Figure 3.5: The 3D lens. The blue cube denotes the 3D space. The yellow square denotes the 2D screen space. The lens has an ellipse-shaped surface on the plane of the screen. Inside the cube, there are three streamlines.

surface region in screen space, even for the right tail of the red streamline that is not inside the cylinder.

Fig. 3.6 (a)-(d) show examples of the interactive 3D lens using the point and the line deformation models. In Figures 3.6a and 3.6b, we use a lens with the point deformation model to move the straight streamlines in the front away and reveal the vortex gradually through animation (please see the accompanying video). In Fig. 3.6c and 3.6d, a lens with the line deformation model is used to break the outside of the vortex in two, so that the inner structure, which was previously occluded, now becomes visible. Even though the streamline around the inner structure only partially intersects with the 3D lens, we treat the entire streamline as the context streamline and let all the vertices on it be deformed out of the screen space focus region in order to ensure the continuity of deformation on this streamline.



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

Figure 3.6: A point-model lens is applied to the streamline visualization in (a) to push layers apart and reveal the hidden red curled vortex in (b). A line-model lens cuts up a flow field in (c) into 2 halves and pushes them aside to expose the olive colored helix twisted vortex in (d).

In the interactive system, users can use the mouse buttons to modify the size, shape, and orientation of the lens surface on the screen to specify the focus region. In addition, users can use the mouse wheel to change the lens depth to explore the streamlines at different layers in z direction.

In summary, the interactive 3D lens uses our view-dependent deformation model to explore and reveal hidden streamlines in the flow field. By using the lens, users can freely move their focus to different locations and change the shapes of the lens interactively to search for interesting streamlines. Because the lens is always perpendicular to the image space, users can rotate the field and change the depth of the lens to explore the 3D space. Users can progressively discover the features at different depths even when the features are deeply buried inside the field, or move the lens around to explore different parts of the focus streamlines when they are very long. Because our deformation can be performed interactively, users can go back and forth to replace the deformed streamlines to enhance their understanding of the 3D features.

3.3.2 Specialized Lenses

The interactive 3D lens described above has received positive feedback and suggestions of extension from researchers. Thus, we enhance our lens with additional features to meet special needs in feature exploration. Streamline features look different at different depths and screen locations. A good interactive lens should adapt itself to those different features in order to better preserve the context and accelerate the process of finding features. In this section, we propose the *layered lens* and the *polyline lens*, which can explore features at different depths and different screen locations with improved flexibility and adaptability.

Layered Lenses

In camera space, since streamlines in different layers have different shapes and orientations, a lens used for the top-most layer may not be able to reveal features clearly at a different depth layer. For example, when users use the open blind lens to cut the streamlines and move them to the side, they usually want to make the cut direction follow the trajectories of the streamlines. If the directions of the streamlines vary in different layers, the cut directions should also be different in different layers. Another reason to have layered lenses is that users usually want to explore the data in a wider screen area first at the top layers, and then focus on a smaller screen area of interest to inspect the inner structures. To make the deformed streamlines on the top layers unchanged as the context while exploring the smaller scale features in the inner layers, our layered lenses can make different screen sizes of focus regions in different depths and present the users with multiple layers of contexts.

The layered lenses are composed of a set of regular lenses with increasing depths and decreasing screen sizes, as shown from left to right in the Fig. 3.7. The lenses are connected to each other in the depth direction. Between two adjacent lenses, the screen area of the lens with a larger depth is contained within the screen area of a smaller depth lens. A streamline passing through more than one lens will be deformed by the lens with the smallest depth, because deforming streamlines by multiple lenses can result in large distortion. Also, lenses with smaller depth have larger screen size, and thus can create a larger void region to look through. For example in Fig. 3.7, the green streamline intersects with both the green lens and the blue lens. Because

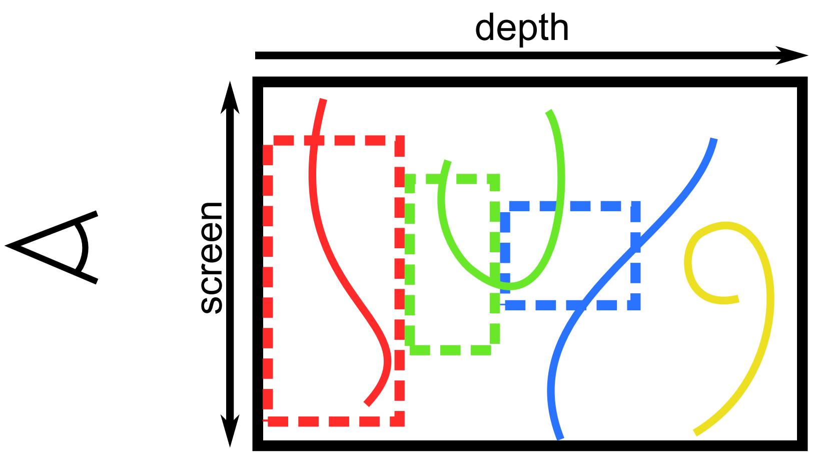


Figure 3.7: Layered lenses are composed of three connected lenses, shown as dash line frames in three different colors. The red lens has the smallest depth and largest screen area, while the blue lens has the largest depth and smallest screen area. The streamlines will be deformed by the lens that has the same color with the streamline. The yellow streamline is the focus feature and will not be deformed.

the green lens has a smaller depth and larger screen size, the green streamline will be deformed by the green lens but not the blue lens. After the deformation, the streamlines originally located in the three lenses will be removed from the three 3D focus regions. Then, users will be able to see the yellow focus streamline and also see the three deformed streamlines (red, green and blue streamlines) as contexts in different layers.

To demonstrate the use of our layered lenses, we apply it on white matter tracts which are generated from a brain diffusion tensor dataset using the *3D Slicer* software [1, 34]. This dataset has a resolution of $256 \times 256 \times 51$ with a 3×3 tensor matrix on each grid point. Because this data is from a patient with a brain tumor, we visualize the tumor as a polygonal surface in addition to the while matter tracts. By visualizing the streamlines and the surface together in Fig. 3.8, users can explore

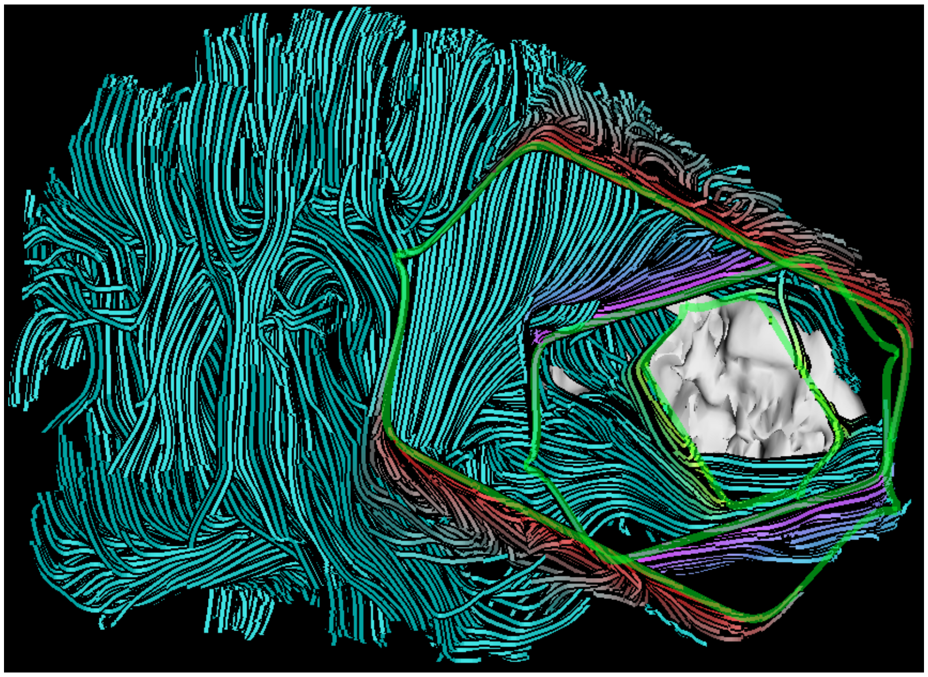


Figure 3.8: Using layered lenses to open the brain tumor data set. Deformed streamlines are colored by lens and deformation magnitude, e.g. red streamlines are deformed by the top-most lens, and regions of darker red are deformed more than regions of lighter red.

the tracts, the tumor and their relationships. From the figure, we can see the white brain tumor through the three lenses of different sizes and orientations. We give the deformed streamlines different colors according to their affecting lens in order to differentiate them from the unchanged ones and the ones deformed by other lenses. The biggest lens is closest to the viewer and the color of its deformed streamline is red; while the smallest lens is the deepest inside the volume and has yellow deformed streamlines. With the layered lens, different layers of streamlines can coexist in the same image, which gives users better context while exploring the 3D space.

Polyline Lens

As said previously, having a lens with an irregular shape, e.g. an arbitrary polygon, is not desired because the affected streamlines will be distorted too much and tend to conform to the shape of the lens. On the other hand, a feature can have an irregular shape, which can be difficult to inspect in detail by either the ellipse lens or the open blind lens. In order to allow the focus region to have more general shapes, we propose the polyline lens, which supports more complex shapes with relatively little distortion. It divides the streamlines by a series of connected straight lines, or a polyline, and pushes the context streamlines to the side. It works similarly to the open blind lens, but with more flexible shapes and can easily fit a curvy streamline.

The deformation of the polyline lens mostly follows the deformation of the line model, except for the vertices near the joint of two connected line segments. For example in Fig. 3.9, polyline AO_1B is part of a polyline lens, which cuts the streamlines and pushes them out of the focus region. Our line model cannot be used to define the movement of the vertices in the red and green regions, because the red/green focus region has completely different shape with the red/green transition region, i.e. one is a triangle and the other is a trapezoid. For example in Fig. 3.10a, the red region with white dotted texture is a triangular focus region; while the red region with checkerboard texture is a trapezoidal transition region. By simply using line model and pushing vertices along the direction perpendicular to the region boundaries, the streamlines in the dots region plus checkerboard region cannot be uniformly fitted in the checkerboard region. To give another example, there is a blue streamline in the red region in Fig. 3.10a and one in the green region in Fig. 3.10b. When they follow the line mode and move along the arrow dash lines, the middle two vertices

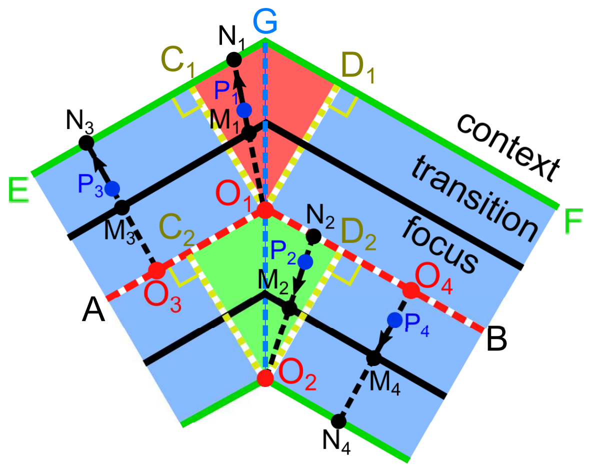


Figure 3.9: Illustration of polyline lens. The red dashed lines, AO_1 and O_1B, are two connected line segments that compose a polyline lens. The region inside the black lines is the focus region. The region between black lines and the green lines are the transition regions. Outside the green lines are the context regions. The deformation regions of two line segments intersect on line GO_2. Draw lines from point O_1 perpendicular to line EG and FG, which intersect at point C_1 and D_1, respectively. Draw lines from point O_2 perpendicular to line AO_1 and BO_1, which intersect at point C_2 and D_2. The deformation region in the entire domain is divided into three regions, marked with red, green and blue background colors. P_1, P_2, P_3 and P_4 are four vertices located in three different color regions. The black arrows show their moving directions during deformation. The extension lines of the directions are the black dash lines, which intersect with the region boundaries at $M_{1,2,3,4}$ and $N_{1,2,3,4}$.

will diverge in the red region and converge in the green region. As a result, the deformed streamline vertices in the red/green transition regions are either too sparse to be smooth (in the red region) or too congested to be visible (in the blue region).

To achieve a smooth and continuous deformation around the joint area of the polyline lens, we want to compress the red region space into its upper transition region, and compress the green region space into its lower transition region. We notice that such compression will cause the vertices to move in a radial direction. For the red region, vertices move away from point O_1; while for the green region, vertices

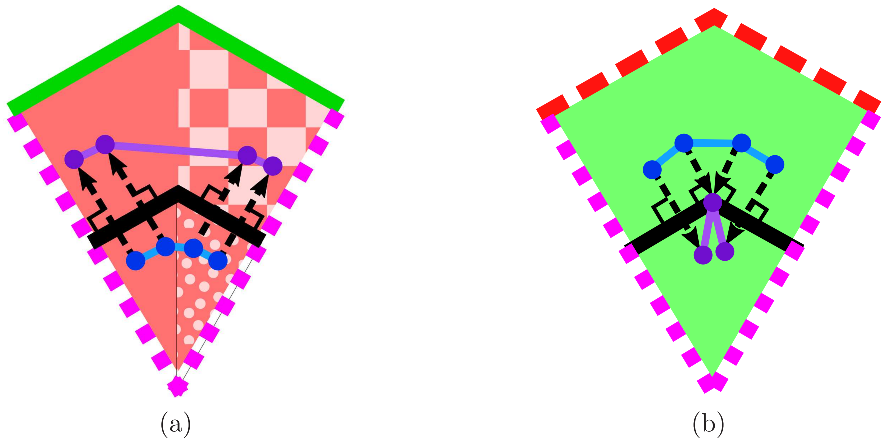


Figure 3.10: Magnified views of the red and green regions in Fig. 3.9. The blue streamlines become the purple streamlines after deformation.

move towards point O_2. Thus, we can apply the point model in the two regions in the joint areas. For the red region, we can directly apply the point model and push vertices away from the center O_1. On the other hand in the green region, instead of pushing, vertices are pulled towards the center O_2.

To determine how far a vertex should move, we still use our displacement function $g(x)$ to compute the vertex's desired distance to the focus region center by Equation (3.4). For example, there are four vertices $P_{1,2,3,4}$ in Fig. 3.9, which are from the regions of three different colors and follow the four black arrows to move.

Vertex P_1 is in the red region and follows the point model. It is pushed away from O_1 until reaching a distance of d_1 from O_1, where

$$d_1 = g\left(\frac{|O_1\vec{P}_1|}{|O_1\vec{N}_1|}\right) \cdot |O_1\vec{N}_1| \tag{3.11}$$

Vertex P_3 is in the blue region and follows the line model. After deformation, its distance to O_3 is

$$d_3 = g\left(\frac{|\vec{O_3P_3}|}{|\vec{O_3N_3}|}\right) \cdot |\vec{O_3N_3}| \tag{3.12}$$

If P_1 and P_3 are very close to each other at the boundary of red and blue region, i.e. line O_1C_1, then we have $|\vec{O_1P_1}| = |\vec{O_3P_3}|$ and $|\vec{O_1N_1}| = |\vec{O_3N_3}|$. From Equations (3.11) and (3.12), we got $d_1 = d_3$. This means that the deformation is continuous around the boundary of the red region and the green region.

Vertex P_2 is a point in the green region and follows the point model. Equivalent to pulling towards O_2, we can think of it as pushed away from N_2 and reach a distance of d_2 from N_2, where

$$d_2 = g\left(\frac{|\vec{N_2P_2}|}{|\vec{O_2N_2}|}\right) \cdot |\vec{O_2N_2}| \tag{3.13}$$

Same as P_3, Vertex P_4 follows the line model and is pushed away from O_4 at a distance of

$$d_4 = g\left(\frac{|\vec{O_4P_4}|}{|\vec{O_4N_4}|}\right) \cdot |\vec{O_4N_4}| \tag{3.14}$$

If P_2 and P_4 are very close to each other around the boundary of the green region and the blue region, i.e. line O_2D_2, then we have $|\vec{N_2P_2}| = |\vec{O_4P_4}|$ and $|\vec{O_2N_2}| = |\vec{O_4N_4}|$. From Equations (3.13) and (3.14), we know that $d_2 = d_4$, which means the deformation around the boundary of the green region and the blue region is continuous.

Because the red region and the green region do not share a boundary, we do not need to verify the deformation continuity between them. Although our polyline lens uses both the point and line models in different deformation regions, its deformation is still continuous at the boundary of the regions using different deformation models.

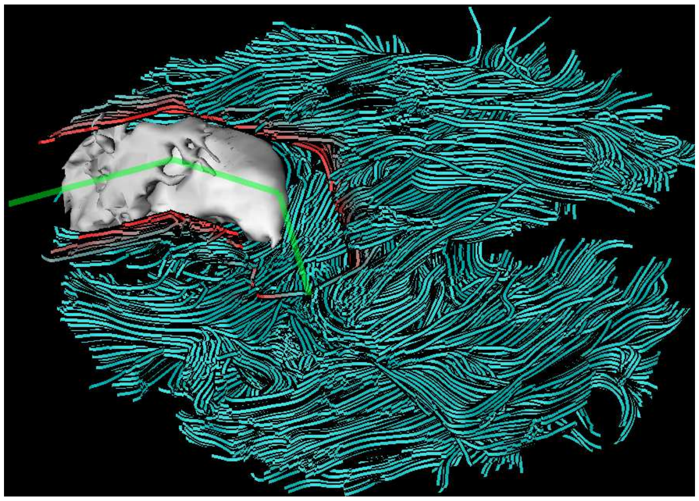


Figure 3.11: Using the polyline lens to open the brain tumor data set.

Fig. 3.11 demonstrates a visualization of the brain tumor in the white matter tracts with the polyline lens. A polyline lens is drawn by the user to make it follow the curvy shape of the tumor. Streamlines around the lens are cut and pushed away from the lens region. The deformed streamlines around the joint of two connected line segments are smoothly deformed and continuously connected to the surrounding deformed streamlines. Compared to the regular ellipse lens and open blind lens, the shape of the polyline lens can closely follow the tumor's shape, which distorts the streamline less and preserves the context better.

3.3.3 Direct-Touch Interaction

Because our deformation model is in screen space, it would be intuitive to interact with the streamlines directly on a direct-touch display, so that users can use the screen as an interaction space and pull away the occluding streamlines with their fingers. It has been reported that touch-based visualization can benefit the users with its ability of direct manipulation and smooth interaction [64]. In our system, users cut and move the streamlines with the interactive 3D lens, which requires users to specify its location and shape. Six degrees of freedom (DOF) need to be supported for the lens: 2D translation on the screen, 1D angle representing the orientation of the lens, 1D scaling factor for the two radii of the ellipse shape lens surface, and the 1D lens depth.

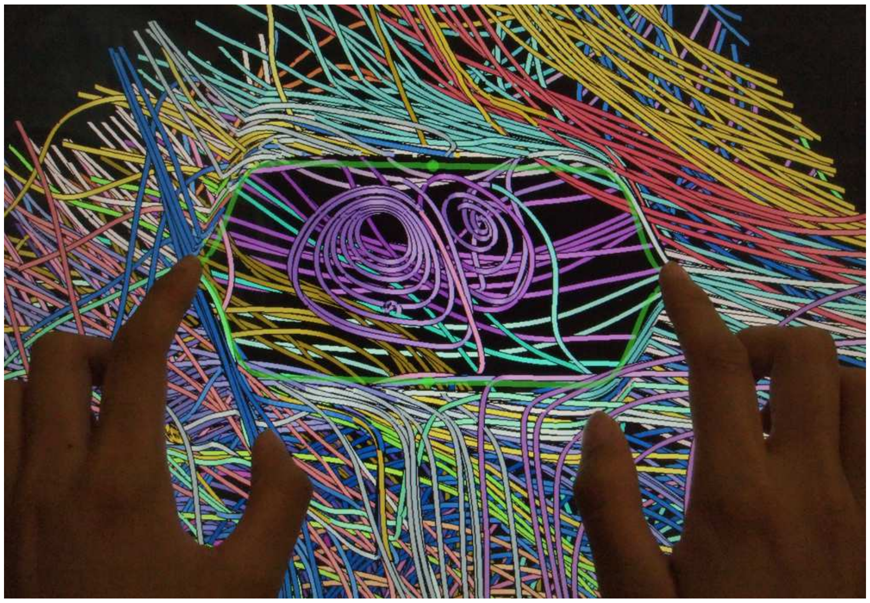


Figure 3.12: Placing an open blade shape lens with two fingers on a multi-touch display.

A traditional way to specify the 6DOF of the lens is through mouse interaction, i.e. adjusting the lens surface screen position and shape with mouse dragging, and adjusting lens depth with mouse wheel scrolling. With a multi-touch display, we can replace all the mouse interaction with more intuitive and efficient direct-touch interaction, as described by the following. With one finger, users can change 1DOF or 2DOF of the lens at a time, i.e. translation or rotation or scaling. When two fingers are placed on the boundary of the lens and move, users can change 4DOF of the lens at the same time. Swiping two fingers changes the 2D screen location of the lens; rotating two fingers changes the 1D lens orientation; and pinching two fingers changes the size of the lens. Pinching out/in two fingers inside the lens allows users to increase/decrease the 1DOF lens depth and simulates pushing away or pulling back the streamlines. Users can also drag one or two fingers on screen to specify a cutting line of an open blade lens. This touch interaction simulates the process of cutting streamlines with finger tips, as shown in Fig. 3.12. The Hurricane Isabel dataset is used in Fig. 3.12. Hurricane Isabel was a strong hurricane in the west Atlantic region in September 2003. The resolution of the dataset is $500 \times 500 \times 100$. An efficient GPU implementation of the streamline cutting process makes the cutting effects responds to the cutting gestures in real-time. The viewpoint navigation is also enabled with multi-touch gestures. The accompanying video demonstrates the interactions with the multi-touch display.

3.4 Case Study

With the proposed deformation algorithm, as well as the interactive 3D lens and direct-touch interface, we design an interactive system for streamline exploration. We perform two case studies using our technique with two types of data: vector field data and tensor field data.

3.4.1 Streamlines From Vector Field

In this section, we provide a case study that uses our system to explore the Solar Plume dataset in three different ways: exploring the streamlines at different depths, from different view directions, and at different locations. In this case study, we provide two ways to define focus streamlines, first as streamline bundles in Section 3.4.1, and second as streamlines passing through a user-specified region in Section 3.4.1. Note that users are free to use other methods to define the focus streamlines depending on their needs. Alternatively, when the 3D lens is used, users are not required to define their focus streamlines but can simply move the lens around the 3D space and search for them. The Solar Plume dataset comes from a simulation of the solar plume on the surface of the Sun. The resolution of the dataset is $126 \times 126 \times 512$. A demonstration of this case study is shown in the accompanying video.

Exploration with Different Depths

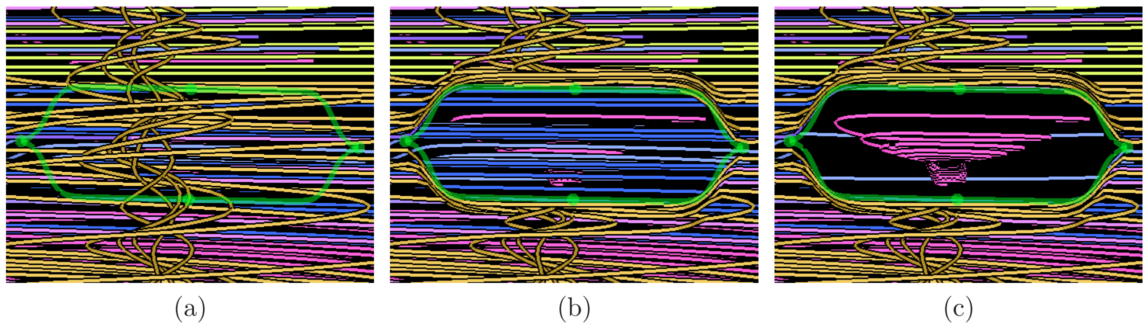
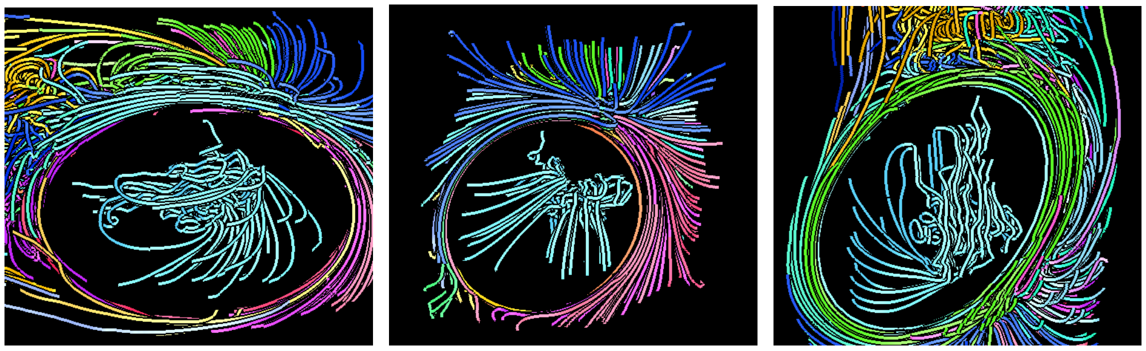


Figure 3.13: Exploring streamlines at different depths by the interactive 3D lens with different lens depths.

Streamlines of different depths may be projected to the same screen location, and those with smaller depths will occlude the ones with larger depths. Our interactive 3D lens can help show the streamlines at all different depths with reduced occlusion. To do this, we can exploit the 3D lens that uses the line deformation model with open blinds to explore the streamlines, as shown in Fig. 3.13. In Fig. 3.13a, initially the lens does not touch any streamlines so no streamline is deformed. We see a vertical vortex-like object close to the surface of the volume. We push the lens into the volume by increasing the lens depth, and then we see the blue horizontal streamlines in the center of the volume, shown in Fig. 3.13b. Finally in Fig. 3.13c, we increase the lens depth even more to remove the straight streamlines and a funnel shaped vortex is revealed in the back of the volume.

Exploration with Different View Directions

A 3D object may look very different from different views; hence it is important to view a 3D feature from different view directions to obtain a complete understanding of its shape. Because our deformation model is view-dependent, we can remove the occlusion regardless of the view direction. Furthermore, users can get different context information from different views.



(a) front (b) left (c) top

Figure 3.14: Views of a bundle from different view directions.

In Fig. 3.14, we view a streamline bundle from three different sides of the volume. A streamline bundle is a cluster of streamlines that are similar in location and shape.

We measure the similarity between the streamlines with the *mean of closest point distance* [22], and cluster the streamlines using hierarchical clustering. The focus region is represented by the convex hull or the minimal enclosing ellipse of the focus streamlines.

This bundle shows a turbulent flow structure with straight tails. From the front view of the bundle in Fig. 3.14a, a yellow vortex is preserved at the upper-left of the figure in the context region. In the left view of the volume (Fig. 3.14b), the contexts appear to be stretched long streamlines, which look very different from the front view. In the top view of the bundle (Fig. 3.14c), we can see some curvy vertical streamlines. The yellow vortex visible in the context of Fig. 3.14a is again visible from this view, which has a more complete shape than that in Fig. 3.14a.

Fig. 3.1 is another example of streamline bundles using the Hurricane Isabel dataset. In the accompanying video, we also use the Isabel dataset to explore the bundle from different view directions.

Exploration with Different Locations

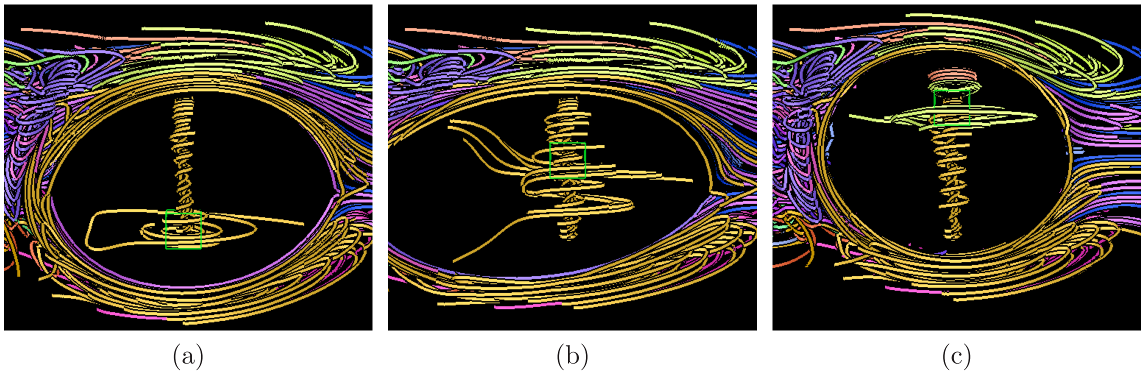


Figure 3.15: Exploring flow features at different locations. The streamline picking cube moves bottom up from (a) to (c).

Users sometimes are interested in the flow behavior in a particular spatial region. In our system, users are allowed to place a small axis-aligned cube in the domain, and then the streamlines passing through this region are selected as the focus streamlines. The minimal enclosing ellipse of the focus streamlines defines the focus region. Here we illustrate the exploration of the streamlines from different locations around a vortex. In Fig. 3.15, the streamlines passing though the selected location indicated by the green cube are shown. In Fig. 3.15a, we place the cube at the bottom of the space. A narrow vortex is selected and shown with some wider vortex-shaped streamlines on the side. This image tells us that the flow passing through the selected region extends to the top and the side of the surrounding area. When we move the cube up, a new set of focus streamlines is selected as shown in Fig. 3.15b. From the tails of the focus streamlines on the left, we know that this selected location is connected to the left side of the flow. We move the cube to the top of the volume, as shown in Fig. 3.15c. As can be seen, the flow behaviors are different on the top and the bottom of the regions. The vortex on the top is located in a small region, while the vortex at the bottom extends to a wider area. Note that the three images in Fig. 3.15 all preserve the context features, such as the purple vortex on the left and the horizontal straight streamlines on the right.

3.4.2 White Matter Tracts from Diffusion Tensor Imaging

Diffusion tensor MRI (DTI) provides directional diffusion information that can be used to estimate the patterns of white matter connectivity in the human brain. Diffusion tensor imaging (DTI) and white matter tract fiber tractography have opened a new noninvasive windows on the white matter connectivity of the human brain. DTI and fiber tractography have advanced the scientific understanding of many neurologic and psychiatric disorders and have been applied clinically for the presurgical mapping of eloquent white matter tracts before intracranial brain tumor resections[85].

The most common technique for visualizing the white matter tracts from DTI uses the major diffusion tensor eigenvector to define the local white matter tract direction[118, 9, 72]. The result is a dense set of tracts, which can be represented as streamlines (or tensorlines[118]).

Exploring Reshaped Tracts

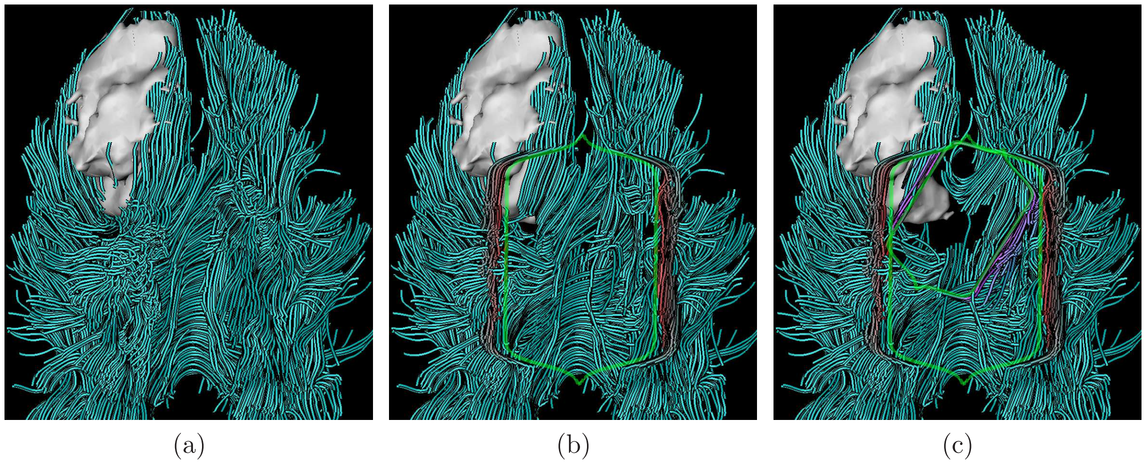


Figure 3.16: Placing the layered lenses on reshaped tracts to see how the tumor influences its surrounding tracts.

A growing brain tumor tends to compress and re-orient white matter tracts into abnormal configurations. As seen in Fig. 3.16a, in the center near the bottom, the tracts are distributed horizontally and connect the left and right halves of the brain. However, the tracts above and to the left of the horizontal tracts are diagonal and not symmetric because they are reshaped by the tumor.

To further explore the tracts near the tumor, in Fig. 3.16b, we place a open blind lens vertically to cut the tracts and move them to the left and right sides. It becomes clearer that the orientations of the tracts change from horizontal (normal) to diagonal from bottom up. We also notice that the tumor has a vertical elongated shape, which is the same orientation of the tracts on the surface of the tumor. This observation supports the finding that the tumor cells move faster along the white matter fiber tracts in the brain [84].

In order to look at the deeper tracts near the tumor and without losing much context information, we place a second smaller layered lens on the top region of the first lens and make the cut following the orientation of the tracts, as shown in Fig. 3.17c. Inside the second lens, we can see the tracts are connected to the right half brain, but not connected to the left half. This may indicate the white matter on the left has been destroyed.

Exploring Tracts Around Tumor

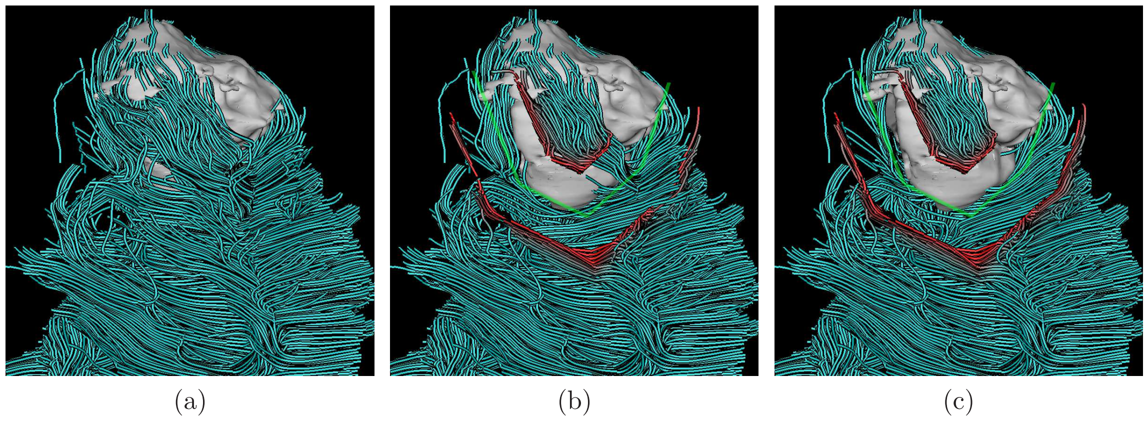


Figure 3.17: Placing a polyline lens around the tumor to inspect the relationship between the tumor and its surrounded tracts.

Because tracts around the tumor are of significant interest to neurosurgeons, we can place a polyline lens around the tumor by following its silhouette, as shown in Fig. 3.17. Before applying the lens as in Fig. 3.17a, tracts are densely distributed above and around the tumor, which makes it difficult to inspect the tracts that are close to the tumor but deeper inside. After placing a polyline lens around the tumor, we create curved band shape focus region, as shown in Fig. 3.17b. When the streamlines on the surface are removed from focus region, we can see that the tracts above the tumor is connected to the tracts on the right side. The tracts on the left side are vertically oriented. Some of them even penetrate through the tumor. After further increasing the lens depth, we see the deeper tracts in Fig. 3.17c. Those tracts at the bottom right of the tumor are relatively smooth and mostly horizontally distributed.

Feedback from Domain Experts

To gain useful, informal feedback, we demonstrated our streamline exploration approach to five neuroscience experts: a biomedical engineer, two neuroradiologists and two neurosurgeons. All experts have used primarily 2D slice- and transparency-based occlusion reduction techniques for tract visualization. They were all very positive about the deformation approach and said that it is "smooth", "very useful" and could be a "terrific tool." One neuroradiologist noted that the 3D environment allowed him to see relationships that he couldn't see with his 2D tools and the biomedical engineer appreciated that the deformation approach gives greater spatial context with less clutter than transparency techniques. Two experts were interested in using the software to explore objects and features deeply embedded beneath tract streamlines for deep brain probing and, interestingly, visualization deep into the kidneys. Additional application suggestions included visualization of tract disruption in trauma cases, intraoperative visualization, tract morphology in cases of Multiple Sclerosis and other neurological disorders, and endoscopy simulation. We based the coloring of deformed streamlines (see Fig. 3.8) from expert feedback, as it gives a clear visual cue of true versus deformed morphology.

3.5 Performance

We measured the performance of our interactive streamline deformation system on a machine running Windows 7 with Intel Core i7 2600 CPU, 16 GB RAM, and an NVIDIA GeForce GTX 560 GPU that has 336 CUDA cores and 2GB of memory. The performance tests were performed in 2014. It can be higher if newer graphics card had been used. Table 3.1 shows the results of two test datasets, Plume and Isabel. In our implementation, the CGAL library [2] is used to extract 2D/3D convex hulls and ellipse-shaped focus areas. To speed up the computation, CUDA and the Thrust library [54] were used to perform the deformation computation.

Four different operations related to our deformation model described previously were tested and the timings were collected. The *Cut* operation is the first step of deformation for the line model (Section 3.2.2), which cuts a streamline into multiple streamlines by a straight line. The *Lens* operation runs in the interactive 3D lens mode (Section 3.3), which searches for the streamlines that intersect the lens, and is done only when the lens or the view is changed. The *Location* operation runs in the streamline selection stage by the location mode (Section 3.4.1), which searches for the streamlines that pass through the cubic region only after the cube location is changed.

The results show that these operations only take a few milliseconds, and they are only executed when some settings are changed. Therefore, they do not affect the overall performance of our algorithm much. The deformation operation is performed at every frame, so its speed is crucial for the interactivity of the system. We measured the deformation computation time and the overall frame rates for the three models described in Section 3.2.2. Note that the overall frame rate was computed from the average time of drawing each frame and is not related to the constant deformation speed. It reflects the speed of our algorithm in all stages, including the CUDA-based deformation operation, data transfer, and coordinates transformation. The point model with the ellipse focus takes the shortest deformation time and has the highest frame rate, while the line model is slightly slower, but highly interactive. They are both suitable for real-time performance. The point model with the convex polygon as its focus area is much slower for deformation and has a comparatively lower frame rate because both the shapes of the ellipse and the open blinds have an analytical representation, but the convex polygon is represented by a point set. Although this model is comparatively slower, it is still moderately interactive in our implementation running at at least 30 frame per second (FPS). We also measured the performances of the three layered lenses with line model and the polyline lens composed of 6 line segments. Their speeds are slightly lower than the basic open blind lens, but still high enough to guarantee the interactivity of the system. Finally, we can also see that the Plume dataset has a higher frame rate than the Isabel dataset because the deformation operates on each vertex and the Plume dataset has fewer vertices.

Table 3.1: Performance test results.

Dataset		Plume	Isabel
Count	Streamlines	921	609
	Total vertices	317,814	522,436
Operation time for preparing deformation inputs (ms)	Cut	22	36
	Lens	6	9
	Location	7	7
Deformation time (ms)	Ellipse(point model)	0.67	1.08
	Polygon(point model)	13.8	25.1
	Blinds(line model)	0.81	1.27
	3 layered lens(line model)	0.783	1.44
	Polyline lens(6 segments)	1.53	1.83
Frame Rate (FPS)	Ellipse(point model)	536	347
	Polygon(point model)	58.3	35.4
	Blinds(line model)	483	293
	3 layered lens(line model)	226	147
	Polyline lens(6 segments)	209	144

3.6 Conclusion

We have presented a streamline deformation technique to achieve occlusion-free F+C streamline visualization, by displacing the occluding streamline vertices in screen space. Our deformation model has the following advantages:

- Creates an occlusion-free view from arbitrary view directions,
- Preserves shapes of the deformed streamlines,
- Provides smooth transition when distorting the deformed streamlines,
- And provides interactive performance.

In this chapter, we describe the deformation model and its two variations regarding the shape model used in deformation, the point model and the line model. To allow users to freely explore the flow field without prior knowledge, our system provides an interactive 3D lens and direct-touch control to move away the streamlines in user-specified regions in screen space at a given depth. To satisfy different user requirements, we provided the layered lenses and the polyline lens that explore the features in different layers and the features of more complex shapes. We develop an interactive streamline exploration system based on our deformation model and demonstrate our system through a case study using the Plume dataset and the brain diffusion tensor dataset with experts feedback. Our deformation algorithm is easy to parallelize and can achieve high performance using GPUs, and thus can be used for interactive exploration of flow datasets. We believe our interactive streamline exploration approach can help the CFD scientists and engineers and the neuroscientists to freely explore the data, and also help students to learn about flows and the brain.

One limitation of our deformation model is that some deformed streamlines close to the center of the focus region may still get significant distortion, so the original shapes of these streamlines are not well preserved. The focus region should be relatively small and local to the feature of interest to prevent too much distortion in the context. Finally, when the viewpoint is changed abruptly, the transition of deformation may not always be smooth, so sometimes users need to restart the deformation for the new viewpoint.

Chapter 4: GlyphLens: View-dependent Occlusion Management in the Interactive Glyph Visualization

4.1 Introduction

While viewing glyph-based visualization, users want to focus on interesting features that are usually represented by one or a cluster of glyphs called *focus glyph(s)*. The glyphs around the focus glyphs in 3D space are called *context glyphs* that provide contextual information, such as the focus glyphs' relative spatial properties. Our goal is to combine occlusion management and focus+context(f+c) so that we can visualize both focus glyphs and their surrounding context glyphs in the same image without any occlusion. There are various 3D occlusion management techniques. Elmqvist and Tsigas [32] summarized them into a few categories including volumetric probes, virtual X-ray, projection distorters, and etc.

In this work, we propose a view-dependent lens called *GlyphLens* to interactively explore glyphs in 3D space. The GlyphLens uses space deformation techniques to open a tunnel of various sizes from the viewer's eyes to reveal the focus glyphs. The context glyphs are displaced away from the lens, so the viewer can look at the focus glyphs through the tunnel without occlusions. Besides, the lens attenuates the brightness of the context glyphs based on the depths to provide additional depth cue as well as highlighting the focus glyphs. We use animation to depict the displacement and attenuation of the glyphs, so users can mentally recover the original positions and appearances of the glyphs through the transitions. Using this lens, users not only can remove the occlusion from arbitrary view directions, but also preserve the shape and relative positions of the context glyphs surrounding the focus glyphs. The GlyphLens can either deform screen space or object space. Deforming screen space completely removes occlusion, while deforming object space can preserve the spatial structure of the deformed glyphs. We provide two shape models for the lens to fit different focus feature shapes and to displace glyphs in different directions. To explore the power of the GlyphLens, we implemented an interactive glyph visualization system using the lens. The system provides a few lens utilities for accurate lens placement and snapping. To view a group of focus glyphs from arbitrary view directions, users are allowed to select their focus glyphs or predefined features to be always visible. We implemented our system using different input devices, such as mouse, touch screen and motion camera, for placing and manipulating the lens. We also explore two output devices, screen monitor and virtual reality headset, for displaying our glyphs and assisting on the view-dependent lens interaction. We provide three case studies to demonstrate the effectiveness of different glyph-based visualizations using our glyph lens and collected user feedback from two domain scientists.

4.2 Algorithm

To design an effective occlusion-free f+c lens for visualizaing 3D glyph, we have three design goals as follows:

- Remove occlusion completely. The lens can create an occlusion-free view to visualize the originally occluded features.
- Preserve contexts. Glyphs around interesting features should also be visible as the contexts. Besides, their relative locations or spatial structures should be preserved.
- Provide depth cue. Because the glyphs at different depths overlapping with each other on the screen, a clear depth cue can help identify the locations of the focus features.

In this section, we first give an overview of our algorithm in Section 4.2.1. Because glyph displacement is the major contribution of our lens, we explain how to displace context glyphs using two space deformation models in Section 4.2.2 and two lens shape models in Section 4.2.3.

4.2.1 Algorithm Overview

We use Fig. 4.1 to illustrate our problem and solutions. In the figure, the red glyphs are the focus glyphs that are of user's interest, while the blue glyphs are the context glyphs. The glyphs around the focus glyphs are the important context glyphs, e.g. the blue glyphs inside the red dashed circle, because they provide spatial reference for the focus glyphs. Some blue glyphs, such as the glyphs marked as A and B in the figure, occlude users' views to the focus glyphs. Our view-dependent GlyphLens is composed of two coaxial cylinders oriented from the viewer's eye to the focus glyphs. The double cylinder lens has two regions: *focus region* and *context region*, as shown in Fig. 4.1a. Inside the inner cylinder is the focus region, because the focus glyphs are right behind it. Between the two cylinders is the context region. In order to see the red focus glyphs, our lens displaces the blue context glyphs from the focus region into the context region. With an empty focus region after displacement as shown in Fig. 4.1b, viewers can see the red focus glyphs behind the lens through the empty focus region without any occlusion. In Fig. 4.1b, the glyphes A and B are moved to the context region and do not occlude the focus glyphs. The glyphs outside the lens are not displaced such as the glyph C and D. The glyph displacement is achieved by deforming the space. We can either deform screen space as described in Section 4.2.2 or deform object space as described in Section 4.2.2 to preserve different kinds of context features. Because the focus glyphs or the features can form different shapes or spatial structures, the Section 4.2.3 describes how to design different shapes of cylinder bases to fit different feature shapes. Users can interactively move the back base of the lens to be in front of different glyphs, so that any glyphs right behind the lens back base become the new focus glyphs and can be viewed without occlusion.

As we know, occlusion is an important depth cue for 3D rendering. However, in the glyph visualization, there are a lot of empty spaces between the glyphs so that the viewers can easily see through the empty spaces to see the glyphs at different depths without much depth cue. Therefore, we use brightness difference to provide extra depth cue for the glyphs behind the lens. The brightness of a glyph decreases with the glyph's distance to the lens back base as shown in Fig. 4.1b. To attenuate a glyph's brightness, we multiply its color RGB channel by an attenuation coefficient a defined as

$$a = max\{(s \times z + 1)^{-1}, a_0\} \tag{4.1}$$

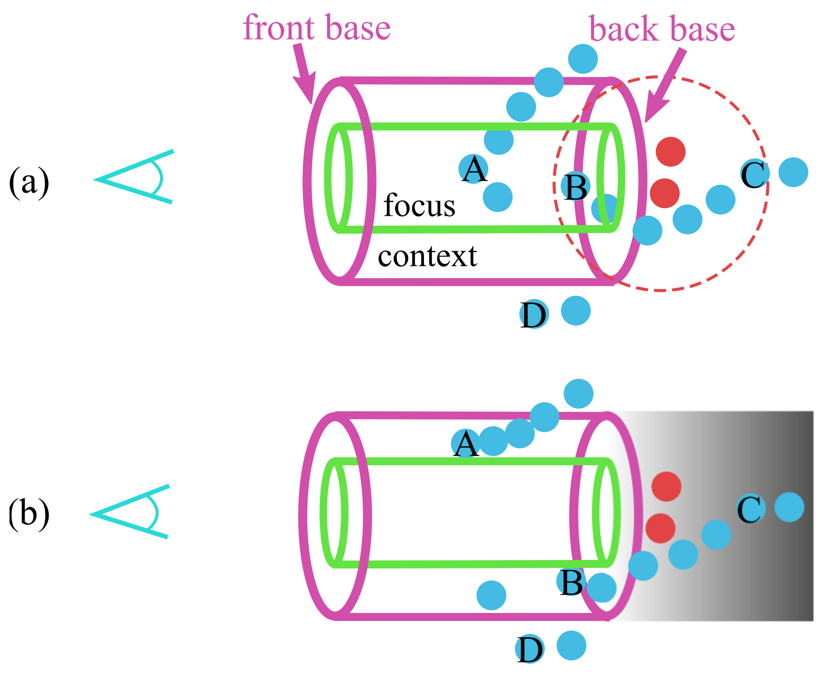


Figure 4.1: Illustration of how to displace the glyphs from (a) to (b). The red and blue circles represent the glyphs. The red glyphs are the focus glyphs. The green and pink cylinders define our lens and divide the space inside the lens into two regions: focus region and context region.

where z is the glyph's distance to the lens back base, s is an adjustable constant coefficient to control the speed of attenuation, and a_0 is the minimum attenuation coefficient to prevent the glyphs from becoming invisible. An empirical value of a_0 is 0.1.

4.2.2 Space Deformation Model

We provide two space deformation models, screen space deformation and object space deformation, to displace the glyphs. The first model directly changes the glyphs' positions based on their screen coordinates and completely removes occlusion. The second model uses a tetrahedral mesh as the object space proxy geometry and deform the mesh based on a physically-based model. The glyphs are indirectly moved along with the deformed mesh that puts constraints on the glyph movement and can better preserve the spatial structures of the glyph clusters.

Screen Space Deformation

The screen space deformation is defined in 3D screen space similar to the one used in the streamline deformation work by Tong *et al.* [107]. It is 3D because we add the z coordinate or the depth from the clip space to make it a 3D space, shown as the blue frame in Fig. 4.2d. Our lens is a cylinder in screen space, or a frustum in the camera space as shown in Fig. 4.2a. We define the shape of cylinder base using its x and y coordinates in screen space and define its visual transformation, such as displacement and attenuation, based on its z coordinate. Fig. 4.2c is the frontal view of 3D screen space that illustrates how the lens and its two regions look from 2D screen. Using this coordinate system, users can easily change the size and shape of the lens base on the screen to fit different shapes of focus glyph clusters, and change the lens position to point to different focus glyphs in different regions.

As described in the algorithm overview, we need to displace the glyphs inside the lens out of the focus region and preserve the glyphs surrounding the focus glyphs as contexts. We divide the cylinder lens region into two halves, front part and back part, based on the depth. The boundary of the two halves is illustrated as the black dashed circle in Fig. 4.2d. The total depth of the lens is d, and the back part of the lens has a depth of Δd. Fig. 4.2b shows a cross section of the Fig. 4.2d. The horizontal coordinate represents the depth direction and the vertical coordinate represents the screen height. We displace the glyphs in the two parts of the lens differently. In the back part of the lens, the space in both the focus region and the context region

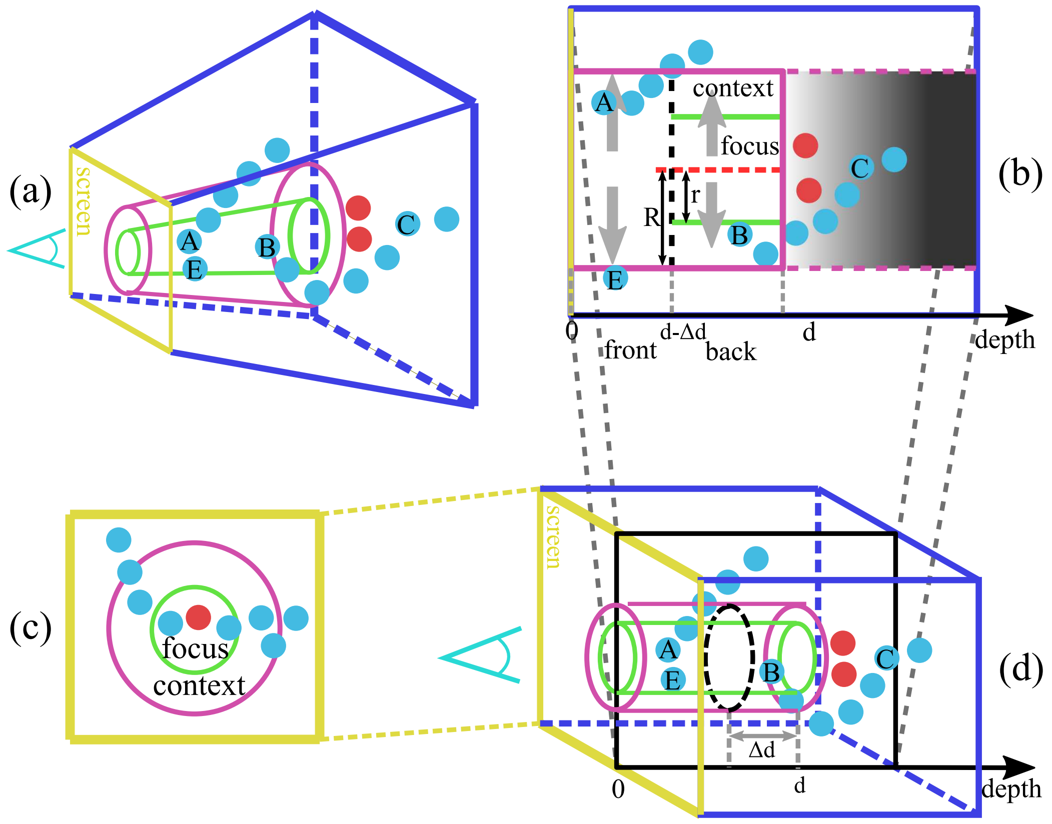


Figure 4.2: Illustrations of the screen space deformation model in camera space (a) and screen space (d). (b) is a cross section of (d). (c) is a 2D frontal view of screen space. The black dashed circle in (b) and (d) divide the lens into front part and back part. (a) (c) and (d) show the glyphs before the displacement, while (b) shows the glyphs after the displacement.

is linearly compressed into the context region. To displace a glyph whose distance to the lens center line is x, it moves towards the direction pointing away from the lens center until its distance to the lens center is $x \cdot \frac{R-r}{R} + r$, where r and R are the distances from the lens center to the inner and outer cylinders, respectively. As for glyphs in the front part of the lens, they are not close enough to the focus glyphs to be treated as contexts and could possibly occlude the displaced glyphs in the context region in the back part of the lens. So we displace the glyphs in the front part of the lens to the outer lens boundary. This means they are almost cut away from the image. For example, if we displace both the glyph B and E into the context region, then glyph E may still occlude users from viewing glyph B. We decide to displace glyph E to the outer boundary of the context region, then it will not cause occlusion to glyph B. After the deformation as shown in the Fig. 4.2b, we not only can look at the red focus glyph without any occlusion, but also see all the glyphs originally around the red focus glyphs as contexts, and hence provide a f+c visualization.

Object Space Deformation

Even though screen space deformation can completely remove the occlusion, the lens may destroy the spatial relationship of the displaced glyphs. For example, in Fig. 4.3, the blue glyphs form two curved clusters in front of the red glyphs. If we use the lens with the screen space deformation model to view the red focus glyphs, the lens will break at least one of the blue clusters whose spatial structures could be important context features. To preserve the spatial structure of the glyphs, we propose an object space deformation model for displacing glyphs. Instead of displacing an glyph in image space, we deform the underlying 3D object space and map the glyph positions to the deformed space. To preserve the spatial structure, we want the regions with a large number of context glyphs less deformable or rigid to keep the glyphs' relative positions. On the other hand, we want the region with fewer glyphs more deformable or elastic, so that this region can either be stretched to create a big void region to see through or be compressed to make space for other rigid regions.

Instead of being a frustum, the lens is a 3D cylinder shape in object space as shown in Fig. 4.3. The lens is always parallel to the view direction and pointing to the camera. To view the red focus glyphs through the lens without occlusion, the

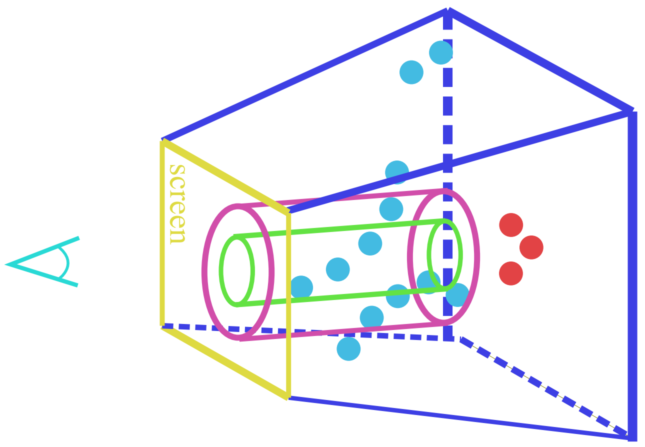
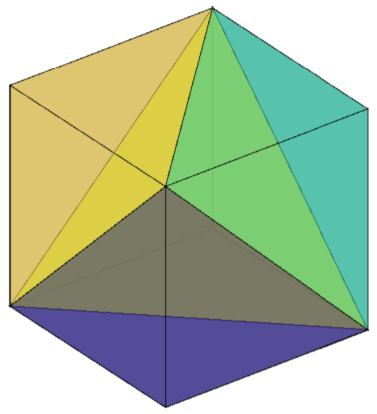


Figure 4.3: In the object space deformation model, the GlyphLens is a cylinder shape in object space, which is not a frustum.

lens needs to push the blue context glyphs out of the focus region into the context region without breaking the two blue glyph clusters.

To simulate the space deformation, we model the 3D volumetric data space using an elastic tetrahedral mesh, shown as Fig. 4.4, and deform the mesh based on physically-based deformable models [87]. Some previous researches [43, 41] also deform tetrahedral meshes built from volumetric dataset for volume rendering. To build a tetrahedral mesh for an arbitrary 3D volume, we first fit a rectangular volume to the target volume and decompose the rectangular volume into stacked cubes as shown in Fig. 4.4b. Then, each cube can be decomposed into 5 tetrahedra as shown in Fig. 4.4a. On each adjacent cube face, the triangles on the two sides should match to make the tetrahedral mesh a manifold. The glyph positions are mapped to the tetrahedral mesh using the barycentric coordinates within each tetrahedron. Then, the glyphs can be displaced by following the mapping with the deformed tetrahedral mesh.


(a) 5 tetrahedra.

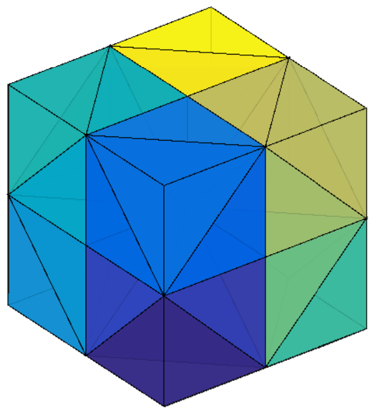

(b) 40 tetrahedra.

Figure 4.4: (a) A cube can be decomposed into 5 tetrahedra. (b) Tetrahedralization of a volume is made by stacking such cubes with correct orientations.

We used the Chebyshev semi-iterative approach in projective and position-based dynamics to solve our elastic tetrahedral model [114], because it supports parallel computation using GPU that can make our lens operation interactive. In the elastic model, the stiffness of the tetrahedron is a variable to measure its ability to deform. In our application, we determine the stiffness of a tetrahedron by the number of glyphs inside the tetrahedron. The tetrahedra containing more glyphs are more rigid or less elastic, while the tetrahedra containing fewer glyphs are more elastic or less rigid. As long as the tetrahedron size we use is smaller than a glyphs cluster's size, we can then model the glyph cluster with a set of rigid tetrahedra. As a result, the glyph spatial relationships in this cluster will not be changed during the deformation. We define the stiffness of a tetrahedron as:

$$g = g_0 + h \times \left(\frac{n}{N}\right)^2; \tag{4.2}$$

where g_0 is the minimum stiffness of a tetrahedron, n is the number of glyphs in this tetrahedron, N is the total number of glyphs in the dataset, and h is an adjustable constant coefficient. In this equation, the stiffness grows quadratically with the number of glyphs. To use the lens to control the deformation, we apply the force F to the mesh vertex V inside the lens and push it away from the lens center shown as the red dashed line in Fig. 4.5a. The force F, shown as the gray arrows in Fig. 4.5b, can be calculated by

$$F = \begin{cases} c \times (R - r) \times \frac{\vec{OV}}{|OV|}, & \text{if } |OV| \leq r \\ c \times (R - |OV|) \times \frac{\vec{OV}}{|OV|}, & \text{if } r < |OV| \leq R \\ \vec{0}, & \text{if } |OV| > R \end{cases}$$

where O is the vertex V's nearest point on the lens center, r and R are the distances from the lens center to the inner and outer lens boundary, and c is an adjustable constant coefficient. Using this equation, the vertices in the focus region receive constant force. For the vertices in the context region, its force reduces as its distance to the lens center increases. The mesh outside the lens boundary, shown as the pink frame in Fig. 4.5, is not deformable in order to restrict the distortion in a local region. Then, values of g and F are used as the inputs to the elastic model simulation.

Fig. 4.5 illustrates the object space deformation model of our lens in 2D as an example. The 2D space is modeled as a triangle mesh. The red glyphs are the interesting glyphs that are occluded by two clusters of blue glyphs. To remove the occlusion without breaking the spatial structures of the blue clusters, we place a green lens in front of the red glyphs. Then the lens deforms the triangle mesh in Fig. 4.5a

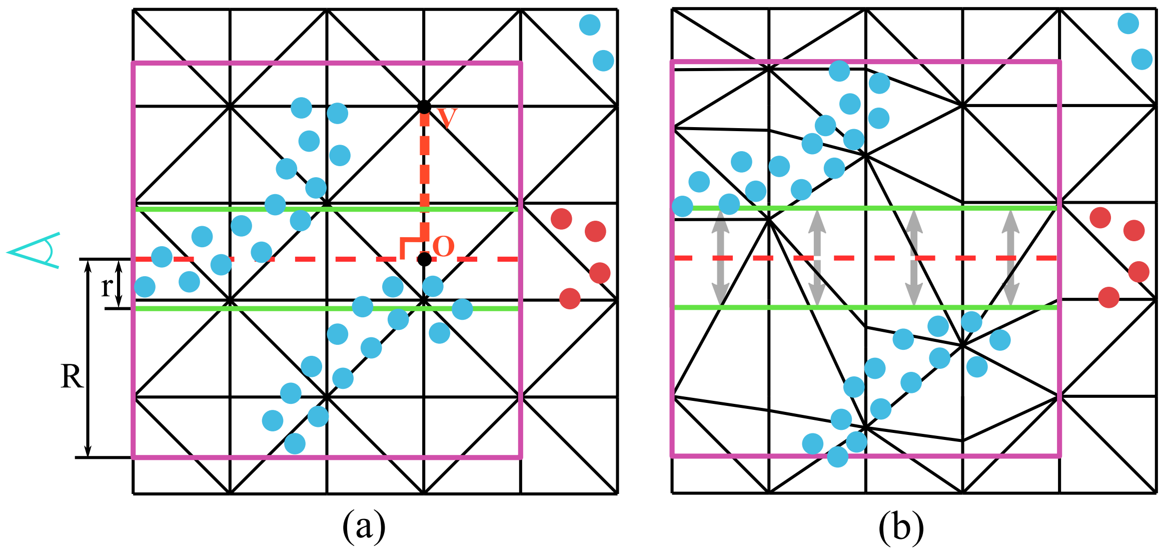


Figure 4.5: As an example in 2D, the tetrahedral mesh reduces to a triangle mesh. The triangle grids represent the object space before (a) and after deformation (b).

into the mesh in Fig. 4.5b. During the deformation, the triangles between the two blue clusters are stretched, while the triangles at the top and bottom of the space are compressed. These deformed triangles are more elastic because they mostly represent empty space that contains few glyphs. On the other hand, the triangles containing the blue glyphs are not deformed much but only moved to the side. In the end as shown in Fig. 4.5b, the viewer can see through the green lens to see the red glyphs without occlusion. Besides, the curved shapes of the two blue glyph clusters are well preserved as contexts.

Even though the object space deformation model is good at preserving the spatial structure within the displaced glyph cluster, it has limitations as well. This model becomes not effective when the glyphs do not form clusters or there is no void spaces between the clusters. This model does not guarantee to create an occlusion-free focus region because the deformed space is still continuous. When glyphs scatter throughout the whole space, the lens cannot sufficiently stretch or compress certain tetrahedra because all the tetrahedra have similar elasticity.

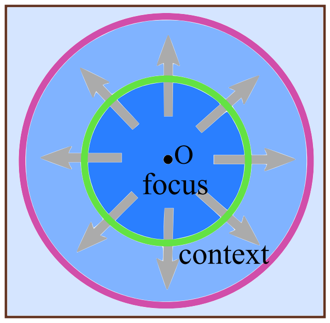
We can decide whether the given glyphs in a dataset are suitable for our object space deformation model. The criterion is that the glyphs need to have spatial structures, i.e. to form clusters. We use the histogram of the glyph locations X to measure the glyph distribution and use the entropy $H(X)$ of histogram as a metric to measure the existence of spatial structures. Each tetrahedron cell k is treated as a histogram bin whose bin counts n_k is the number of glyphs in this cell. The probability of each bin is $p_k = n_k/N$, where N is the total number of glyphs. Then we compute the entropy value using the following equation [52]:

$$H(X) = -\sum_{k=1}^{m} p_k \; log(p_k) \tag{4.3}$$

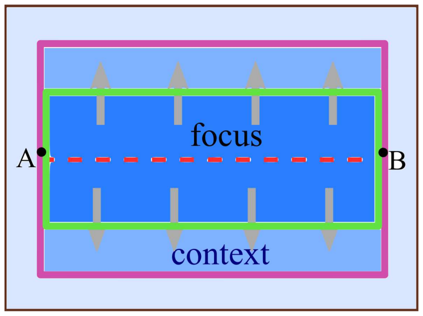
where m is the total number of tetrahedra. Only when the entropy is lower than a threshold, there exists spatial structures in the data, so we can use the object space deformation model. Otherwise, we just use the screen space deformation model mentioned in Section 4.2.2 to remove occlusion more directly.

4.2.3 Lens Shape Models

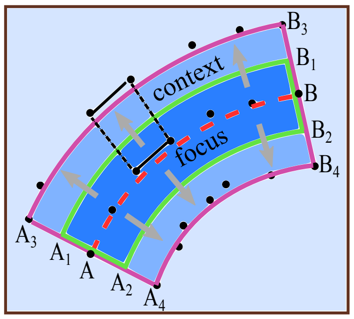
Although both our space deformation models deform the space in the directions parallel to the screen, we need to define the directions and amounts of deformation on the plane of the 2D cylinder base. In this work, we propose two basic lens shapes, *round lens* and *band lens*, to deform the space. Those two shapes are simple enough to minimize the distortion of the space. It is easy to implement and provides real-time performance and convenient user-interaction. The optimal shape of the lens depends on the distribution of the focus glyphs on the screen and the choice of the space deformation models. Besides, our basic lens shapes have the potential to extend to other more complex shapes.



(a) Round lens with circle boundary



(b) Band lens with straight line center



(c) Band lens with curve center

Figure 4.6: Different shapes of the lens screen projection or the cylinder base. In (c), the black points are the control points of the curves.

Round Lens

The round lens provides a circular shape focus region. For example, in Fig. 4.6a, the focus region is represented by the inner green circle. The context region is a concentric, but larger circular region outside of the focus region. It outer boundary is shown as the pink circle in Fig. 4.6a. The center point of the round lens is the circle center O. The space deformation direction and the glyph displacement direction is pointing away from the center O, shown as the gray arrows in Fig. 4.6a. The goal of the space deformation is to move the glyphs in the focus region into the context region.

Ellipse and other near circular shapes can also be used as the boundary shape of the focus region. However the amounts of distortion are not the same in the different deformation directions because of different radii in different directions. So using a circle as the boundary shape of the circular lens is preferred over other shapes such as ellipse. If using the screen space deformation model, the round lens will create large distortions around the point O, because space at this point is deformed into a much bigger green circle. Then, the features originally located at the point O can be distorted more. Therefore, for the round lens, the object space deformation model is preferred, because it generates smaller space distortions near the point O than the screen space deformation model.

Band Lens

The band lens defines a band shape region as the focus region, whose boundary is shown as the green lines in Fig. 4.6b. Instead of deforming the space outward from a point, the band lens deforms the space away from its center line which could be a line or a curve depending on whether the band is straight or not. In Fig. 4.6b, the center line of the band is a straight line AB. The space deformation direction is perpendicular to line AB and away from it. Compared to the round lens, the band lens with a straight line center provides a more uniform space distortion, because the void focus region is opened from a line that is larger than a point. Therefore, the band lens can be easily used with the screen space deformation model and completely remove the occlusion.

In addition to the straight line center, we also defined the band lens whose center line is a curve that can be drawn arbitrarily by the users, as the red dashed line shown in Fig. 4.6c. Drawing a curve on the screen gives users the maximum flexibility of fitting the lens shape to the different feature shapes with a slightly higher computation cost. The users only need to draw a curve on the screen along the feature and specify the width of the band. In order to make the curve smoother and filter out the hand shaking artifacts from the user's hand drawing, the curve is fitted by a Bézier curve using a sequence of control points shown as the black dots in Fig. 4.6c. We can compute the boundary of the context region using the offset of the center curve. The offset of a curve is another curve that is generated by moving the vertices on the original curve along its local curve normal or negative normal for a fixed distance. We use Tiller and Hanson's method [106] to compute the offset curve. The two curves A_1B_1 and A_2B_2 shown in Fig. 4.6c are the two offsets on both sides of the center curve using the given band width as the offset distance. Connecting the end points of these two offsets,A_1A_2 and B_1B_2, can form the focus region. The two curves A_3B_3 and A_4B_4 are the context region boundaries that can be computed in the same way. In this way, each control point of the four offset curves have an one-to-one correspondence with the control point on the center curve, as the black dashed lines shown in Fig. 4.6c. Connecting the corresponding control points between the center curve AB and the offset curves, A_3B_3 and A_4B_4, and the adjacent control points on these curves, as the black lines, can divide the band lens into multiple quadrilateral regions. Then, the space in each quadrilateral is deformed by a sheared version of the displacement using the band lens with a straight center line.

4.3 Interactive Visualization System

Using the proposed lens, we provide an interactive glyph visualization system. To fully explore the power of our interactive lens in the visualization system, we use

Table 4.1: Interactions techniques.

operations	mouse	touch screen	motion camera
rotate	drag left button	drag one finger	rotate head
zoom	scroll wheel	pinch in/out	move head
pan	drag right button	drag with two fingers	move head
change location	drag lens center	drag lens center	move a fingertip
change depth	scroll wheel in lens	pinch inside focus	move a fingertip
change size	drag lens boundary	drag lens boundary	move two hands
change lens shape	draw with left button	draw with one finger	n/a

different input devices, such as mouse, touch screen and motion camera, to manipulate the lens. Besides, we display our glyphs and lens on two different display devices, monitor and virtual reality headset, to study how the display devices can help the interactions of different input devices. We also designed some software utilities to help users interact with user-interested features. Our accompanying video demonstrates the interactive visualization system.

4.3.1 Input Devices

To interact with the glyph visualization, we have two sets of operations to perform. The first set of operations manipulates the 3D transformation of the rendered model, such as rotating, zooming, and panning. The second set of operations controls the parameters of the lens, such as screen location, size, shape and depth. We list how the three input devices are used to perform these operations in Table 4.1 and analyze their advantages and disadvantages as follows.

Mouse The mouse is capable of performing all the operations and is the most familiar device to most users. It requires the least efforts because users only need to slightly move their fingers and arms to control the mouses. The mouse wheel only measures the relative value changes, but it cannot specify an absolute value. So scrolling the mouse wheel can only indirectly increase or decrease the zooming level or the lens depth, but not directly specify their values. On the other hand, the mouse can directly and accurately give the absolute values of the screen position to perform all the other operations. However, users have to mentally map the mouse interaction space (i.e. desktop) to the display space (i.e. monitor screen).

Touch Screen Similar to the mouse, the touch screen can also be used to accurately specify absolute screen positions. Besides, users directly interact with the display space, which makes specifying screen positions more intuitive. Furthermore, touch screens provide intuitive gestures. For example, pinching gestures are more intuitive when controlling the zooming and lens depth than mouse wheels. The only disadvantage of touch screen interactions is that it requires more users' efforts to move fingers over a big screen. Fig. 4.7 shows how the user specifies both the location and the size of a round lens at the same time on a touch screen using two fingers.

Motion Camera Motion cameras, such as Leap Motion [117] and Intel RealSense, provide 3D positions and hand gestures of fingers that can be used to interact with computers. In this work, we used the Leap Motion hand motion camera. As shown in Fig. 4.8, we use the 3D positions detected by the camera to specify the lens location in the object space. By using two hands, the distance between two index fingertips is used to specify the lens size. In this way, we can efficiently control all the lens parameters at the same time as if the lens is in our hands and our hands move the glyphs in the 3D data space. The disadvantage of the motion camera is that

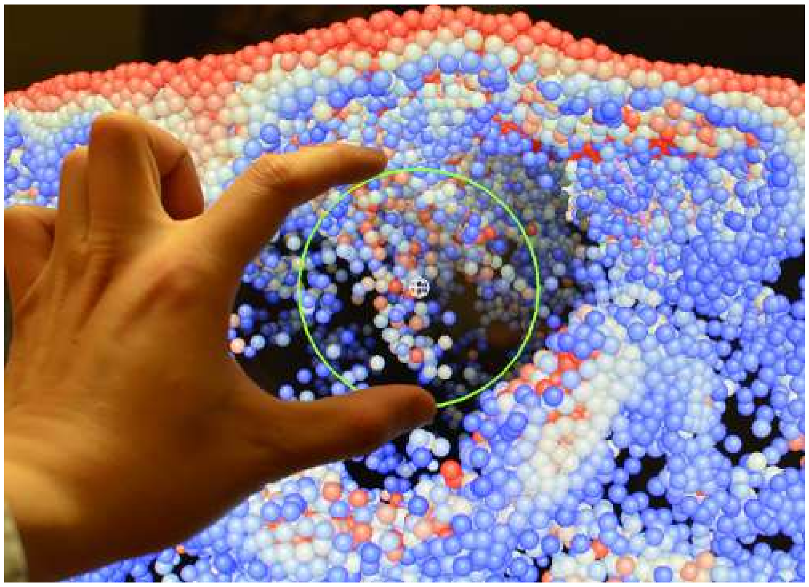


Figure 4.7: A user uses touch gestures to control the lens and explore the particle dataset.

it also captures hand shakes that may make the lens position unstable. But the instability can be alleviated by stabilization algorithms that average a sequence of finger positions.

4.3.2 Display Devices

Besides the traditional monitor screen, we also explore the use of virtual reality headsets to display our glyphs as stereoscopic images. Different display devices can be effective only with specific input devices, but not all of the three mentioned above.

Monitor The monitor displays one image that is a perspective 2D projection of 3D glyphs. Because the viewer's two eyes see the same image, the image looks flat and lacks the sense of distance to the viewer. In this study, we used a regular 23 inches LCD monitor. All of our three input devices work with the monitor. The touch screen works the best with the monitor because the interaction space and the display space are unified. On the other hand, if the users want an effective method to specify the 3D lens position, they can use the motion camera by placing it in front of the monitor.

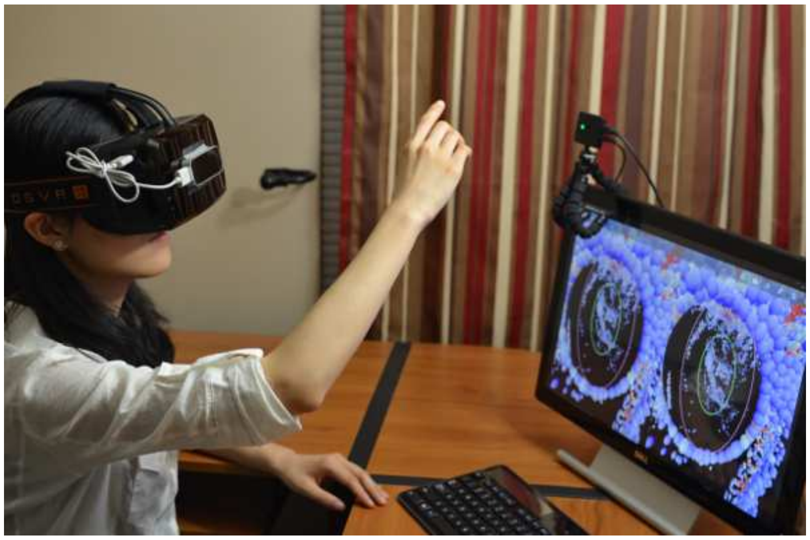


Figure 4.8: The user views stereoscopic glyph visualization in virtual reality headset and controls the GlyphLens by the motion camera mounted on the headset. Head tracking through the camera attached on the monitor is used to change the camera position.

Virtual Reality Headset The virtual reality headset creates an immersive 3D environment by displaying two different images side by side for a viewer's two eyes. In this study, we use the Open-Source Virtual Reality (OSVR) Hacker Dev Kit 1.3 as our headset shown in Figure 4.8. The head tracking device coming with the headset has six degrees of freedom (3D position and orientation) and can be used to control our camera. We used two different modelview matrices to render two different images for the two eyes. In the screen space deformation model, we compute the deformation based on one of the two images without noticing the artifacts. The stereoscopic display and the head tracking simulate a physical presence of users in the glyph visualization space, so it provides users a better sense of 3D shapes and positions of the glyphs. Using the headset, users cannot see the outside environment, so mouses and touch screens become ineffective and only motion cameras can be used. By combining the virtual reality headset and the motion camera, users can see the glyphs and the interactive lens in the same space, so the display space and the interaction space are unified. A disadvantage of the virtual reality headset for visualizing scientific datasets is that users may lose the spatial reference and don't know where they look at when glyphs are displayed too close to them.

4.3.3 Utilities

In the visualization system, we not only implemented the basic user interactions to control the lens position and shape, but also designed a few utilities to allow more efficient glyph exploration.

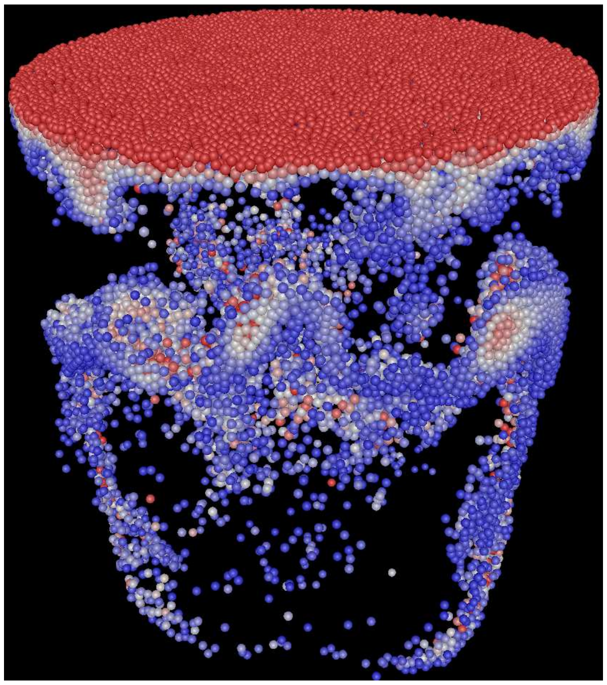
Feature Locking A feature of interest in a dataset can be a set of glyphs with particular properties depending on users' interests. Because the features are what the users focus on, we can lock them in their original positions, so they cannot be displaced by the lens. Then, the users can freely examine the features and their surrounding glyphs without worrying about losing the features.

Glyph/Feature Picking A lens can be moved to a desired position by adjusting its position manually. But sometimes users may want to directly move a lens to be centered at a glyph or a feature that they see or partly see. In these cases, we provide a function to allow users to pick a glyph or feature by pointing on it. Then the lens will automatically center on the picked glyph or feature, i.e. the center of the cylinder lens back base overlaps with the position of this glyph or feature. The picked glyph or feature will be fixed inside the lens focus region and will not be displaced. The lens always centers on the picked glyph or feature even when users change view directions.

Glyph/Feature Snapping It is not always easy to pick a small glyph or feature on a 2D screen or using hand motion camera. Instead of having to point at the exact position of a glyph or a feature, we can let the cursor automatically find the nearest glyph or feature from the lens. This utility not only helps users quickly move the lens onto their target glyphs/features, but also helps stabilize the unstable lens position caused by hand shaking when using hand motion cameras. The nearest glyph search computation is implemented in GPUs by simply comparing the lens's distances to all the glyphs. It achieves a real-time performance even with large number of glyphs.

4.4 Case Studies

We provide three case studies to demonstrate our GlyphLens using two different datasets. We compare the two deformation models in Section 4.4.1 and the two shape models in Section 4.4.2. We also demonstrate how to lock features in Section 4.4.3. More demonstrations of our interactive visualization system using different display and interaction techniques can be found in our accompanying video.


(a) Particle dataset.

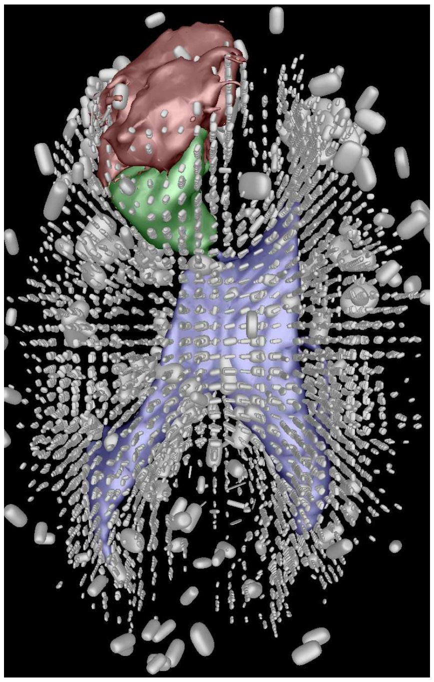
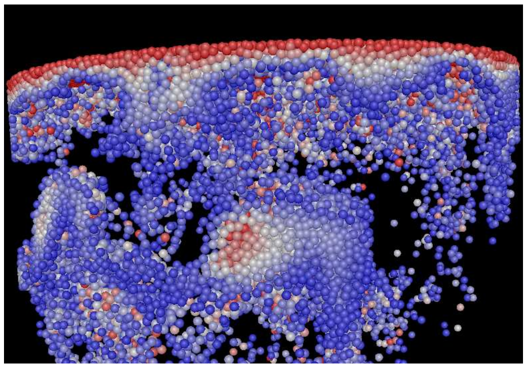

(b) Diffusion tensor dataset.

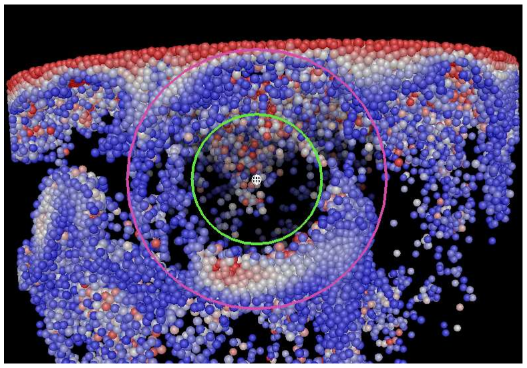
Figure 4.9: Glyph-based visualizations for the two datasets in our experiments. (a): Spherical glyph visualization of the particle dataset. The particles on the top surface have the highest concentration values. The particles in the middle and bottom of the image form strip regions that are the viscous fingers. (b): Top view of the superquadric glyph visualization for the diffusion tensor in a tumor patient's brain. The three colored surfaces represent ventricles (blue), cystic part of the tumor (green), and solid part of the tumor (red).

4.4.1 Different Deformation Models

In this case study, we use a particle dataset from a fluid dynamics simulation of viscous fluids [3]. This simulation uses the Finite Pointset Method (FPM), which is a meshfree method, and generates particle datasets [67] containing large number of particles. It studies how the salt supplied from the top of water is dissolved in the water in a cylinder domain. The concentration of salt solution is a scalar attribute of the particle. The area of high concentration is called the viscous finger, which is the interesting feature we want to observe without occlusion. Although this dataset is an ensemble time varying dataset, we pick one time step of one ensemble member to demonstrate our technique. Fig. 4.9a visualizes 19,483 particles whose concentrations are greater than a threshold. Each spherical glyph represents a particle whose concentration value is mapped to the sphere's color using the cool/warm color map. The goal of the visualization is to explore all the viscous fingers, including the ones hidden inside the volume.

Fig. 4.10 gives an example of how to use the circular round lens to pull away the particles on the volume surface and see the viscous fingers inside. Using a round lens with the screen space deformation model, we can completely remove the occlusion as shown in Fig. 4.10b. In the lens focus region, we can see the viscous finger with the shape of an inverted triangle. The particles inside the focus region have different brightness values that provide additional depth cue. In the bottom of the context region, another big viscous finger, which originally occludes our focus feature, is still visible but compressed into an arc shape. The object space deformation model as shown in Fig. 4.10c can preserve the spatial structure or the shape of the displaced viscous finger in bottom of the context region better than the screen space deformation model in Fig. 4.10b. This is because in the object space deformation model, the region with the viscous finger has denser glyphs and hence is more rigid, which prevents excessive local space distortion. On the other hand, we can still see some context glyphs in the focus region in Fig. 4.10c, because the object space deformation model cannot ensure complete removal of occlusions.


(a) Without lens.


(b) Screen space deformation model

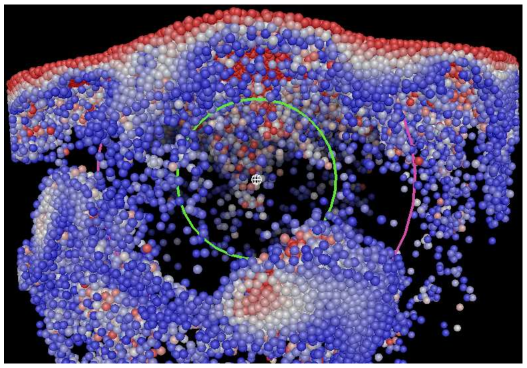
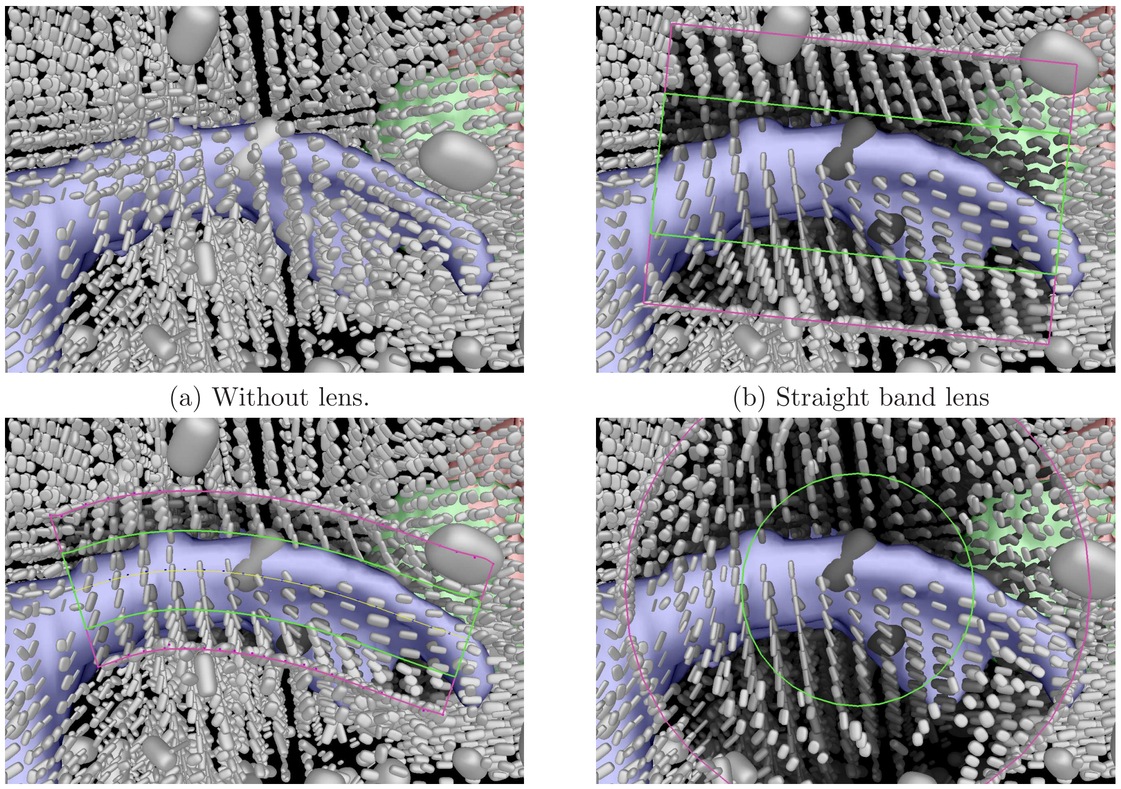

(c) Object space deformation model

Figure 4.10: Comparison of the two deformation models. (a) is a view without using our GlyphLens, which shows high concentration particles on the top and a big reddish viscous finger in the middle. In (b), the screen space deformation model lens pulls away the big viscous finger into the bottom of its context region, as well as removing the sparse blue particles from the focus region. In (c), the object space deformation model is usded, which preserves the shape of the deformed viscous finger better. Both deformation models reveal the originally occluded viscous finger in the focus region.

4.4.2 Different Shape Models

In this case study, we visualize a brain diffusion tensor imaging (DTI) from a patient with brain tumor using superquadric glyphs and explore the white matter especially around the tumor. Brain DTI image is used in neuroscience to study the fibrous structure of white matter. The 3×3 tensor matrix can be visualized by superquadric tensor glyphs [62] with the eigen decomposition results of the tensor matrix. The orientation and shape of the superquadric glyph depict the direction and crossing of the white matter tracts. As shown in Fig. 4.9b, we put 7,549 superquadric glyphs, which are generated by the Teem Toolkit library [4], on the grid points whose fractional anisotropy (FA) value is greater than 0.4. The tensor field around the ventricles and the tumors are the interesting features to the neuroscientists. However, those glyphs are mostly occluded by many other glyphs.



(a) Without lens.

(b) Straight band lens

(c) Curved band lens

(d) Circular round lens

Figure 4.11: Visualizations of the ventricles and its surrounding tensor field before and after using different shapes of the Glyphlens. Using the lens in (b)(c) and (d), we can clearly see the glyphs near the ventricle surface in the focus region without any occlusions. Among the three lens shapes, the curved band lens (c) fits the ventricle shape better than the other two.

Fig. 4.11 compares the displaced glyphs using lenses of different shapes. In Fig. 4.11b, we see that the glyphs near the left side of the ventricle are mostly vertically orientated and thin, which means the white matter tracts in this region are vertical and homogeneously uni-directional. The glyphs from left to right in the focus region gradually change to the horizontal orientation and become fatter, which indicates the tensor becomes less isotropic. In the context regions, we can see the displaced context glyphs. The other occluding glyphs that are far from the focus glyphs are pushed to the outer boundaries of the context region. Note that the GlyphLens only changes the glyphs' positions but not their orientations, so that users can mentally translate a glyph back to its original position while keeping its orientation. Because the ventricle is of an arc shape, we also tried the curved band lens as shown in Fig. 4.11c. Because of its thinner size, the curved band lens is going to displace less context glyphs than the straight band lens. As a comparison, we tested circular round lens in Fig. 4.11d. It displaces more glyphs than necessary and wastes a lot of screen space in this case.

4.4.3 Feature Locking and Snapping

The diffusion tensor around the tumor is an interesting feature, so our system identifies the glyphs near the tumors as the feature. When users pick one feature, e.g. the glyphs near the cystic part of the tumor, these glyphs are highlighted with cyan color in Fig. 4.12b. When we apply a circular round lens on the feature, the white context glyphs occluding the feature are pulled away. The lens can be snapped on the feature center, so users can look at the feature from arbitrary view directions without having to adjust the lens position. Looking from the top of the green tumor in Fig. 4.12a, we see the glyphs in the upper-left of the focus region following the

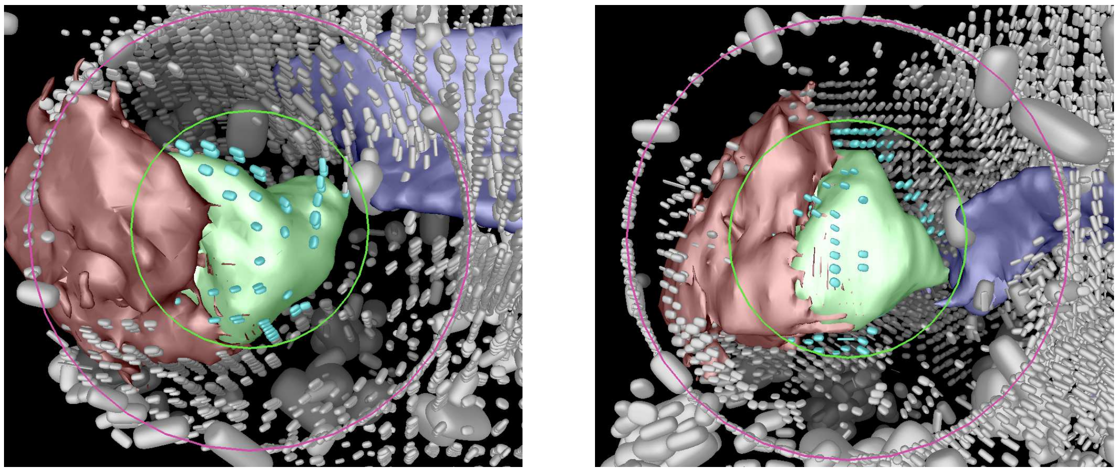

(a) Top view. (b) Side view.

Figure 4.12: The cyan glyphs near the cystic part of the tumor (the green surface) are the features and hence locked in their original positions without being displaced by the round lens. The lens snaps on the feature center, so users can freely change their view from top view (a) to side view (b) and look at the feature without any occlusion.

silhouette of the tumor; while the glyphs in the upper-right of the focus region are vertically oriented. Looking from the side of the green tumor in Fig. 4.12b, we see that the glyphs in the middle of the focus region are flat, which means planar anisotropy and demonstrates complex branching and crossing of the white matter tracts. The flat glyphs can also be seen in the bottom of the context region, which means the tensor field on that side of the tumor also has planar anisotropy.

4.5 User Feedback

We found one domain scientist for each of the two datasets used in the case studies. The domain scientist for the particle dataset is a PhD candidate in material science who studied simulations based on meshfree methods and have related journal publications. He is familiar with the particle dataset in our case study, which is also based on a meshfree method. After training him to use our system face to face, he was able to control the lens and explore the data by himself. The domain scientist for the DTI dataset is a faculty member who is experienced in neuroinformatics and has related publications. Because of a long distance, we demonstrated our system to him through a video conference and then collected his feedback.

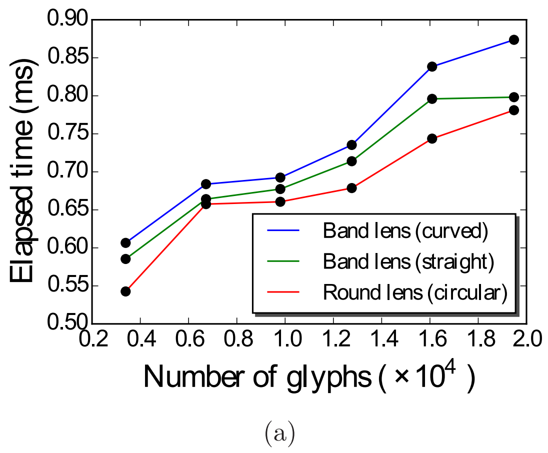
The main goals of the user study are to find out whether the occlusion is removed, whether the context is preserved, and whether the distortion is understood. Both scientists thought occlusion existed in glyph-based visualizations and can be removed when using the GlyphLens. They thought contexts are preserved but can be distracting sometimes. The DTI expert said, from a static image of displaced glyphs, the glyphs that are displaced and not displaced are mixed in the context region. This makes the context information somewhat confusing, especially when the orientations of the tensor glyphs varies largely in a local region. However, he also thought the animation of glyph movement can solve the confusion, because the animation can help distinguishing the displaced context glyphs from the static glyphs, observing the relationship between focus and context glyphs, and reminding users the depth relationship among the displaced glyphs. The material scientist thought the displacement can be a little confusing in the beginning. But when he became familiar with how the displacement works, the confusion was gone.

We also collected their feedback on our other designs. The DTI expert thought keeping the glyphs orientation during displacement is better than changing it, because changing the orientation of glyphs may create some artificial features. He can do transformation in his mind to reconnect the glyphs and form streamlines using the displaced glyphs. By comparing the two band lenses, he thought the specific advantage of straight band lens is its simplicity, while the curved band lens is slightly more complicated but gives more flexibility. Both scientists hoped the lens depth control can be improved. One scientist hoped to know the numerical value of the lens depth, while the other scientist thought a slider bar should be added to control the lens depth as well as the animation. The material scientist thought the back and forth movement in the object space model animation is a little distracting and should be reduced. Besides, both scientists thought the depth cue from the brightness attenuation is very clear and helpful.

4.6 Performance

We measured the performance of our technique on a machine running Windows 10 with an Intel Core i5-6600K CPU, 16 GB RAM and an NVIDIA GeForce GTX 970 GPU with 4GB memory. We implemented all the deformation computations using CUDA, the GPU-based parallel computation, to provide interactive performance. The viscous fluids particle dataset, visualized by spherical glyphs, is used in the performance tests.

Fig. 4.13a shows the computation time for three different lens shapes using the screen space deformation model. The computation time increases with the increasing number of glyphs. The circular round lens with the simplest geometry uses the smallest amount of time, while the curved band lens with more complex geometry uses the largest amount of time. The deformation times for the three lens shapes are very close and all below 1 millisecond even for displacing around 20,000 glyphs. Fig. 4.13b gives the frame rates for the two deformation models. The rendering ended up taking more time than the deformation. In the experiments of object space deformation



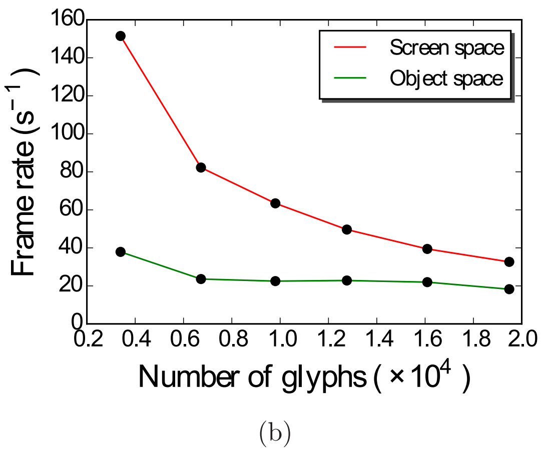


Figure 4.13: (a): Computing times for different shape models using screen space deformation model. (b): Frame rate for different deformation models.

model, we used a tetrahedron mesh composed of 78,125 ($25 \times 25 \times 25 \times 5$) tetrahedra. We can see that the screen space deformation model is much faster than the object space deformation model, because the physically-based modeling is expensive. The performance of the object space deformation model is less sensitive to the number of glyphs, because the deformation computation is performed on the tetrahedral mesh. The frame rate of object space deformation is just slightly above 20FPS, which is acceptable for user interactions. The frame rate can be increased if we use a coarser tetrahedral mesh, but then it will be difficult to preserve features in small sizes.

4.7 Conclusion

We have presented a view-dependent occlusion management tool, GlyphLens, for interactive glyph visualization and exploration. Our GlyphLens has the following major strengths:

- Remove occlusion completely using screen space deformation model. Or reduce occlusion using object space deformation model while better preserving the spatial relationships of the displaced glyphs.

- Ensure the visibility of the glyphs around the focus glyphs as the context glyphs in the lens.

- Specify the lens shape easily and intuitively using two shape models and various interaction devices.

A limitation of our lens is that we can only use one of the two deformation models at a time, but cannot combine the advantages of the two together into a single deformation model. The object space deformation does not work well when glyphs distribute relatively uniformly. Another limitation is the lack of visualization for the space deformation. A visualization of how the space inside the lens is deformed, maybe using deformed grids, can help viewers to mentally recover the displaced glyphs' original positions.

Chapter 5: Crystal Glyph: Visualization of Directional Distributions Based on the Cube Map

5.1 Introduction

With increases in spatial resolution and the emergence of ensemble simulations, massive amounts of data with uncertainty are commonly generated by simulations. In addition, storing and analyzing datasets at the full spatial or ensemble resolution has become unrealistic due to the required storage space and computing requirements. On the other hand, down-sampling the data to a lower resolution loses the details of data. Using data aggregation, such as a histogram, becomes a trade-off between data size and details. Histograms can be generated from aggregations of spatial partitions, ensemble members, or an analyzed distribution models. The resulting histogram can be used to describe and detect features. Dalal and Triggs [27] uses histograms of oriented gradient (HOG) as a feature descriptor to detect objects in images. Thompson *et al.* used the hixel to store a histogram for scalar values at each sample point and devised feature detection and visualization methods based on the hixels [105]. However, little work has been done on histogram representations of 3D vector data.

Visualizing the movement of a group of objects can assist the understanding of the motion of a large number of objects. Cosmology simulations generate a discrete set of particles. Visualizing the velocities of these particles can help understand their gravitational interaction and collapse [116]. In particle simulation of fluid dynamics, visualizing the movements of particles can help scientists understand the transport of physical quantities [67]. Plotting arrows to visualize the velocities of significant numbers of particles, however, is unrealistic due to the resulting visual clutter. Integrating streamlines or stream surfaces based on the particle velocities does not make sense if the data only describe local motion. To address the issue, we can use histograms of the velocities to represent the velocity distribution for a group of vectors and visualize the histograms. Neuroth *et al.* proposed two-dimensional (2D) velocity histogram to interactively visualize the large-scale velocity field. They showed that their histogram based representation can clearly describe the overall decomposition of velocities without causing visual clutter [89]. However, visualizing 3D vector distributions remains unsolved.

Glyph-based visualization is an ideal approach to visualize multivariate data. It allows users to quickly perceive the pattern of multivariate data item within the context of a spatial relationship [12]. Polar histograms [11], as shown in Figure 5.1, visualize the distribution of a group of 2D vectors based on their angles. Placing many polar histograms as glyphs in the data space can visualize the distributions of groups of 2D vectors in local regions. Jarema *et al.* designed similar glyphs to visualize 2D directional distributions in 2D vector field ensemble datasets [60]. The spherical histograms proposed in [47, 111, 93] can visualize distributions of three-dimensional vector fields, but their sphere partition methods are based on spherical coordinate

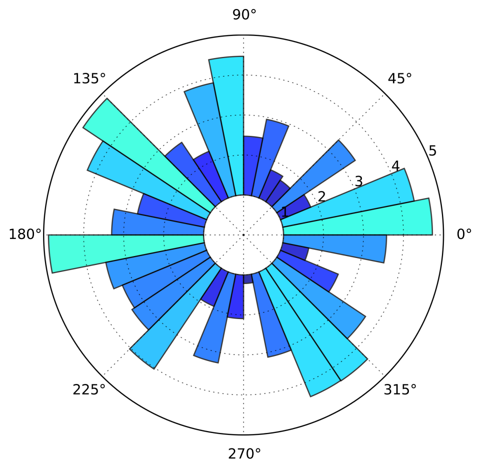


Figure 5.1: A polar histogram visualizes the distribution of 2D vector directions.

system that suffers from polar effects and inaccurately represent the distributions. To better visualize the distribution of 3D velocities, we need an accurate 3D visual representation of directional histograms.

In this work, we first introduce the concept of cube map histogram, a 3D directional histogram inspired by the environment cube map algorithm in computer graphics [44]. The cube map histogram can be efficiently and accurately computed from the Cartesian coordinates of vectors and stored using a much smaller size than the original vectors. Compared with the directional histogram based on other sphere discretization methods such as icosahedron, the cube map histogram can be more easily computed, interpolated and visualized. We introduce a 3D directional distribution glyph called crystal glyph. To create the crystal glyph, an OpenGL cube map texture is used to efficiently map the cube map histogram onto a sphere and deform its shape through OpenGL shading language (GLSL). We place the glyphs on a slicing plane and interactively change the plane's orientation and position as well as the glyph's density, in order to visualize the velocity distributions in different spatial locations and with different levels of details. Users can use mouse to pick one glyph and see the unfolded view of its cube map histogram without occlusion. To provide a dynamic and intuitive visualization of global vector directions, we animate a texture on the glyph's surface that follows the velocity directions. In the remainder of this work we review previous work in Section 2.4, explain the computation of the cube map histogram in Section 5.2, describe the glyph-based visualization in Section 5.3, present case studies on different datasets to show the effectiveness of our crystal glyph visualization method in Section 5.4, and measure its performance in Section 5.5.

5.2 3D Directional Histogram

A 3D directional histogram records the distribution of 3D vectors. For 3D velocity data, we omit their velocity magnitude, and only record their directions in our histogram. Users can choose to generate the velocity magnitude distribution using the traditional one-dimensional histogram if desired.

A bin in this histogram represents a range of similar 3D vector directions. For 2D vector histogram as shown in Figure 5.1, each bin is a fan and all bins have the same size. For 3D vector histogram, binning the 3D vectors is the same as partitioning the sphere surface into small patches. Each patch represents a bin, and its area describes the bin size. In a unit sphere, this patch area is equal to the solid angle, a two-dimensional angle in three-dimensional space, subtended from the sphere center. When bin sizes are equal, the probability density of a bin is proportional to the bin frequency (bin counts); when the bin sizes are not equal, a bin's probability density is equal to its frequency divided by the bin size. It is difficult to have small bins of the same shape and size for 3D vectors because partitioning a sphere into small same shape patches is non-trivial. In order to compute an accurate probability density of a bin, we have to compute the sizes of all bins.

In this section, we fist describe how to partition the sphere surface into quadrilateral patches using the concept of the cube map in computer graphics environment mapping. Our cube map based sphere partitioning ensures sphere patches of similar sizes, and it achieves easy binning as described in Section 5.2.1. Then we show how to compute the bin sizes, which can be used to normalize the bin frequency and produce the probability density of the distribution, in Section 5.2.2.

5.2.1 Cube Map Histogram Construction

There are existing methods to partition the sphere into patches of equal area or similar areas, such as spherical polar grid (latitude-longitude), icosahedral-based grid (triangles or hexagons), cubed grid, spiral grid [57], Fibonacci grids [102], Leopardi [75]'s grid, and unstructured grid. To represent a 3D directional distribution, we need to design a histogram that satisfies two requirements. First for a given vector direction, its corresponding bin should be easy and fast to determine. Second, it should be easy to interpolate the histogram to a continuous distribution in the 2D space of the sphere surface in order to sample and render the distribution, which means the bins' adjacency information should be easy to determine. In some partitioning methods, such as the icosahedral-based grid and unstructured grid, determining the bin is not easy because it requires an expensive intersection test between the polygons and a ray of the direction. In some partitioning methods, such as the spiral grid, Fibonacci grid and Leopardi's method, there are no accurate interpolation methods to evaluate an arbitrary point on the sphere. The spherical polar grid suffers from the pole effects, so the interpolation near its two poles may produce artifacts. Westerteiger *et al.* used HEALPix grid [119] to decompose the sphere surface hierarchically and record the grid in the GPU memory for rendering the terrain on the sphere. Even though this hierarchical grid is too complex to record our simple histogram, it inspires us to use a grid that is easy to be recorded and looked up in the GPU memory. On the other hand, the cubed grid-based sphere partitioning, as shown in the Figure 5.2, which produce our cube map histogram, satisfies our two requirements. The bin of a vector can be determined by simple division among its x, y, z coordinates. Interpolating the cube map histogram is mostly a simple bilinear interpolation among its easily determined adjacent bins. When rendering the cube map histogram, the OpenGL cube map texture and the GLSL sampler can take care of all the adjacency lookups and value interpolation automatically and efficiently.

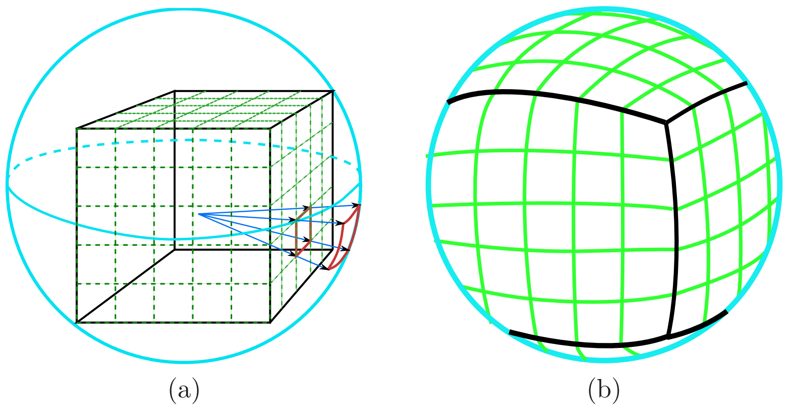


Figure 5.2: (a) The projection between a patch on the sphere surface and a grid cell on the inscribed cube. (b) Cube map grid on the sphere.

Our cubed grid on the sphere is created by projecting a cubic uniform grid onto a sphere surface. Assume there is an inscribed cube in a unit sphere, whose faces are evenly divided into a regular grid as shown in Figure 5.2a. Projecting the cube's grid from the cube center towards the sphere surface produces a grid on the sphere. The points on these two grids then have an one-to-one correspondence after the projection. We use the grid cells as the histogram bins of our cube map histogram. From another perspective, if we cast a ray from the sphere center following a 3D vector, the vector belongs to the bin of the cell that it intersects with.

How to compute the cube map bin based on a given 3D vector coordinate is computationally simple and is explained in the original environment mapping paper [44]. Here, we briefly describe it using our application. For a given vector in Cartesian coordinates $v = (v_x, v_y, v_z)$, we need to determine its bin represented as three integer indices (b_f, b_x, b_y), where b_f is the index of the cube face in the range of $[0, 5]$, and b_x and b_y are the bin's indices in the x and y directions on each face. Table 5.1 gives the computation details, where m is the dimension of the 2D grid on each face, i.e. each face has $m \times m$ cells. The 3D bin index can be further reduced to 2D as $(m \times b_f + b_y, b_x)$ to render the histogram as a 2D image, or be reduced to 1D as $(m^2 \times b_f + m \times b_y + b_x)$ to store in the linear memory space or to load into the OpenGL cube map texture memory. From the formulas in the table, we see that the computations only involve simple floating point operations that gives low computation cost. This small computation can be helpful when dealing with big datasets containing large number of vectors.

Table 5.1: Binning: computing the bin indices (b_f, b_x, b_y) for the given vector $v = (v_x, v_y, v_z)$.

Condition	b_f	b_x	b_y
$v_x \geq \lvert v_y \rvert$ & $v_x > \lvert v_z \rvert$	0	$\lfloor (1 - \frac{v_y}{\lvert v_x \rvert}) \cdot \frac{d}{2} \rfloor$	$\lfloor (1 - \frac{v_z}{\lvert v_x \rvert}) \cdot \frac{d}{2} \rfloor$
$v_x < -\lvert v_y \rvert$ & $v_x \leq -\lvert v_z \rvert$	1	$\lfloor (1 - \frac{v_y}{\lvert v_x \rvert}) \cdot \frac{d}{2} \rfloor$	$\lfloor (1 + \frac{v_z}{\lvert v_x \rvert}) \cdot \frac{d}{2} \rfloor$
$v_y \geq \lvert v_z \rvert$ & $v_y > \lvert v_x \rvert$	2	$\lfloor (1 + \frac{v_z}{\lvert v_y \rvert}) \cdot \frac{d}{2} \rfloor$	$\lfloor (1 + \frac{v_x}{\lvert v_y \rvert}) \cdot \frac{d}{2} \rfloor$
$v_y < -\lvert v_z \rvert$ & $v_y \leq -\lvert v_x \rvert$	3	$\lfloor (1 - \frac{v_z}{\lvert v_y \rvert}) \cdot \frac{d}{2} \rfloor$	$\lfloor (1 + \frac{v_x}{\lvert v_y \rvert}) \cdot \frac{d}{2} \rfloor$
$v_z \geq \lvert v_x \rvert$ & $v_z > \lvert v_y \rvert$	4	$\lfloor (1 - \frac{v_y}{\lvert v_z \rvert}) \cdot \frac{d}{2} \rfloor$	$\lfloor (1 + \frac{v_x}{\lvert v_z \rvert}) \cdot \frac{d}{2} \rfloor$
$v_z < -\lvert v_x \rvert$ & $v_z \leq -\lvert v_y \rvert$	5	$\lfloor (1 - \frac{v_y}{\lvert v_z \rvert}) \cdot \frac{d}{2} \rfloor$	$\lfloor (1 - \frac{v_x}{\lvert v_z \rvert}) \cdot \frac{d}{2} \rfloor$

5.2.2 Histogram Normalization

After determining the bins of all the vectors and counting the frequencies for all the bins, the frequencies need to be normalized by their corresponding bin sizes or solid angles. To normalize the histogram, we need to compute the histogram bin size or the solid angle of each partition that is the sum of the two constituent spherical triangles' solid angles. The spherical triangle's solid angle is equal to the spherical excess that can be computed by l'Huilier's theorem [132].

The solid angles of other cells can be computed similarly. Because the six faces are symmetric, we only need to compute the solid angles for the cells on one face and reuse them for the cells on the other 5 faces.

Figure 5.3a shows the solid angles of the cells (or bin sizes) on one face of a high resolution cube map. We notice that the bins around the center have larger solid angles than the bins near the boundaries. To verify the correctness of using the solid angle to normalize the cube map histogram, we generate uniform samples by randomly sampling a large number of 3D vectors, and then generate the cube map of the bins' frequencies whose one face is as shown in Figure 5.3b. After dividing it by

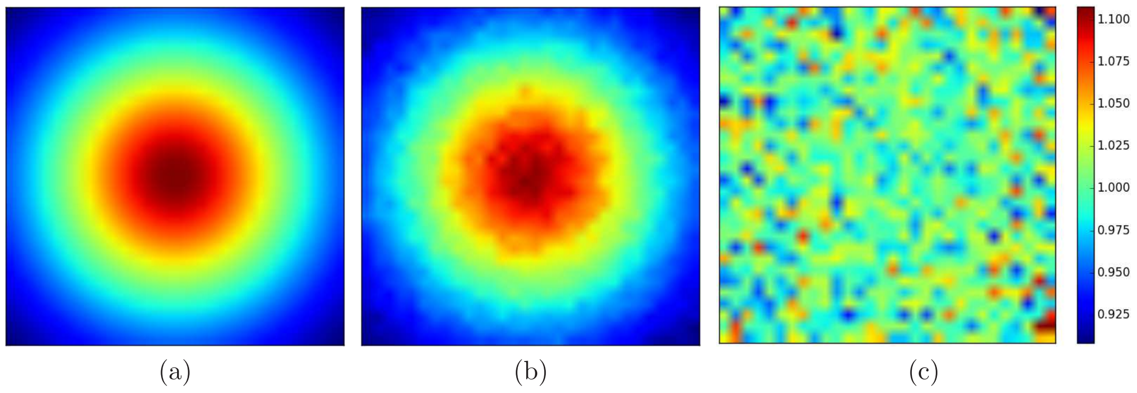


Figure 5.3: (a) Solid angles for the cells on one cube face. (b) The cube map histogram of bin frequencies for random samples. (c) The histogram normalized by the solid angles.

the solid angle shown in Figure 5.3a, we get the cube map histogram of probability densities shown in Figure 5.3c. We can see the values in Figure 5.3c are mostly around the value of 1 with small variance that are from the sampling error. Thus, we verified that the computed distribution normalized by the solid angles has uniform bin densities as expected.

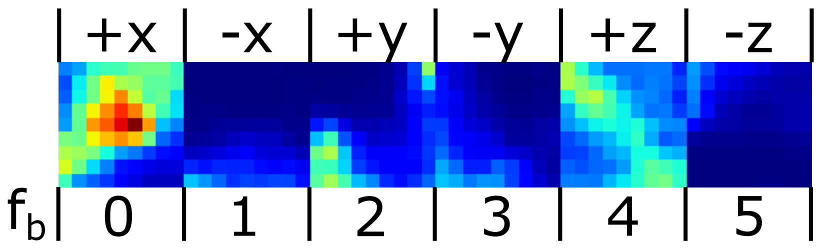


Figure 5.4: The 6 faces of the cube map histogram ordered by the cube face index b_f.

To apply it on a vector field dataset, we compute the cube map histogram of an example dataset and concatenate the 6 faces into a 2D image shown in Figure 5.4.

From this figure we can tell that the directions on the $+x$ face ($b_f = 0$) have the highest probability densities. Besides, the histogram bins on the $+z$ face ($b_f = 4$) also have relatively high probability densities.

5.3 Directional Distribution Visualization

In the previous section, we plot a cube map histogram on a 2D image in Figure 5.4. However in this 2D visualization, velocity directions corresponding to the bins lack 3D spatial reference, which makes this visualization not intuitive. In this section, we demonstrate the pipeline of storing and visualizing directional distributions using the cube map histogram, describe how to use 3D glyphs to visualize the directional histogram in 3D space, and evaluate its effectiveness based on our design goals.

5.3.1 Data Processing Pipeline

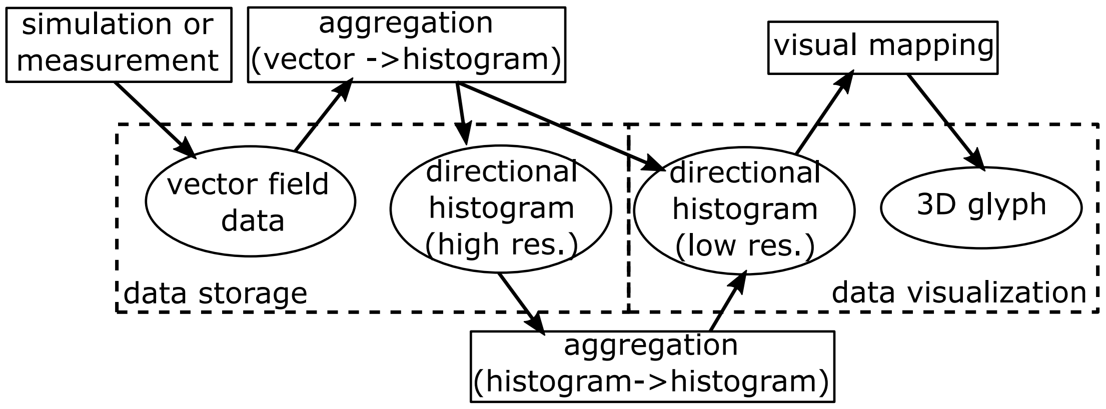

simulation or measurement
aggregation (vector ->histogram)
visual mapping
vector field data
directional histogram (high res.)
directional histogram (low res.)
3D glyph
data storage
data visualization
aggregation (histogram->histogram)

Figure 5.5: Data storage and visualization pipeline.

Our cube map histogram can be used for both storing and visualizing velocity distributions. Figure 5.5 gives our data processing and visualization pipeline. Vector data generated from simulations or collected from measurements are output first. In order to save storage, one can divide the vectors into groups based on their spatial locations, then aggregate the velocities into the directional histogram for each group. To visualize the velocity distributions, we need to generate directional histograms for visualization by either aggregating the original vector data or by further aggregating the stored directional histogram. In the end, we map the data attributes in the directional histogram onto our 3D glyph for visualization.

5.3.2 Visualization Design Goals

Designing the glyph is essentially mapping two data attributes of each histogram bin, 3D vector direction and the probability density, to the visual channels on the glyph, such as size, color, shape and orientation. Borgo *et al.* [12] provide thirteen general considerations and guidelines for glyph design by summarizing a few previous glyph-based techniques. By combining Borgo *et al.*'s design guidelines and our application, we propose the following design goals to guide the design of our glyph:

[**DG1**] Vector direction should be naturally and intuitively mapped to a visual channel. With natural mapping, users can easily infer the direction from the visual output.

[**DG2**] Use perceptually uniform and accurate visual channels to map the probability density. Equal differences in the data values should be perceived as equal from the visual channels.

[**DG3**] Redundantly map more visual channels on one data attribute if possible. Redundant mapping can reduce the possibility of information loss. Since we only have two data attributes, redundant mapping is affordable.

[DG4] Visual channels are orthogonal to each other. Each visual channel can be perceived independently without being interfered with other visual channels.

[DG5] Minimize occlusion and visual clutter. The surfaces of all glyphs should be visible. Different glyphs should not occlude each other.

[DG6] Depths of glyphs should be easy to discern. Depth cues should be provided for perceiving the 3D information inside a glyph and among different glyphs.

5.3.3 Glyph Design and Evaluation

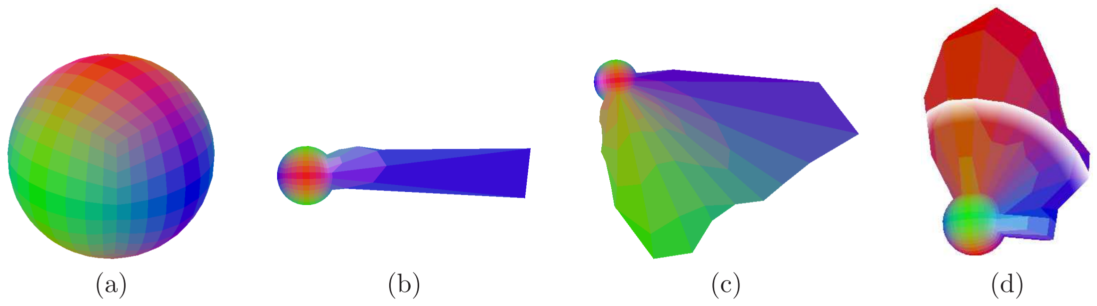


Figure 5.6: (a) The glyph is a sphere before drawing a histogram on it. (b-c) are the crystal glyphs with the velocities distributed in (b) z direction, (c) y and z directions; (d) white band moves from the spherical base outward as animation.

Our crystal glyph is generated from a sphere, whose grid is formed by projecting from an inscribed cube grid, as shown in Figure 5.6a. Note that the grid does not have to match the grid on the cube map histogram, because we can easily sample the cube map histogram and interpolate the values. Each face of the spherical mesh is a quadrilateral with a uniform color. The color is determined by the sphere's normal direction (x, y, z) at the center of the quadrilateral. The absolute values of the unit vector's components are used as the red, green and blue color components, i.e. $(r, g, b) = (abs(x), abs(y), abs(z))$. Each point on the sphere has a unique normal direction $\vec{v}$. This point then represents the vector direction of $\vec{v}$ in the directional distribution. We define p_v as the probability density of direction $\vec{v}$ in the directional distribution. Then as shown in Figure 5.6b and 5.6c, we extrude each sphere vertex along its normal direction $\vec{v}$ by an amount $H(p_v)$ that is a function of p_v and form a terrain on the sphere. We let $H(p_v)$ be a monotonically increasing function of p_v, so that higher terrain on the sphere surface corresponds to a higher probability density. More specifically, we have

$$H(p_v) = a \times r \times p_v^{1/d} \tag{5.1}$$

where a is a scaling coefficient that can be adjusted by users to control the overall amount of extrusion; r is the radius of the original sphere. d is a non-linear scaling factor from the three values $\{1, 2, 3\}$ chosen by the users to control different mapping scenarios based on their goals. When $d = 1$, p_v is proportional to $H(p_v)$, so the probability density maps to the vertex's distance to the original sphere surface. When $d = 2$, p_v is proportional to $H^2(p_v)$, so the probability density maps to the area of the extruded quadrilateral. When $d = 3$, p_v is proportional to $H^3(p_v)$, so the probability density maps to the volume of the extrusion that is a hexahedron. Since the cube map histogram serves as a lookup function T, to find the probability density p_v for a given direction $\vec{v}$, we have $p_v = T(\vec{v})$. By using the OpenGL cube map texture to store the cube map histogram, we can efficiently interpolate the cube map histogram into a continuous directional distribution. Plugging the lookup function into Equation 5.1, we get

$$H'(\vec{v}) = H(p_v) = H(T(\vec{v})) = a \times r \times (T(\vec{v}))^{1/d} \tag{5.2}$$

Using this equation, we can compute the amount of extrusion $H'(\vec{v})$ for any given point on the sphere whose normal is $\vec{v}$.

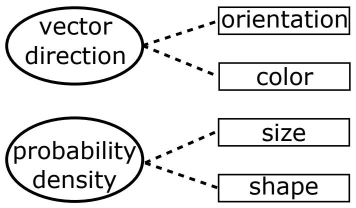


Figure 5.7: Mapping (dotted lines) between data attributes (ellipses) and the visual channels (rectangles).

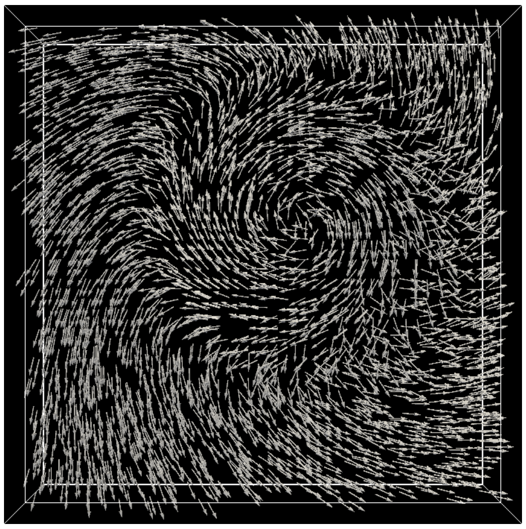
Figure 5.7 gives the mapping between our data attributes on the left and the visual channels on the right. The vector direction is mapped to two visual channels, the orientation of the extruded quadrilateral patch and its color. It is difficult to interpret the orientation after projected to a 2D image, especially when the glyph is small in the image, so the color can help to represent directions. Mapping color on direction is intuitive because red, green, and blue colors are commonly used as colors to draw x, y, and z axes that show directions. The XYZ-RGB color mapping scheme has been widely used in showing the 3D orientation in tensor data [90, 108]. Mapping the vector direction to the extrusion direction of the patch is intuitive and natural, which satisfies our design goal [**DG1**].

The probability density of each histogram bin is mapped to two visual channels: the size of the extrusion and the overall shape of the glyph. For example, the two glyphs in Figure 5.6b and Figure 5.6c have different shapes: one with a linear shape and the other one with a planar shape. The shape clearly illustrates the overall trend of the distribution. We allow users to choose the size of different geometric properties, such as length, area or volume, to map bin frequency by choosing a different d in Equation 5.1 as stated. According to [21], the length is a relatively accurate visual channel, so our bin frequency can be accurately perceived when we choose $d = 1$, which satisfies our design goal [**DG2**]. The three glyphs in Figure 5.6 b-d all use the parameter $d = 2$. In this case, different glyphs' projected image areas are similar. When looking at a group of glyphs on the image, we can find the total probability of a particular vector direction by looking at the total image area covered by its corresponding color.

Both data attributes are mapped to two visual channels, so our design satisfies the design goal [**DG3**]. From the glyph rendering as shown in Figure 5.6b, using a uniform color on each face shows the mesh grid that helps viewers perceive the surface curvature [17]. The Phong illumination model is used to illuminate the glyph. These shading effects provides depth cues inside the glyph and satisfy the design goal [**DG6**]. Regarding to the design goal [**DG4**], color is orthogonal to all other visual channels, which satisfies the design goal. However, because of projecting on a 2D screen, two patches with the same terrain heights (the amount of extrusion) may have different sizes on the image because they extrude to different directions. So the visual channels of size and orientation are not always orthogonal. On the other hand, by interactively rotating the glyph, we can compare the surface terrain heights from different viewpoints that help filter out the interference of orientation.

As shown in Figure 5.6d and the accompanying video, when animation is activated on the glyph visualization, a white band is drawn on the glyph. The white band moves from the spherical base outward by following the surface's extrusion directions, which are essentially the velocity directions. The white bands on different glyphs start from their spherical base at random times to avoid artifacts. The animation is implemented in the OpenGL fragment shader with high performance.

5.3.4 Visualization System Design and Evaluation


(a) 3D arrow plot.

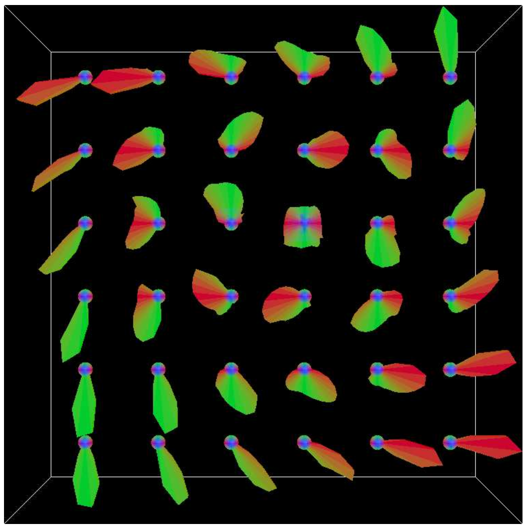

(b) Crystal glyph.

Figure 5.8: Visualizing the flow in the tornado dataset.

The synthesized $48 \times 48 \times 48$ tornado dataset is used as an example. Our glyph placement strategy is putting glyphs uniformly on a slicing plane to represent the vector field of a layer as shown in Figure 5.8b to eliminate visual clutter. By interactively changing the plane orientation and position in the volume, the glyphs can visualize the velocity of the entire spatial domain without causing much occlusion, which satisfies our design goal [**DG5**]. From glyphs' relative positions on the slicing plane, we can find out the depth order among glyphs, which satisfies design goal [**DG6**].

Figure 5.8 shows the visualization of the 3D velocity field in the bottom $48 \times 48 \times 8$ layer using both an arrow plot and our crystal glyph. In the arrow plot (Figure 5.8a), 5000 arrows are placed in the volume. The arrows overlap with each other and result in severe visual clutter, so a majority of them are difficult to be identified. On the other hand, in our crystal glyph visualization (Figure 5.8b), glyphs are well separated and have different fan shapes. Those in the center have planar shapes showing high variance in their directions, and the ones on the sides have linear shape showing low variance. Each of them points to different directions from the spherical base. The glyph at the 3rd row and 4th colume has velocity distributed in all directions in this slicing plane, which means the tornado center is nearby.

From a glyph's screen projection as in Figure 5.9a, the bins on the backside are not visible if viewing from a fixed view direction. In order to help users explore a specific glyph of interest and observe its represented cube map histogram without occlusion, we visualize the cube map histogram by unfolding it onto a 2D image. Figure 5.9 gives an example about how to unfold (or project) the cube map histogram on the glyph in Figure 5.9a. Two types of projections are provided: one (Figure 5.9b) is cutting along the cube edges and unfolding the 6 cube faces without distorting the grid; the other one (Figure 5.9c) is projecting to 2D spherical coordinates [14]. In the spherical coordinates projection, the x coordinate is the longitude $\theta \in [-\pi, \pi]$ and the y coordinate is the latitude $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. To draw the 2D image, one can simply draw each pixel at (θ, ϕ) using the directional probability of the 3D vector

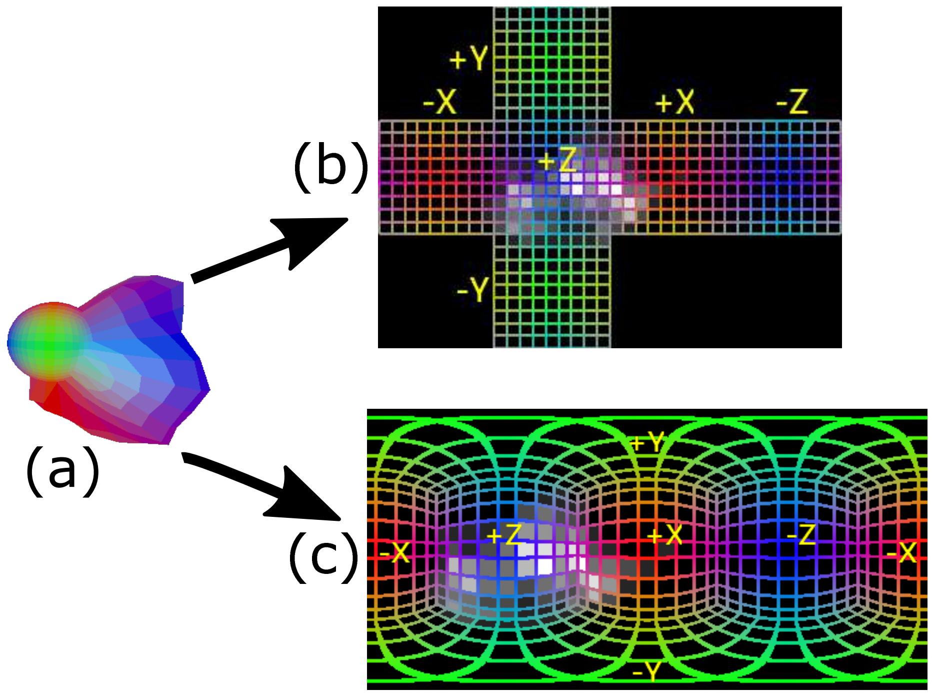


Figure 5.9: (b) and (c) are two different unfolded views of the cube map histogram on the user selected glyph (a).

direction $(\cos\phi\cos\theta, \sin\phi, \cos\phi\sin\theta)$ that can be easily obtained from the OpenGL cube map texture. The six face view of the cube map histogram in Figure 5.9b shows the bins using the same size, but the boundaries of the faces are mostly disconnected. This view is good for visualizing the spread (or range) of the distribution. On the other hand, the spherical coordinate view in Figure 5.9c gives mostly continuous bins but distorted bin shapes, which is good for identifying the number of peaks in the distribution. Users can choose either one based on their preference and goal. In the 2D views, the bin densities are mapped to color brightness on the image with a gray-scale colormap. A colored wireframe of the histogram bins is overlapped on the image to discretize the image into bins. By using the same color as the bins on the crystal glyph, users can easily find out their correspondence. In this way, users can clearly view the full velocity distribution of the selected glyph using the unfolded view of the cube map histogram. This unfolded view eliminates the occlusion on the glyph, which satisfies design goal [**DG5**].

In summary, our glyph and visualization system design meets almost all of our design goals.

5.4 Case Studies

To explore the effectiveness of our technique we applied our algorithm to two vector field datasets from different application domains: a cosmology dataset (particles) and a thermal hydraulics dataset (regular grid).

5.4.1 Cosmology

Our goal is to visualize the particle velocities in the cosmology dataset from the Dark Sky Simulations [116] that study the evolution of the large-scale Universe. We use the last time step of the dataset that contains 3D velocities of 2,097,152 particles. A halo is a group of gravitationally bounded particles with coherent structure. In this case study, we visualize the velocity direction distributions of 7,383 halos. The directional distribution of the particle velocities in every halo is computed and visualized as a crystal glyph. The glyph size is scaled with the radius of the represented halo. By observing the colors and shapes of the glyphs, we can understand the directional distribution of the particle velocities in each halo and how different halos move in space.

From Figure 5.10, we see many halos of different sizes. In the middle of the image, there is a big halo whose velocity directions spread across many directions,

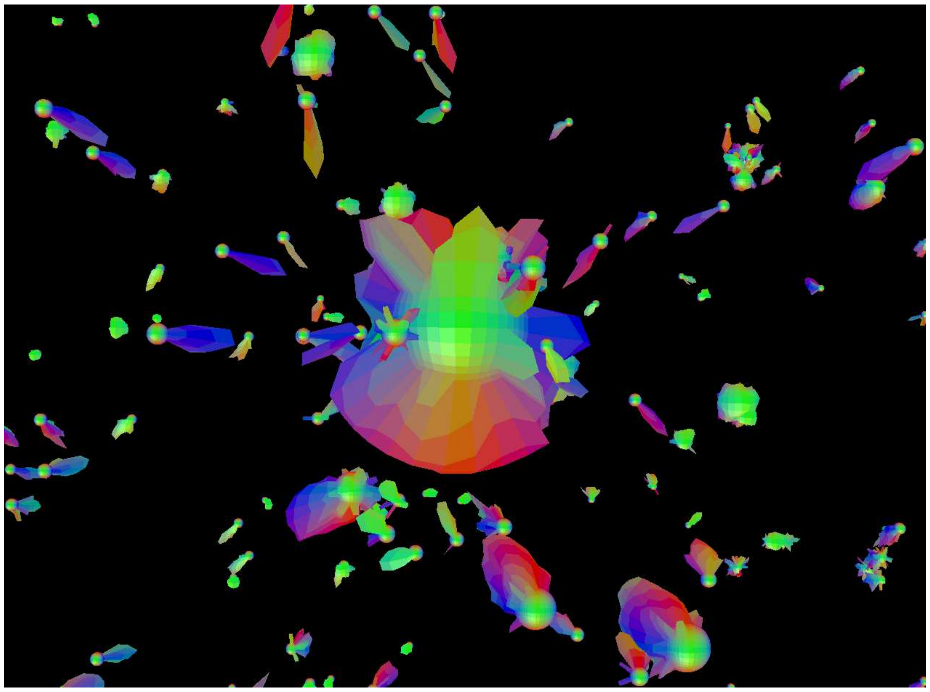


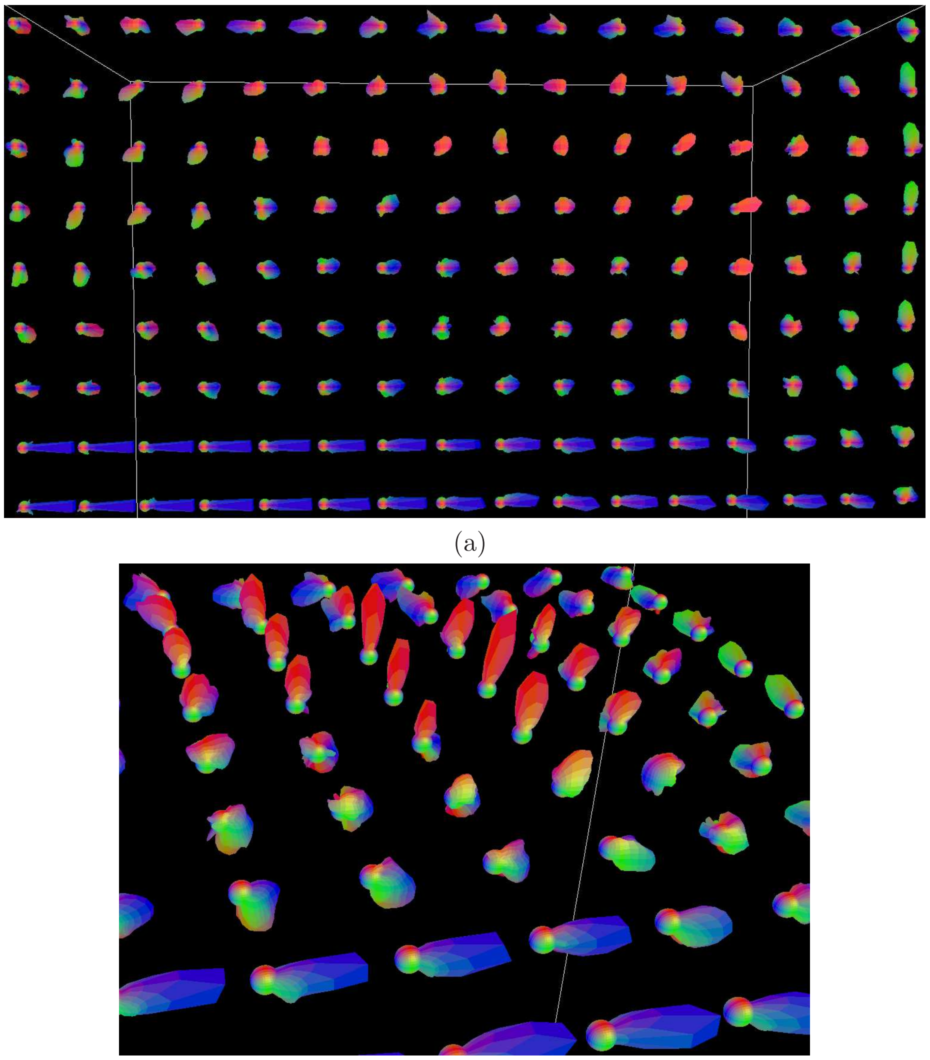
Figure 5.10: Crystal glyphs are used to visualize halos containing moving particles in the cosmology dataset.

which means the movements of the inside particles are very complex. Most of the surrounding halos are attracted by the gravity of the big halo and move towards the center. The velocities in some halos spread in a small range of directions, such as the purple glyphs on the upper-left and upper-right of the image; some others' directions spread in a big range, such as the two reddish glyphs at the bottom of the image. From these two glyphs, we not only see the major directions shown as the red high peaks, but we also see the outlier directions as the green low peaks. The shapes of most green glyphs are not clear because the extrusion direction of the green surface is perpendicular to the screen. To see them, we need to interactively change the view direction that is shown in our accompanying video.

5.4.2 Thermal Hydraulics

The *Nek* dataset is generated from the Nek5000 simulation that simulates thermal hydraulics in a nuclear reactor using the spectral element method [36]. The simulation demonstrates the coolant flowing between and around the pins in the reactor. The dataset we use is a 3D velocity field on a $512 \times 512 \times 512$ regular grid resampled from the original topology in the simulation.

The coolant flow in this dataset is turbulent, so the local velocities are very complex. Our crystal glyphs can visualize the directional distributions of the local regions and provide simple statistical representations of the complex velocity field. We place crystal glyphs on a slicing plane perpendicular to the x axis as shown in Figure 5.11a. Each glyph represents the vector directions in the local cubic region. The thin blue glyphs at the bottom of the image show the water moves straight to the right along the z direction. When reaching the right boundary, the water starts moving up shown as the green glyphs. Then, it moves left at the top of the image and then comes down at the left of the image. These movements form a circle that demonstrates how water circulates in the space. The glyphs at the top row have mixed colors of red and blue, which means the water flow has both components of x direction (towards the viewer) and z direction (towards the left). Because the red surfaces on the crystal glyphs extrude towards the viewers and are difficult to see from this view direction, we changed the view direction and look at this slice from the side as shown in Figure 5.11b. In this view, we can easily identify the regions containing x directions flows using the red color surfaces on the glyph. Among the big red glyphs, we can compare their heights and fatness to find out their relative probability densities and the variance of the distributions. The glyphs on the middle left of the image have low


(a)

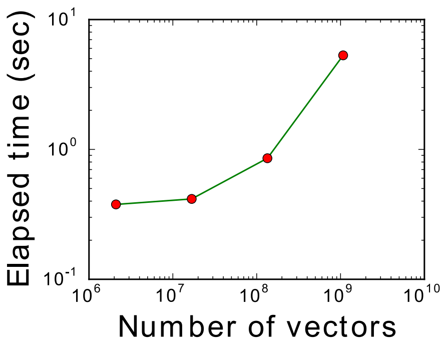
(b)

Figure 5.11: Crystal glyphs are used to visualize the velocity field in one layer of the thermal hydraulics dataset. (a) and (b) show the same group of glyphs in different view directions.

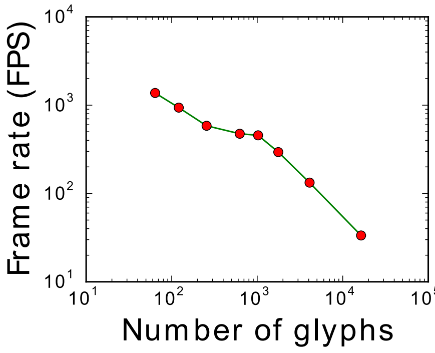
extrusions and rough surfaces that show turbulent flows going all different directions. For those regions, we can interactively place glyphs more densely, as demonstrated in the accompanying video, to see the distribution in smaller local regions and more details.

5.5 Performance

We measured the performance of our technique on a machine running Windows 7 with an Intel Core i7-4770 CPU, 16 GB RAM and an nVidia GeForce GTX 980 Ti GPU with 6GB of frame buffer memory. The thermal hydraulics dataset is used.



(a)



(b)

Figure 5.12: (a) Elapsed time of computing histograms from vector data. (b) Frame rate of rendering crystal glyphs.

Four resolutions (128^3, 256^3, 512^3 and 1024^3) are used to measure the elapsed times of computing the cube map histograms. In the computation, we first divide the data into a fixed number of $128 \times 128 \times 128$ partitions and then compute a histogram for each partition. In this way, no matter what resolution the original vector field dataset is, we produce the same number of histograms. OpenMP is used to accelerate the computation with multithreading. Figure 5.12a gives the execution time for the four input vector datasets of different resolutions. Processing the smallest 3 datasets only took less than 1 second. Even for the biggest dataset with over one billion vectors, it only took 5.3 seconds. The generated histograms can be either saved in disk or directly used as input to crystal glyph rendering.

To compute the cube map histogram during rendering, we only need to aggregate the already computed $128 \times 128 \times 128$ histograms in the glyphs' represented regions. The aggregation for all the experiments took less than 0.1 seconds. In the test, each glyph is rendered with a different $9 \times 9 \times 6$ cube map texture. The produced image resolution is 1419×996. Figure 5.12b shows the frame rates for an increasing number of glyphs. The frame rate is computed from the average time of rendering one frame. Even for rendering the largest number of glyphs (16,384 glyphs), the frame rate is higher than 30 frames per second (FPS), which is sufficient for supporting effective user interaction. We have not tested more than 16,384 glyphs to avoid visual clutter.

5.6 Conclusion

We have presented a technique to efficiently compute and store distributions of three-dimensional vector directions using a cube map histogram. This histogram allows easy interpolation of the probability density for any arbitrary direction. Besides, we designed the crystal glyph to visualize local 3D directional distributions with the OpenGL cube map texture. To allow users to freely explore the vector field, we designed an interactive visualization system to present glyphs in the user specified data layer without visual clutter. Additionally, we presented two case studies using cosmology and thermal hydraulics datasets, and we reported the performance of both the cube map computation and the glyph-based visualization.

Chapter 6: Salient Time Steps Selection from Large Scale Time-Varying Data Sets with Dynamic Time Warping

6.1 Introduction

Salient time steps are the time steps that are from a time series and contain the most interesting features. Previously researchers have proposed various methods to select salient time steps from time-varying data sets. One approach is based on user input. By providing an overview of the time-varying data in a 2D layout, the user can judge what time steps are more important and hence should be further investigated. Examples of this approach include the spreadsheet-like layout for time series data [124], storyboard-like layout in the form of images [80], or the time histogram method that concatenates data distributions over time [5, 6]. For certain applications such as data compression or in situ analysis, nevertheless, it will be challenging to have direct intervention from the users so automatic approaches are more preferred. For this purpose, there exist several algorithms that can find representative time steps. One method is to group similar time steps using greedy approaches, and then choose one time step from each group. Such an approach is taken by Akiba *et al.* [5] to group time steps based on data distribution. Also, a time step can be deemed salient if it changes drastically from the previous time steps. Based on this concept, Wang *et* *al.* measure the mutual information in local time steps to see when the distribution has a larger amount of changes. [113]. Those approaches, however, only consider the data in a local time range. Given a constraint in the total number of time step to select, users are not able to interactively pick any number of time steps. Besides, the selected time steps can be suboptimal if the global information over the entire time sequence is not considered.

To allow scientists to focus on the most salient data for interactive analysis, we present a novel technique for identifying key time steps from a time-varying data set using a global optimization scheme. A key time step is a time step that can best represent the data in its surrounding time steps. Given a user-desired number of key time steps and a distance metric to compare data between two time steps, time steps that can minimize the total cost of representing the whole data sequence are chosen as the key time steps. Our technique is inspired by *D*ynamic Time Warping (DTW), a technique that non-linearly warps one time series to another at a minimum cost. Instead of mapping two different time series, as in most of the DTW applications, in our technique we treat the input time-varying data as one time series and warp it to a subset of its own time steps. We design an efficient computation algorithm based on dynamic programming that can rapidly identify the time steps with the minimum cost under nonlinear time warping. We present a data browsing tool that can allow the user to interactively visualize and analyze time varying data with the key time steps selected by our algorithm. The key time steps selection process can be done very efficiently and hence allows the user to select any number of key time steps in real-time. When the user detects an interesting time interval by visualizing the current key time steps, they can request additional time steps in the time interval, invoking our key time step selection algorithm recursively.

We apply our key time step selection algorithm to a cloud-resolving simulation using the Earth Mover's Distance [96] as the dissimilarity measure. From the visualization and analysis of the data in the key time steps, we can observe salient moments of the Madden-Julian Oscillation, a long term weather event. Another application of our algorithm is to identify key time steps for time varying isosurfaces. Based on the isosurface dissimilarity map [16], our algorithm allows the user to obtain insight into the temporal evolution of isosurfaces.

6.2 Time Sequence Alignment

The goal of this work is to identify the best K time steps from a time-varying data set of N time steps, where K is specified by the user. The selected time steps should be the most representative among all the possible choices, i.e., they should maximize the information conveyed by the original data. Since K is smaller than N, to estimate the cost of choosing the K key time steps we need to first align the original time-varying data to the selected subset. For this, we can consider the entire data set as a time series, where each time point represents the volume data at one time step. The concept of time sequence alignment appears in the Dynamic Time Warping(DTW) algorithm. DTW algorithm generates an optimal alignment between two time series, and measures the cost of the alignment based on the similarity between the time points in two series. Inspired by DTW, we apply a similar method to measure the cost of key time steps. In our case, the time series is mapped to a subsequence of its own. Our goal is to find the best K time steps that give the minimum cost determined by DTW. In the following, we describe our approach in detail.

6.2.1 Alignment in Dynamic Time Warping

Given two time sequences X and Y, to measure the similarity between the two sequences, a non-linear mapping can be done to align points considering their similarity. An example of such a mapping is shown in Figure 6.1.

To represent the mapping, let an alignment between two sequences $X = (x_1, x_2, ..., x_N)$ and $Y = (y_1, y_2, ..., y_M)$ be $P = \{p_1, p_2, ..., p_L\} = \{(x_{n_1}, y_{m_1}), (x_{n_2}, y_{m_2}), ..., (x_{n_L}, y_{m_L})\}$. Each (x_{n_i}, y_{m_i}) is a bi-directional mapping between x_{n_i} in sequence X and y_{m_i} in sequence Y, and L is the total number of pairs between X and Y. Because one element in a sequence can map to multiple elements in the other sequence and vice versa, and P lists all the element-element alignment pairs, the number of pairs L in P, is equal to or greater than both M and N, i.e., $L \geq M$ and $L \geq N$. Figure 6.1 shows the alignment pairs in green lines. Because of the non-linear nature, it can be seen that an element on one sequence can map onto more than one element on the other sequence. For example, x_1 maps to both y_1 and y_2, $x_{n_1} = x_{n_2} = x_1$.

To measure the cost of aligning two sequences X and Y, a dissimilarity function between two data points x_{n_i} and y_{m_i}, $D(x_{n_i}, y_{m_i})$, is defined to measure the difference between the two elements. Then, the *overall mapping cost* of the alignment is defined as [86]:

$$C = \sum_{i=1}^{L} D(x_{n_i}, y_{m_i}) \tag{6.1}$$

DTW cost is defined as the overall minimum cost among the set of all possible mappings $\mathbb{P}$:

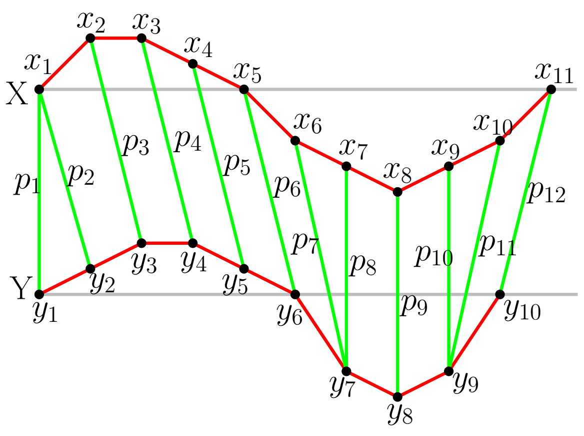


Figure 6.1: Alignment between two sequences X and Y

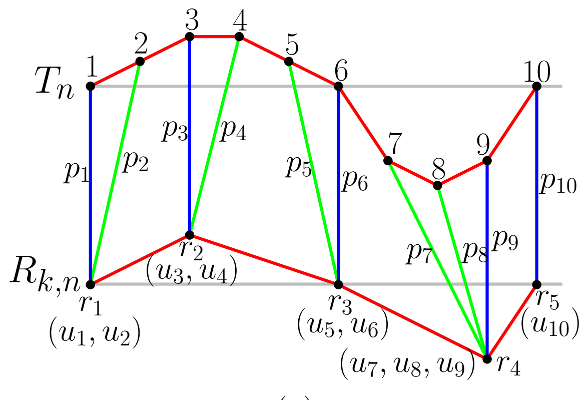
$$C^* = \min_{P \in \mathbb{P}} (C) = \min_{P \in \mathbb{P}} \left(\sum_{i=1}^{L} D\left(x_{n_i}, y_{m_i}\right) \right) \tag{6.2}$$

This lowest overall mapping cost defines an *optimal alignment* P^*, which is the dissimilarity between these two sequences. DTW uses a dynamic programming algorithm to generate the optimal alignment efficiently.

6.2.2 Mapping in Key Time Steps Selection

With the basic concept of DTW explained, in this section we discuss how key time steps can be selected from a full time series. To simplify the notations, hereafter we only list the time step indices rather than the actual data to represent a time sequence.

Given a sequence of n time steps $T_n = \{1, 2, ..., n-1, n\}$, called *full sequence*, we can choose k time steps $R_{k,n} = \{r_1, r_2, ..., r_{k-1}, r_k\}$ from the full sequence, where $1 \le r_1 < r_2 < ... < r_{k-1} < r_k \le n$. There are many possible selection of $R_{k,n}$. We



(a)

p_i	(i, u_i)	value
p_1	$(1, u_1)$	$(1, r_1)$
p_2	$(2, u_2)$	$(2, r_1)$
p_3	$(3, u_3)$	$(3, r_2)$
p_4	$(4, u_4)$	$(4, r_2)$
p_5	$(5, u_5)$	$(5, r_3)$
p_6	$(6, u_6)$	$(6, r_3)$
p_7	$(7, u_7)$	$(7, r_4)$
p_8	$(8, u_8)$	$(8, r_4)$
p_9	$(9, u_9)$	$(9, r_4)$
p_{10}	$(10, u_{10})$	$(10, r_5)$

(b)

Figure 6.2: (a): Mapping between full sequence T_n and key sequence $R_{k,n}$ (b): DTW mapping between the two sequences in (a)

want the one $R_{k,n}$ that is the most similar to the full sequence T_n. To achieve this, we will need a similarity measure for the two sequences.

If we apply DTW to T_n and $R_{k,n}$, the overall mapping cost would tell us how similar they are. To evaluate the cost, we start with constructing an alignment between the full sequence and the selected sequence, as shown in Figure 6.2(a).

The mapping is defined by $P = \{p_1, p_2, ..., p_n\} = \{(1, u_1), (2, u_2), ..., (i, u_i), ..., (n, u_n)\}$. Each (i, u_i) means time step i maps to time step u_i, where $u_i \in R_{k,n}$. In the example shown in Figure 6.2(a), All $n = 12$ mappings of P between the two sequences T_n and $R_{k,n}$ are listed in Figure 6.2(b). Since in our case each time step i in T_n has only a unique key time step r_i in $R_{k,n}$ to map to, we can simplify the more general bi-directional mapping mentioned in section 3.1 by a uni-directional mapping function $u_i = f(i)$, which is a special case of bi-directional mapping. On the other hand, a time step r_i in $R_{k,n}$ still may have multiple time steps in T_n to be mapped onto. From the example, it can be seen that time step 1 and time step 2 are mapped to r_1, so in the mapping $u_1 = u_2 = r_1$.

Let the dissimilarity between time step i and time step u_i be $D(i, u_i)$. The *DTW cost* of mapping between T_n and $R_{k,n}$, according to Equation 6.2, is defined as:

$$C_{k,n} = \min_{P \in \mathbb{P}} \left(\sum_{i=1}^{n} D(i, u_i) \right) \tag{6.3}$$

The specific P that gives the lowest mapping cost $C_{k,n}$ is the optimal mapping, denoted as P^*, for a given $R_{k,n}$.

Given a n-step time series, there are many different ways to choose k time steps, i.e. many possible $R_{k,n}$. If we simply exhaust all the possible $R_{k,n}$, and choose the one with the lowest DTW cost as the *optimal key sequence* or *optimal key time steps*, we have to evaluate:

$$R^*_{k,n} = \arg\min_{R_{k,n}} C_{k,n} = \arg\min_{R_{k,n}} \left(\min_{P \in \mathbb{P}} \left(\sum_{i=1}^{n} D(i, u_i) \right) \right) \tag{6.4}$$

Clearly, the amount of computation to find $R^*_{k,n}$ is going to be very large. To avoid the huge computation, in the next section we present an efficient approach to select the optimal key sequence.

6.3 Algorithm

To allow efficient selection of key time steps based on Equation 6.4, we need to place three constraints. First, each time step in the original time sequence maps to one and only one key time step. Also, since the key time steps in $R_{k,n}$ is to represent the full time sequence T_n, it does not make sense if the key time step does not represent itself. Therefore, our second constraint is that any key time step r_i in the selected sequence should be mapped to itself in the original time sequence, i.e., $u_{r_i} = r_i$, as shown in the blue lines in Figure 6.2(a). The third constraint is that the last time step in the original time sequence is always used as a key time step, i.e., $r_k = n$. In fact, if this is undesired, we can easily amend it by adding an artificial time step in the end of the full sequence that is very different from all the other time steps in T_n. We can select one more key time steps from the total number of $n + 1$ time steps, and discard this artificial step after $R^*_{k+1,n+1}$ is generated.

Given k key time steps, the full sequence is divided into k segments, as shown in Figure 6.3. Each segment $S_{i,j}$ is bounded by two key time steps, time step i and time step j, i.e. $S_{i,j} = \{i, i+1, ..., j-1, j\}$. Note that the first segment S_{1,r_1} always starts from the first time step and hence bounded only by one key time step, r_1.

Each segment $S_{i,j}$ represents a mapping to the key time steps i and j, and is associated with a cost, referred to as the *segment cost* and denoted as $\parallel S_{i,j} \parallel$. This cost is determined by DTW, as will be explained later. The overall cost of the mapping from the full sequence to the key time steps is the sum of the costs from all k segments, denoted as $C_{k,n}$, and can be computed as:

$$C_{k,n} = \parallel S_{1,r_1} \parallel + \sum_{i=1}^{k-1} \parallel S_{r_i,r_{i+1}} \parallel \tag{6.5}$$

Equation 6.5 in fact solves the same problem as Equation 6.3, except that the mappings in each segment can be solved independently because of the constraint that each key time step maps to itself. This constraint forces each time step to map to itself because DTW prohibits the mapping to contain crossing between the segments, which allows us to solve the DTW cost of each segment independently.

The result of the optimal key time steps selection is $R^*_{k,n}$, which gives the lowest $C_{k,n}$ value, that is:

$$R^*_{k,n} = \arg\min_{R_{k,n}} \left(\parallel S_{1,r_1} \parallel + \sum_{i=1}^{k-1} \parallel S_{r_i,r_{i+1}} \parallel \right) \tag{6.6}$$

Similarly, Equation 6.6 gives the same result as Equation 6.4 because all possible selections of key time steps $R_{k,n}$ are considered, and the optimal mapping is used within each segment.

The simplest but a brute force way to solve the above equation is to exam all the $\binom{n}{k}$ possible combinations of $R_{k,n}$, and then pick the one with the lowest $C_{k,n}$ value as the optimal key time steps. However, the number of trials, $\binom{n}{k}$, is going to be very large when n and k are large, which makes it impractical.

Actually, the DTW cost computations of different selection of $R_{k,n}$ are not independent. If two key sequences share a sub-sequence of key time steps, the DTW mapping within this sub-sequence is the same, and hence need not be computed more than once. Dynamic programming can be used to avoid this redundant computation, because it stores each step's result for later use. In the next section, we describe how dynamic programming can be used to solve the problem.

6.3.1 Dynamic Programming Selection

Dynamic programming divides a problem into sub-problems, and solve the sub-problem in the same way as the original problem until a stop condition is reached.

Suppose the mapping costs of all possible segments $S_{i,j}$ are known. We want to find the selection $R_{k,n}$ for the lowest $C_{k,n}$ value. Liu et al. proposed a dynamic programming to select key frames from video by maximizing the energy function [77].

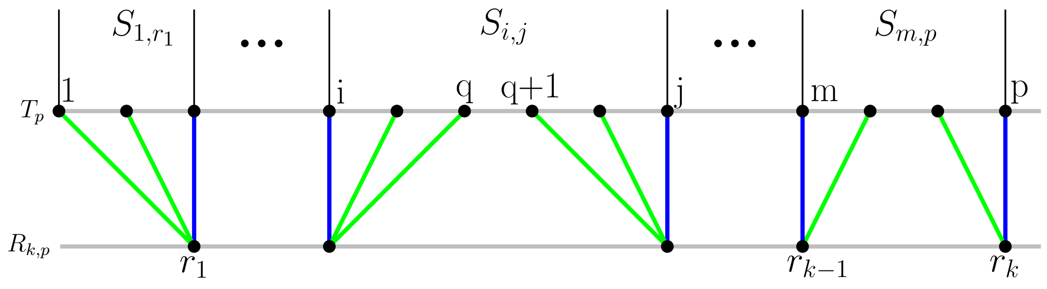


Figure 6.3: Segments in the mapping between T_p and $R_{k,p}$

Their method could be adapted to help us figure out the key time steps and minimize the DTW cost.

Let $T_p = \{1, 2, ..., p\}$ be the first p time steps. Let $R_{k,p} = \{r_1, r_2, ..., r_{k-1}, r_k\}$ be the selection of the k key time steps from T_p, and the corresponding mapping cost be $C_{k,p}$. Although we are dealing with only a subsequence T_p, the nature of the problem is essentially the same as finding the key time steps for the full sequence, where T_n, $R_{k,n}$ and $C_{k,n}$ are involved.

Suppose we have a constraint in T_p that the last key time step has to be time step p, i.e. $r_k = p$. Selecting k key time steps is then the same as selecting $k-1$ in the first m time steps, $m \in [k-1, p-1]$, and selecting one more from the rest. The overall mapping cost is the sum of the costs of two parts,

$$C_{k,p} = C_{k-1,m} + \parallel S_{m,p} \parallel \tag{6.7}$$

Let $R^*_{k,p}$ be the optimal selection for $R_{k,p}$, whose corresponding overall cost is $C^*_{k,p}$, the smallest among all possible $C_{k,p}$. We have the following optimal substructure: [77]

$$C^*_{k,p} = \begin{cases} \underset{m\in[k-1,p-1]}{min} \left(C^*_{k-1,m} + \parallel S_{m,p} \parallel\right) & \text{if } k > 2 \\ \underset{m\in[1,p-1]}{min} \left(\parallel S_{1,m} \parallel + \parallel S_{m,p} \parallel\right) & \text{if } k = 2 \end{cases} \tag{6.8}$$

To prove the correctness of Equation 6.8, it should be noted that if a key sequence is optimal, then its sub-sequence is also an optimal key time step selection for its corresponding subset of the full sequence. In other words, we could say if $R^*_{k,p} = \{r_1, r_2, ..., r_{k-1}, r_k\}$ is the optimal k key time steps selection from T_p, then its prefix $\{r_1, r_2, ..., r_{k-1}\}$ is also an optimal $k-1$ key time steps selection from $T_{r_{k-1}}$. This can be proved by contradiction: in the case of $k > 2$ in Equation 6.8, if $C^*_{k-1,m}$ is not the cost of the optimal mapping for the first m time steps, we could have replaced this sub-sequence by the optimal mapping with a lower $C_{k-1,m}$ value, and hence a lower $C_{k,p}$ from Equation 6.7, with a fixed $\parallel S_{m,p} \parallel$. This contradicts the facts that the mapping for $C^*_{k,p}$ is minimal.

In Equation 6.8, $C^*_{k,p}$ is the minimum of $C^*_{k-1,m}$ plus $\parallel S_{m,p} \parallel$ among the different selections of m. The $(k-1)$'th key time step in $R^*_{k,p}$ is

$$r^*_{k-1} = \begin{cases} \underset{m\in[k-1,p-1]}{argmin} \left(C^*_{k-1,m} + \parallel S_{m,p} \parallel\right) & \text{if } k > 2 \\ \underset{m\in[1,p-1]}{argmin} \left(\parallel S_{1,m} \parallel + \parallel S_{m,p} \parallel\right) & \text{if } k = 2 \end{cases} \tag{6.9}$$

In other words, r^*_{k-1} in Equation 6.9 is equal to the m that gives the minimum cost $C^*_{k,p}$ in Equation 6.8.

Based on Equations 6.8 and 6.9, the optimal key time steps $R^*_{k',p}$ can be solved by dynamic programming: We iteratively find k' key time steps, $k' = 2 \ldots k$, from the first p time steps $p = k' \ldots n$. Once the cost $C^*_{k',p}$ has been computed, this cost and the corresponding time step m will be stored in a 2D table indexed by k' and p. For $k' = 2$, the cost only relies on the segment mapping costs $\parallel S_{1,m} \parallel$ and $\parallel S_{m,p} \parallel$, $m = k'-1, \ldots, p-1$. For $k' > 2$, the cost is based on $C^*_{k'-1,m}$, $m = k'-1, \ldots, p-1$, which have been solved. Once the cost $C^*_{k,p}$ is obtained, the key time steps $R^*_{k,p}$ can be found by backtracking the time step stored in the table.

6.3.2 Segment Mapping Cost

To solve the above problem, we need to know the cost $\| S_{i,j} \|$. It can be defined differently for different use. In Liu's work, an energy function [77] is used to define the cost. Here to solve our problem of optimal mapping, we define $\| S_{i,j} \|$ as the cost of mapping from the sequence $S_{i,j} = \{i, ..., j\}$ to the key sequence $\{i, j\}$.

As proved in Section 6.3.1, if a key sequence is optimal, its sub-sequence is also an optimal selection of key time steps. Consequently, the mapping within each segment is also optimal. Given a dissimilarity metric $D_{i,j}$ for a pair of time steps i and j, which will be defined later. The mapping cost $\| S_{i,j} \|$ can be computed as the sum of the optimal mapping costs within the segment $S_{i,j}$.

$$\| S_{i,j} \| = \min_{i \leq q < j} \left(\sum_{t=i}^{q} D_{i,t} + \sum_{t=q+1}^{j} D_{t,j} \right) \tag{6.10}$$

Time step q is a time step between time step i and time step j, as shown in Figure 6.3. Essentially the formula above computes the cost $\| S_{i,j} \|$ as the minimum cost for mapping the time steps before q to i, and the rest time steps to j among all the possible $q \in [i, j)$.

It should be noted that in certain simulation, the first a few time steps can have very small value during the initialization of the model. If the initial time step is undesired, the algorithm can be modified in order not to always choose the first time step. The basic idea behind this modification is to treat the first segment as a special case. Given the first segment from time steps 1 to j, its cost $\| S_{1,j} \|$ is computed by mapping all time steps to the j-th time step alone, other than both time steps 1 and

j. i.e.

$$\| S_{1,j} \| = \sum_{t=1}^{j} D_{t,j} \tag{6.11}$$

Hereafter this special treatment is applied to all case studies.

For all the other cases when $S_{i,j}$ is not the first segment, $S_{i,j}$ is divided into two parts $S_{i,q}$ and $S_{q+1,j}$ by time step q, where the time steps in $S_{i,q}$ map to time step i, and the time steps in $S_{q+1,j}$ map to the last time step j, as shown in Figure 6.3. We define time step q as the *jump time step*, because at the point of time step q, each element in $S_{i,j}$ changes from mapping to time step i to mapping to time step j. No crossing is allowed in the mapping in our case considering the temporal coherence. $\| S_{i,j} \|$ is the minimum of the sum of the mapping cost in $S_{i,q}$ and $S_{q+1,j}$, for different choice of q. We may also view the solution to Equation 6.10 as finding the DTW warping between two sequences $S_{i,j}$ and $\{i, j\}$.

If the total number of time steps is n, then there are $\frac{n(n-1)}{2}$ pairs of (i, j), or possible segments. If we solve the cost for all the $\frac{n(n-1)}{2}$ segments, the results can be filled into a triangular matrix, which can be stored as input to the main algorithm described in Section 6.3.1.

Finally, $D_{i,j}$ is the dissimilarity metric between time step i and j. The metric can be defined differently for different goal. For example, if our focus is to compare two isosurfaces in two time steps, we can use the isosurface similarity map [16]. If we want to compare the distributions in two time steps, we can use Earth Mover's Distance [96] or K-L divergence [68]. Our algorithm does not depend on a particular kind of dissimilarity metric, so can be broadly applied to different problems.

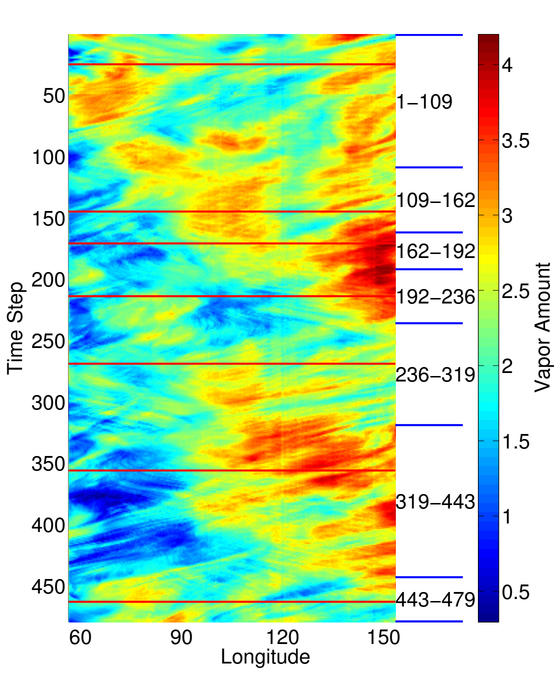
6.4 Results

To test the efficacy of our algorithm, two case studies were conducted. The first case study is based on the data generated from a simulation of Madden-Julian Oscillation, a well known weather phenomenon, and the second case study is on the analysis of radiation from an astrophysics simulation.

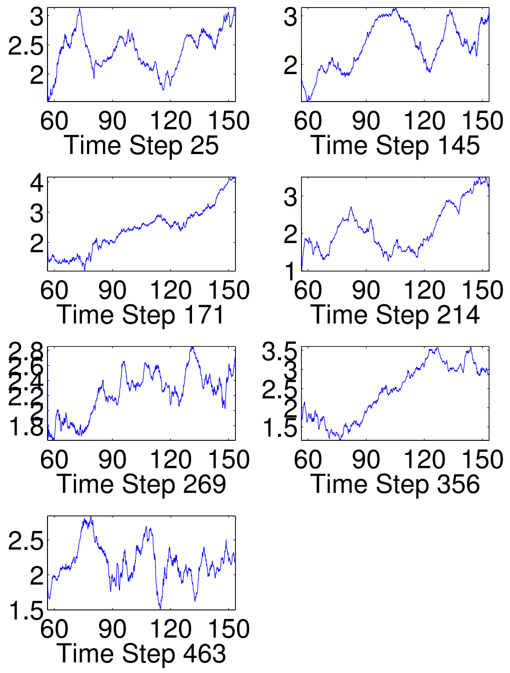
6.4.1 Madden-Julian Oscillation

Madden-Julian Oscillation (MJO) is a weather phenomenon that consists of 30-60 days oscillation of surface and upper level winds in the tropical area near Indian and Pacific oceans. MJO can be characterized by an eastward progression of both enhanced and suppressed tropical rainfall. Study of MJO is important because it explains intra-seasonal variability in the tropics. MJO also influences the precipitation during the summer months in North America. The data set of the simulation consists of 479 time steps, where each time step contains of $2699 \times 599 \times 27$ voxels. The simulation was done by scientists in the Pacific Northwest National Laboratory [51].

Scientists usually observe MJO by Hovmoller diagrams generated from different variables, such as cloud intensity and water vapor. Figure 6.4(a) shows a Hovmoller diagram of the water vapor mixing ratio at the 583hPa pressure level . In the diagram, the X axis represents longitude, and the Y axis represents time. Each pixel in the diagram represents the average water vapor from points of different latitudes but a constant longitude and time step. Hovmoller diagram is commonly used for plotting meteorological data to highlight time progression of certain phenomena over a given spatial region [55]. The limitation of Hovmoller diagram is that it can only show a single spatial coordinate, e.g. longitude or latitude, at a time. If one desires to see the



(a)



(b)

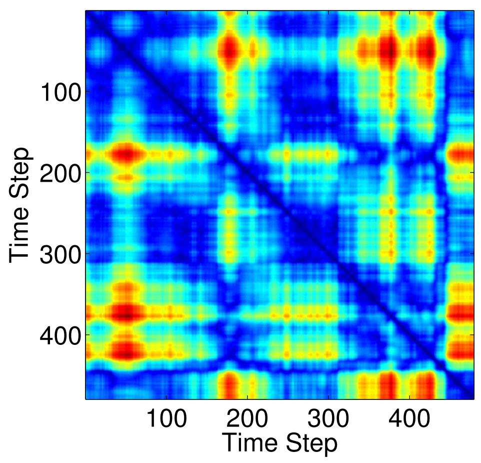
Figure 6.4: (a): Water vapor mixing ratio on Hovmoller diagram. (b): Water vapor mixing ratio on 7 key time steps marked as red lines in (a).

data in multiple dimensions, 2D or 3D space for example, Hovmoller diagram will be insufficient. In this case study, we use the sequence of scanlines from the Hovemoller diagram shown in Figure 6.4(a) as the time-varying data input. We detect salient time steps using our algorithm so that scientists can perform further analysis of the data based on the key time steps.

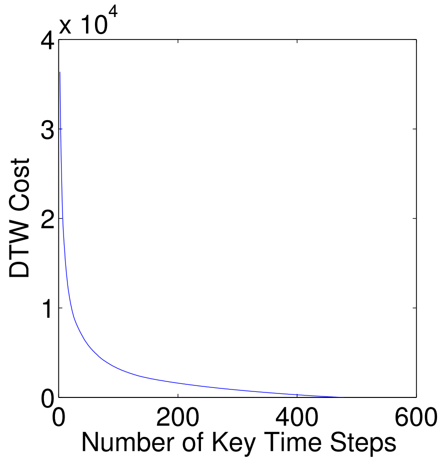
MJO can be observed from cloud movement along the longitude direction, and the cloud intensity is associated with water vapor mixing ratio. For this reason, we use the time-varying water vapor intensity data over a range of longitudes as the input to our algorithm to identify key time steps. We use the Earth Mover's Distance(EMD) to measure the dissimilarity of water vapor distribution between different time steps [96]. The EMD measures the minimum amount of work to fill a mass of earth into a collection of holes in the same space. The work is the sum of each unit of earth times the ground distance that it moves. EMD will assign a larger dissimilarity value for two time steps if their peaks of water vapor are far away.

Figure 6.5(a) shows a dissimilarity matrix among the time steps using EMD for the water vapor data, where blue represents low and red represents high values. The large blue squares in the dissimilarity matrix along the diagonal line indicates there exists temporal coherence for data in adjacent time steps. Figure 6.5(b) shows the relationship between the DTW cost and the number of key time steps selected by our algorithm. It can be seen that the DTW cost drops sharply as more key time steps are selected. Applying our key time steps selection algorithm, we selected 7 key time steps. The plots of water vapor mixing ratio versus longitude for the key time steps are shown in Figure 6.4(b).

From these plots, we can see the movement of MJO over time. In the plots, MJO can be seen as peaks of the water mixing ratio, which is initially on the west (left in the plot) side of the domain. The peaks then move to the middle in key time step 2, and go to east with a large water vapor value in key time step 3. Before this MJO has not fully left the domain, a second MJO is formed in the east again, as shown in key time step 4. This is expected because MJO is a cyclic weather event. It can be seen that this peak moves to the middle in key time step 5, and to east in key time step 6. The small peak on the left of key time step 7 may indicate the formation of a third MJO.



(a)



(b)

Figure 6.5: (a): Water vapor EMD dissimilarity map. (b) DTW cost with increasing number of key time steps for MJO data set.

In Figure 6.4(a), the selected key time steps are marked over the Hovmoller diagram by red lines, and the jump time steps are marked on the right side of Hovmoller diagram in blue. The distributions of the time steps between two adjacent jump time steps are very similar, and are represented by the key time step between them. There are more key time steps in the area of larger variance around time step 170, and fewer key time steps in the area of lower variance around time step 80 and time step 370.

6.4.2 Radiation of Astrophysics Turbulence

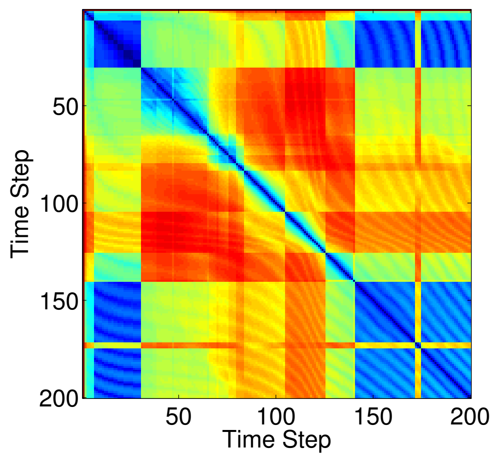
The astrophysics turbulence data set is from a three-dimensional radiation hydrodynamical simulation of ionization front instabilities [120]. The data set consists of 200 time steps, where each time step is a $600 \times 248 \times 248$ point regular mesh. The first star emits energetic UV radiation to the universe. The radiation ionized the surrounding gas to a temperature around $20,000K$. The distribution of the hot gas is interesting to the scientists, so we visualize the data with the isosurface of the temperature at $20,000K$, and see how it evolves change over time.

Among the 200 time steps in the dataset from which isosurfaces are computed, many of them look quite similar. This information redundancy makes it difficult for the user to focus on only the most essential information, in this case, how does the isosurface evolve. To assist the user, we apply our algorithm where the focus is to compare the isosurfaces and choose the most representative ones.

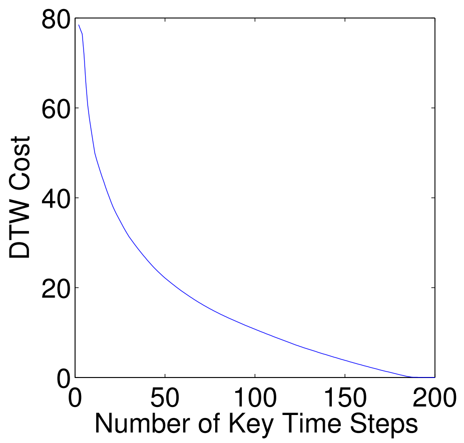
An important input to our algorithm is the dissimilarity between isosurfaces of all pairs across the entire time sequence. For this, we use *isosurface similarity map* metric presented in [16]. In essence, this metric computes the mutual information between distributions of two distance fields derived from the isosurfaces. In our application, we use the similarities of all pairs of isosurfaces to build the 200×200 isosurface similarity map. After normalizing the similarity to a $[0, 1]$ range, the value of one minus similarity is used to build the dissimilarity map input for our key time step selection algorithm.

The dissimilarity map is visualized in Figure 6.6(a). Blue color represents low dissimilarity, and red color represents high dissimilarity. The entries around the diagonal are mostly blue, which means the isosurface of one time step is similar to the isosurfaces in its adjacent time steps. Several different sized blue squares can be seen along the diagonal of dissimilarity map. Each square shows a group of similar isosurfaces.

Based on the dissimilarity map, we ran our key time steps selection program and selected 6 key time steps. Table 6.1 shows information about the selection results. In this table, each row is a segment $S_{i,j}$. It means that the time steps in this segment,



(a)



(b)

Figure 6.6: (a): Radiation isosurface dissimilarity map. (b) DTW cost drops with increasing number of key time steps for radiation data set.

Table 6.1: 6 key time steps and the mapping for radiation isosurfaces

p^*	i	j	characteristics
24	1	30	one plane
48	31	65	a clear circle in front of the plane
78	66	83	small isosurfaces in front and back of the circle
93	84	104	small isosurfaces only in front of the circle
113	105	125	a broken cylinder in front of the plane
141	126	200	possible small isosurfaces in front of the plane

time step i to time step j, map to the key time step, time step p^*, where $i \leq p^* \leq j$. The segments listed in Table 6.1 show a good correspondence with the blue squares on the isosurface dissimilarity map in Figure 6.6(a). Thus, our algorithm is capable of making temporal classification of similar isosurfaces in a time-varying data set.

We visualized the 6 isosurfaces in Figure 6.7. Each of the 6 isosurfaces is representative for its corresponding time segment. We can see that they are quite different

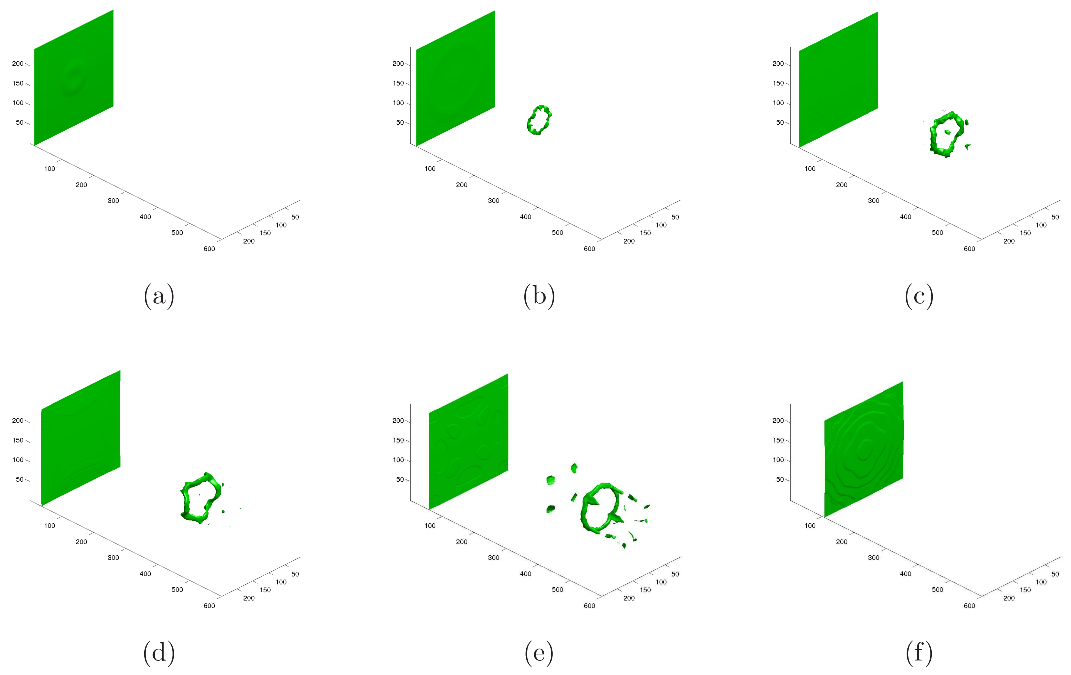


Figure 6.7: $20,000K$ isosurfaces of temperature field of 6 key time steps

from each other, and each has its unique characteristics, as described in Table 6.1. The selected isosurfaces give the user a clear overview of how the isosurfaces evolve.

The relationship between the DTW cost and the number of key time steps selected is shown in Figure 6.6(b). The DTW cost does not drop as sharply as the one in our previous case study, the MJO data set, which is an indication that we may need more key time steps to represent the whole data set. However, limited by the screen space, we could not visualize too many key time steps at a time. In Section 6.5, we will describe an interactive key time steps browser, taking advantage of our algorithm, to explore time-varying data hierarchically.

6.4.3 Performance

A prototype of our algorithm was tested on a machine with Intel Core i7 2600 CPU, 16GB system memory, and nVidia GeForce GTX 560 GPU. The performance tests were performed in 2012. It can be higher if newer processor and graphics card had been used.

The dissimilarity matrix from the time-varying data is precomputed and used as the input to the dynamic programming algorithm. The performance and scalability of this preprocessing step depends on the dissimilarity metric we use. For the data set MJO, the dissimilarity matrix is computed based on the approximated EMD between the rows in the Hovemoller diagram, where each row represents data across different longitudes at a particular time step. Because data at each time steps are aggregated to a function of longitude, the computation of the dissimilarity matrix only took 25.6 seconds, including 24.2 seconds to generate the Hovemoller diagram from raw NetCDF data, and 1.4 seconds to compute EMD from the Hovemoller diagram using MATLAB. For the radiation data set, the dissimilarity computation first evaluates the distance from each grid point to each isosurface, which took 4.64 hours on GPUs for data at a full resolution. The mutual information computation is time-consuming, too, since joint histograms need to be constructed. For each pair of time steps, the mutual information is computed by scanning all voxels in their distance fields, which totally took 8.4 hours for 200 time steps. Thus, the total preprocessing time for radiation data set is $4.64 + 8.5 = 13.14$ hours.

The implementation of the dynamic programming algorithm has two major stages. The first stage is to run our dynamic programming algorithm to generate the time step index table. Given n time steps, the time complexity of our dynamic programming is

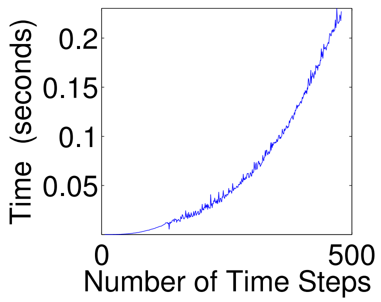


Figure 6.8: Performance of key time step selection for MJO.

$O(n^3)$. Figure 6.8 shows the time required to select different numbers of time steps from the MJO dataset. It can be seen that the algorithm took less than 0.25 second for all cases. Because the computation can be done very efficiently, the user can select different sub time interval and then re-compute the table interactively. Once the time step index table has been computed, the next stage is to select the key time steps of any user-desired number. Because the task of this stage essentially is to scan the table, the time spent on it is totally negligible compared to other stages. This allows the user to query different number of key time steps in real time.

6.5 Key Time Steps based Time Varying Data Browser

To enable interactive analysis of time-varying data sets, we design a data browser that can take advantage of the information computed from our key time step selection algorithm. The key time steps and the mapping between the full sequence and the key sequence allow the user to navigate time-varying data at different levels of temporal detail. In the browser, we show information related to the nature of the time-varying data, including the dissimilarity matrix, the key time steps, the jump time steps, and the warping path between the full sequence and the key sequence. The data browser

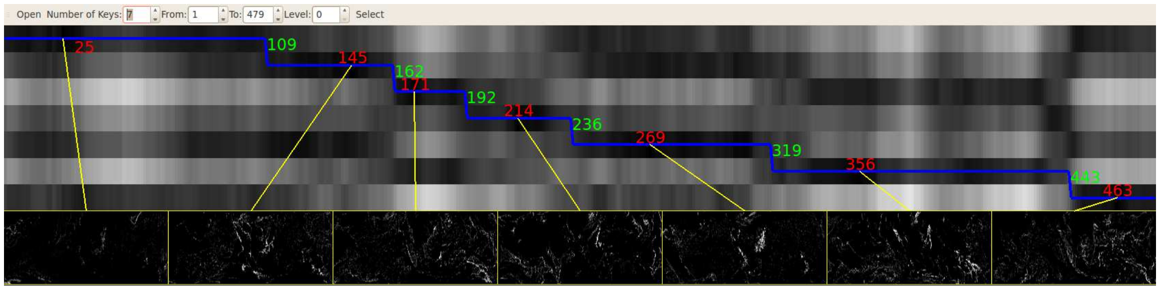


Figure 6.9: Time-varying data browser. Here 7 key time steps from all 479 time steps of the MJO data are selected. The top 7×470 2D map show the rows of the selected time steps from the original dissimilarity map. The data of the key time steps are rendered below the 2D map.

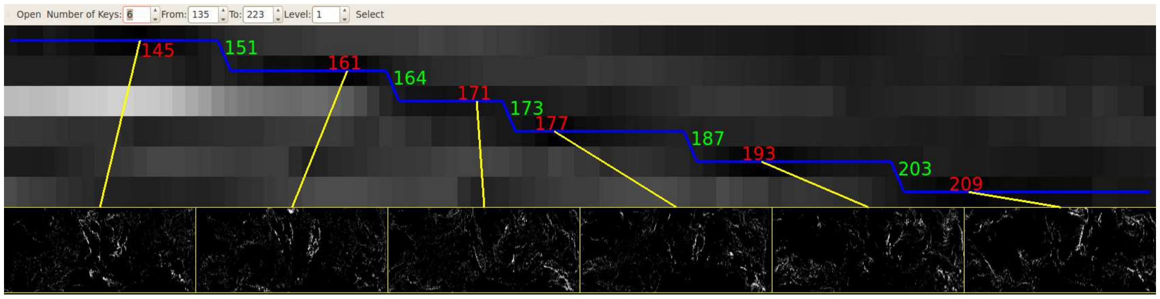


Figure 6.10: Zooming into a time interval in the browser. Here 6 key time steps are selected from the time step interval [135, 223] of the MJO data.

allows the user to interactively specify the desired number of key time steps, across the entire time sequence or within a local time segment. Figure 6.9 shows a snapshot of our system.

6.5.1 User Interface and Visual Display

A key component of this interface is the visualization of the dissimilarity matrix. It is visualized as a 2D greyscale image in the background. Each rectangle represents an entry of the matrix, where black represents low dissimilarity values, and white represents high dissimilarity values. Let n be the total number of time steps, and k be the number of key time steps. In the dissimilarity matrix viewer, we do not show the whole $n \times n$ dissimilarity matrix, but only the rows that correspond to the selected key time steps versus the full sequence in the columns, i.e. a $k \times n$ dissimilarity matrix. This matrix shows the dissimilarities between the key sequence and the full sequence.

From the results of key time steps selection, we establish a mapping between the key sequence and the full sequence. This mapping can be represented as a path in the dissimilarity matrix, shown as the blue lines in Figure 6.9. The warping path is composed of multiple horizontal lines segments. Each path segment is a mapping between one key time step to all its represented time steps.

In Figure 6.9, the key time step of each path segment is marked by a red time step index. The warping path between a pair of adjacent key time step is a time segment mentioned earlier in the algorithm section. Between a pair of adjacent key time steps, the warping path jumps from one row to the next row at the jump time step, which is marked by a green time step index. In order to provide detailed information for the time varying data set, visualization of data in the key time steps is shown below the dissimilarity map, each connected with the corresponding key time step shown on the warping path by a yellow line.

With this user interface, to browse the time varying data at different levels of detail, the user can interactively input the time interval of interest, and specify the number of desired key time steps. From the key time steps, the user may zoom in to a smaller time interval. They can repeat this process until the time steps that contain salient features are found.

6.5.2 Use Case for the MJO Data Set

Here we use the MJO data set described earlier in Section 6.4.1 to demonstrate the use of our browser. Given the pre-computed dissimilarity map, we pick 7 key time steps with our algorithm from the whole sequence, i.e., from time step 1 to time step 479.

Shown in Figure 6.9, the background is an image that shows a 7×479 dissimilarity matrix. 7 key time steps and 6 jump time steps are marked along the warping path. The 2D cloud intensity on the pressure level 850hPa for the 7 key time steps are displayed below the dissimilarity matrix, connected to their corresponding key time steps by yellow lines.

Since our algorithm run very efficiently after the dissimilarity map is computed, the selection of key time steps can be done interactively. By viewing the 2D cloud renderings at the key time steps, we found that time step 145 has high cloud intensity in the middle, which later moves to the southeast at time step 171. In time step 214, a second strong cloud appears in the southwest.

Assuming we are interested in the data around the three time steps mentioned above, we can drag an interval on the dissimilarity map, e.g. from time step 135 to time step 223. To get more details, we can double the number of key time steps in this interval to 6 key time steps. The new results are shown in Figure 6.10. Time step 145 and 171 are still the key time steps, with an additional key time step added in between. Time step 214, a key time step in the previous selection, is replaced by another 3 time steps, 177, 193, and 209. The new key time step, time step 161, shows that before the middle point cloud moves to southeast, it first goes eastbound for some distance. Time steps 177, 193, and 209 show that this cloud wave stays at the east with a high intensity for a long time, and the second cloud wave is gradually formed in the southeast. We may further reduce the size of the time interval based on our interested key time steps to extract more detailed information of the MJO data set.

To summarize, with our system, the user can first obtain an overview of the entire data set with a small number of key time steps. From the key time steps, interesting time intervals can be selected where additional key time steps can be generated recursively until the desire features are found.

6.6 Conclusion

We present a novel technique for selecting key time steps from large scale time-varying data sets. Our goal is to identify a subsequence from the entire time steps that can give a globally minimum cost. To achieve this goal, we apply Dynamic Time Warping to establish the mapping between the full and the sub-sequences, and choose the best subsequence among all the possible choices as the key time steps. The dynamic programming scheme we developed is very efficient, and thus can facilitate interactive data browsing. A time-varying data browsing system is designed to allow exploring the time varying data interactively in different levels of temporal scales. The algorithm was tested on a Madden-Julian Oscillation simulation data set using water vapor distributions over areas of different longitude. We also tested our algorithm on an astrophysics turbulence data set to select salient isourfaces over a long time sequence.

The main limitation of our work is that computing the dissimilarity matrix requires all time steps. Besides, because the complexity of distance computation for n

time steps is $O(n^2)$, the distance computation can dominate the preprocessing stage. One example is our case study for radiation dataset, which took 4 hours on GPUs. Another direction of our future work is to enhance our time-varying data browser. The current browser only displays a submatrix of the dissimilarity matrix. We believe that displaying the entire dissimilarity matrix to the user can provide additional useful information. For instance, by highlighting the time segments on the matrix, the user can visually evaluate whether the segments are sufficient to represent the whole data. Besides, the current design uses most of the screen space to display the dissimilarity matrix. Our new design will give more space to display the data in the selected time steps. This will allow the user to clearly compare the selected time steps side-by-side. One possible design is to shrink the dissimilarity matrix and arrange the time steps around the matrix. Also, the browser will have more user control. For instance, while currently the intensity of the dissimilarity map are automatically scales based on the current range of distance scope, the future design will allow the user to control the color mapping.

Chapter 7: Contributions and Future Works

Visual clutter problem comes along with big data and will change as the data size grows and data contents change. This dissertation provides solutions to solve visual clutter and demonstrate the solutions using flow visualization, glyph-based visualization, and time-varying data visualization. In scientific visualization, there are more scientific data types and visualization techniques than what were present in this dissertation. One particular visual clutter management technique cannot solve visual clutter in all kind of scientific data or all visualization techniques. However, our general strategies described in Section 1.2 should generalize most visual clutter management methods. By following these general strategies, one can design more visual clutter management methods to solve the visual clutter in specific applications.

We categorize our future works into three categories and describe them as the following items:

1. Deformation technique

   (a) We want to compare our deformation technique with more of other occlusion removal techniques and perform user studies.

   (b) We can combine our deformation technique with other techniques, such as transparency and cutaway, to solve the occlusion problems in different situations. For example, we can use transparency on the streamlines whose shapes cannot be well preserved by deformation.

   (c) For the object space deformation model, we can allow users to interactively cut the tetrahedral mesh and break a hole on the mesh. In this way, the force from the lens can enlarge the hole to allow the users to see through it.

   (d) In order to work with time-varying datasets, our deformation model can be designed to work with an animation of streamlines or glyphs.

   (e) Besides solving visual clutter in 3D visualizations, this strategy can also be used to tackle visual clutter in 2D visualization, such as parallel coordinates plots (PCP). The difference between 2D and 3D is whether to consider depth (or layer) information of geometries.

2. Interactive Visualization System

   (a) More depth cues can be provided in the visualization of deforming geometries.

   (b) The lens shape can be more flexible in order to adapt to different shapes of focus regions.

   (c) To allow focus streamlines properly distributed in different regions of the image space, the system should allow deformation in multiple focus regions simultaneously.

   (d) Multiple users can interact with lenses simultaneously using different input and output devices.

   (e) The system should automatically suggest positions of interesting features. Users can save time if being guided on where to place the lens.

3. Immersive Visualization and Interaction

   (a) Occlusion should be removed in stereoscopic displays. For viewing stereoscopic images of visualization, user's two eyes see different images. Ensuring the visibility of focus features on both images can benefit the stereoscopic display quality.

   (b) View navigation in virtual reality is challenging. It is easy to become lost in a virtual environment when viewing data too closely. A side window showing the viewer's position in the spatial domain and the direction of viewing can be very helpful.

   (c) Accuracy and stability of hand tracking should be increased. More interaction devices should be explored to control the lens. Hand motion camera, such as Leap Motion, is limited by its mechanism and does not recognize particular gestures or hand positions. Other controllers, such as HTC VIVE controllers, provide more accurate and stable tracking and may be a better alternative to manipulate our lens.

   (d) Visual and haptic feedback from manipulated objects should be enhanced in order to allow users to sense the existences of the objects. For example, when a hand moves close to an object, the object changes color or the controller vibrates. This feedback can increase the accuracy of lens placement.

4. Others

(a) For crystal glyph, the spherical base is not always easy to see, especially when the distribution spreads in a big range of directions. Then, users lack reference to observe the terrain heights and the corresponding probability densities. Overcoming this limitation is a future goal.

(b) For the key time step selection work, we want to modify the algorithm to make the dynamic programming algorithm work with a subset of the dissimilarity matrix. In such a case, we do not need to access the entire dataset, and thus our algorithm can be extended for in situ data reduction as well.

We hope the general strategies and the presented techniques of visual clutter management in scientific visualization can help viewers to better understand and solve their visual clutter problems. We also wish the general strategies and the mentioned future works can inspire researchers to study more along this direction and tackle more specific problems.

Bibliography

[1] 3D Slicer, a free, open source software package for visualization and image analysis. http://www.slicer.org.

[2] Cgal, computational geometry algorithms library. http://www.cgal.org.

[3] VIS Contest 2016, http://www.uni-kl.de/sciviscontest/.

[4] Teem toolkit, version 1.11.0, December 2012. http://teem.sourceforge.net/index.html.

[5] Hiroshi Akiba, Nathaniel Fout, and Kwan-Liu Ma. Simultaneous classification of time-varying volume data based on the time histogram. In *Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis) 2006*, pages 171–178, 2006.

[6] Hiroshi Akiba and Kwan-Liu Ma. A tri-space visualization interface for analyzing time-varying multivariate volume data. In *Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis) 2007*, pages 115–122, 2007.

[7] Michael P. Allen. Introduction to molecular dynamics simulation, 2004.

[8] Ryan Bane and Tobias Hollerer. Interactive tools for virtual X-ray vision in mobile augmented reality. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '04, pages 231–239, Washington, DC, USA, 2004. IEEE Computer Society.

[9] Peter J Basser, Sinisa Pajevic, Carlo Pierpaoli, Jeffrey Duda, and Akram Aldroubi. In vivo fiber tractography using dt-MRI data. *Magnetic Resonance in Medicine*, 44(4):625–632, 2000.

[10] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 73–80, New York, NY, USA, 1993. ACM.

[11] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, June 1991.

[12] Rita Borgo, Johannes Kehrer, David H.S. Chung, Eamonn Maguire, Robert S. Laramee, Helwig Hauser, Matthew Ward, and Min Chen. Glyph-based visualization: Foundations, design guidelines, techniques and applications. *Eurographics State of the Art Reports*, May 2013.

[13] E. Boring and A. Pang. Directional flow visualization of vector fields. In *Proceedings of IEEE Visualization 1996*, pages 389–392, Oct 1996.

[14] Paul Bourke. Converting to and from 6 cubic environment maps and a spherical map. Technical report, May 2006.

[15] Andrea Brambilla. *Visibility-oriented Visualization Design for Flow Illustration*. PhD thesis, Department of Informatics, University of Bergen, Norway, December 2014.

[16] Stefan Bruckner and Torsten Möller. Isosurface similarity maps. In *Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis) 2010*, pages 773–782, 2010.

[17] T. Butkiewicz and A.H. Stevens. Effectiveness of structured textures on dynamically changing terrain-like surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):926–934, January 2016.

[18] M. S. T. Carpendale, D. J. Cowperthwaite, and F. D. Fracchia. Distortion viewing techniques for 3-dimensional data. In *Proceedings of IEEE Information Visualization (InfoVis) 1996*, pages 46–53, October 1996.

[19] M. S. T. Carpendale and Catherine Montagnese. A framework for unifying presentation space. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST) 2001*, pages 61–70, 2001.

[20] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-D rotation using 2-D control devices. *SIGGRAPH Computer Graphics*, 22(4):121–129, June 1988.

[21] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[22] I. Corouge, S. Gouttard, and G. Gerig. Towards a shape model of white matter fiber bundles using diffusion tensor MRI. In *Proceedings of IEEE International*

Symposium on Biomedical Imaging: Nano to Macro 2004, volume 1, pages 344–347, April 2004.

[23] C. Correa, D. Silver, and M. Chen. Illustrative deformation for data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1320–1327, November 2007.

[24] Roger Crawfis and Nelson Max. Direct volume visualization of three-dimensional vector fields. In *Proceedings of the 1992 Workshop on Volume Visualization*, VVS '92, pages 55–60, New York, NY, USA, 1992. ACM.

[25] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 135–142, New York, NY, USA, 1993. ACM.

[26] J. Cui, P. Rosen, V. Popescu, and C. Hoffmann. A curved ray camera for handling occlusions through continuous multiperspective visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1235–1242, November 2010.

[27] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2005*, volume 1, pages 886–893, June 2005.

[28] W. C. de Leeuw and J. J. van Wijk. A probe for local flow field visualization. In *Proceedings of IEEE Visualization 1993*, pages 39–45, October 1993.

[29] J. Diepstraten, D. Weiskopf, and T. Ertl. Transparency in interactive technical illustrations. *Computer Graphics Forum*, 21(3):317–325, 2002.

[30] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive cutaway illustrations. *Computer Graphics Forum*, 22(3):523–532, 2003.

[31] D. Dovey. Vector plots for irregular grids. In *Proceedings of IEEE Visualization 1995*, pages 248–253, October 1995.

[32] N. Elmqvist and P. Tsigas. A taxonomy of 3D occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1095–1109, September 2008.

[33] Zhe Fang, Torsten Möller, Ghassan Hamarneh, and Anna Celler. Visualization and exploration of time-varying medical image data sets. In *Proceedings of the Conference on Graphics Interface (GI) 2007*, pages 281–288, 2007.

[34] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J.V. Miller, S. Pieper, and R. Kikinis. 3D slicer as an image computing platform for the quantitative imaging network. *Magnetic Resonance Imaging*, 30(9):1323–1341, 2012. cited By 189.

[35] S. K. Feiner and D. D. Seligmann. Cutaways and ghosting: satisfying visibility constraints in dynamic 3D illustrations. *The Visual Computer*, 8(5):292–302, 1992.

[36] P. Fischer, J. Lottes, D. Pointer, and A. Siegel. Petascale algorithms for reactor hydrodynamics. *Journal of Physics Conference Series*, 125(1):012076, July 2008.

[37] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, November 1991.

[38] A. Fuhrmann and E. Groller. Real-time techniques for 3D flow visualization. In *Proceedings of IEEE Visualization 1998*, pages 305–312, Oct 1998.

[39] G. W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 1986*, pages 16–23, 1986.

[40] E.R. Gansner, Y. Koren, and S.C. North. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):457–468, July 2005.

[41] Jorge Gascon, Jose M. Espadero, Alvaro G. Perez, Rosell Torres, and Miguel A. Otaduy. Fast deformation of volume data using tetrahedral mesh rasterization. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, pages 181–185, New York, NY, USA, 2013. ACM.

[42] Rocco Gasteiger, Mathias Neugebauer, Oliver Beuing, and Bernhard 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, December 2011.

[43] O. Goksel and S. E. Salcudean. B-mode ultrasound image simulation in deformable 3-D medium. *IEEE Transactions on Medical Imaging*, 28(11):1657–1669, November 2009.

[44] N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[45] C. P. Gribble, T. Ize, A. Kensler, I. Wald, and S. G. Parker. A coherent grid traversal approach to visualizing particle-based simulation data. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):758–768, July 2007.

[46] Christiaan P. Gribble and Steven G. Parker. Enhancing interactive particle visualization with advanced shading models. In *Proceedings of the 3rd Symposium on Applied Perception in Graphics and Visualization*, APGV '06, pages 111–118, New York, NY, USA, 2006. ACM.

[47] Edward Grundy, Mark W. Jones, Robert S. Laramee, Rory P. Wilson, and Emily L.C. Shepard. Visualisation of sensor data from animal movement. *Computer Graphics Forum*, 28(3):815–822, 2009.

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

[49] Tobias Günther, Christian Rössl, and Holger Theisel. Opacity optimization for 3D line fields. *ACM Transactions on Graphics*, 32(4):1201–1208, July 2013.

[50] Hongbin Guo, Rosemary Renaut, Kewei Chen, and Eric Reiman. Clustering huge data sets for parametric PET imaging. *BioSystems*, 71(1–2):81–92, 2003.

[51] Samson Hagos and L. Ruby Leung. Moist thermodynamics of the madden-julian oscillation in a cloud resolving simulation. *AMS Journal of the Atmospheric Sciences*, 24(21):5571–5583, April 2011.

[52] B. Harris. Entropy. In *Encyclopedia of Statistical Sciences (2nd Edition, vol.3)*. John Wiley & Sons, Inc., 2006.

[53] M. Hlawatsch, P. Leube, W. Nowak, and D. Weiskopf. Flow radar glyphs - static visualization of unsteady flow with uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1949–1958, December 2011.

[54] Jared Hoberock and Nathan Bell. Thrust: a parallel template library, 2010.

[55] Ernest Hovmöller. The trough-and-ridge diagram. *Tellus*, 1(2):62–66, 1949.

[56] C. Hurter, R. Taylor, S. Carpendale, and A. Telea. Color tunneling: Interactive exploration and selection in volumetric datasets. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2014*, pages 225–232, March 2014.

[57] Christian Hüttig and Kai Stemmer. The spiral grid: A new approach to discretize the sphere and its application to mantle convection. *Geochemistry, Geophysics, Geosystems*, 9(2), 2008. Q02018.

[58] N. Ihaddadene and C. Djeraba. Real-time crowd motion analysis. In *Proceedings of 19th International Conference on Pattern Recognition*, pages 1–4, December 2008.

[59] B. Jackson, T. Y. Lau, D. Schroeder, K. C. Toussaint, and D. F. Keefe. A lightweight tangible 3D interface for interactive visualization of thin fiber structures. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2802–2809, December 2013.

[60] Mihaela Jarema, Ismail Demir, Johannes Kehrer, and Rdiger Westermann. Comparative visual analysis of vector field ensembles. In *Proceedings of IEEE Conference on Visual Analytics Science and Technology (VAST) 2015*, 2015.

[61] Fangxiang Jiao, J.M. Phillips, Y. Gur, and C.R. Johnson. Uncertainty visualization in hardi based on ensembles of odfs. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2012*, pages 193–200, February 2012.

[62] Gordon Kindlmann. Superquadric tensor glyphs. In *Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis) 2004*, VISSYM'04, pages 147–154, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[63] R. M. Kirby, H. Marmanis, and David H. Laidlaw. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *Proceedings of IEEE Visualization 1999*, VIS '99, pages 333–340, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[64] Tijmen Klein, Florimond Gu&#x00e9;niat, Luc Pastur, Fr&#x00e9;d&#x00e9;ric Vernier, and Tobias Isenberg. A design study of direct-touch interaction for exploratory 3d scientific visualization. *Computer Graphics Forum*, 31(3pt3):1225–1234, June 2012.

[65] S. Kottravel, M. Falk, E. Sundn, and T. Ropinski. Coverage-based opacity estimation for interactive depth of field in molecular visualization. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2015*, pages 255–262, April 2015.

[66] J. Krüger, J. Schneider, and R. Westermann. Clearview: an interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):941–948, September 2006.

[67] Jrg Kuhnert. Meshfree numerical schemes for time dependent problems in fluid and continuum mechanics. In S. Sudarshan, editor, *Advances in PDE modeling and computation*, pages 119–136. New Delhi: Ane Books, 2014.

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

[69] O. H. Kwon, C. Muelder, K. Lee, and K. L. Ma. A study of layout, rendering, and interaction methods for immersive graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1802–1815, July 2016.

[70] E. LaMar, B. Hamann, and K.I. Joy. A magnification lens for interactive volume visualization. In *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, pages 223–232, 2001.

[71] Robert S. Laramee. First: a flexible and interactive resampling tool for CFD simulation data. *Computers & Graphics*, 27(6):905–916, 2003.

[72] M. Lazar, D.M. Weinstein, J.S. Tsuruda, K.M. Hasan, K. Arfanakis, E. Meyer, B. Badie, H.A. Rowley, V. Haughton, A. Field, and A.L. Alexander. White matter tractography using diffusion tensor deflection. *Human Brain Mapping*, 18:306–321, 2003.

[73] Teng-Yok Lee, O. Mishchenko, Han-Wei Shen, and R. Crawfis. View point evaluation and streamline filtering for flow visualization. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2011*, pages 83–90, March 2011.

[74] Teng-Yok Lee and Han-Wei Shen. Visualizing time-varying features with tac-based distance fields. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2009*, pages 1–8, 2009.

[75] Paul Leopardi. A partition of the unit sphere into regions of equal area and small diameter. *Electronic Transactions on Numerical Analysis*, 25:309–327, 2006.

[76] L. Li and H.-W. Shen. Image-based streamline generation and rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):630–640, May 2007.

[77] Tiecheng Liu and John R. Kender. Optimization algorithms for the selection of key frame sequences of variable length. In *Proceedings of the 7th European Conference on Computer Vision (ECCV)-Part IV*, pages 403–417, 2002.

[78] S.K. Lodha, A. Pang, R.E. Sheehan, and C.M. Wittenbrink. Uflow: visualizing uncertainty in fluid flow. In *Proceedings of IEEE Visualization 1996*, pages 249–254, October 1996.

[79] Julian Looser, Mark Billinghurst, and Andy Cockburn. Through the looking glass: The use of lenses as an interface tool for augmented reality interfaces.

In *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '04, pages 204–211, New York, NY, USA, 2004. ACM.

[80] Aidong Lu and Han-Wei Shen. Interactive storyboard for overall time-varying data visualization. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2008*, pages 143–150, 2008.

[81] S. Marchesin, Cheng-Kai Chen, C. Ho, and Kwan-Liu Ma. View-dependent streamlines for 3D vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1578–1586, November 2010.

[82] Oliver Mattausch, Thomas Theußl, Helwig Hauser, and Eduard Gröller. Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In *Proceedings of the 19th Spring Conference on Computer Graphics*, pages 213–222, 2003.

[83] M.J. McGuffin, L. Tancau, and R. Balakrishnan. Using deformations for browsing volumetric data. In *Proceedings of IEEE Visualization 2003*, pages 401–408, October 2003.

[84] B H Menze, E Stretton, E Konukoglu, and N. Ayache. Image-based modeling of tumor growth in patients with glioma. In C S Garbe, R Rannacher, U Platt, and T Wagner, editors, *Optimal control in image processing.* Springer, Heidelberg/Germany, 2011.

[85] P Mukherjee, JI Berman, SW Chung, CP Hess, and RG Henry. Diffusion tensor mr imaging and fiber tractography: theoretic underpinnings. *American journal of neuroradiology*, 29(4):632–641, 2008.

[86] Meinard Müller. *Information Retrieval for Music and Motion.* Springer Verlag, 2007.

[87] Andrew Nealen, Matthias Mller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[88] Petra Neumann, Tobias Isenberg, and M. Sheelagh T. Carpendale. NPR lenses: interactive tools for non-photorealistic line drawings. In *Proceedings of Smart Graphics 2007*, volume 4569, pages 10–22, 2007.

[89] Tyson Neuroth, Franz Sauer, Weixing Wang, Stephane Ethier, and Kwan-Liu Ma. Scalable visualization of discrete velocity decompositions using spatially organized histograms. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV) 2015*, 2015.

[90] Sinisa Pajevic and Carlo Pierpaoli. Color schemes to represent the orientation of anisotropic tissues from diffusion tensor data: Application to white matter fiber tract mapping in the human brain. *Magnetic Resonance in Medicine*, 42(3):526–540, 1999.

[91] Sung W. Park, Brian Budge, Lars Linsen, Bernd Hamann, and Kenneth I. Joy. Dense geometric flow visualization. In *Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis) 2005*, pages 21–28, 2005.

[92] T.H.J.M. Peeters, V. Prckovska, M. van Almsick, A. Vilanova i Bartroli, and B.M. ter Haar Romeny. Fast and sleek glyph rendering for interactive hardi data exploration. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2009*, pages 153–160, April 2009.

[93] Zhenmin Peng, Zhao Geng, Michael Nicholas, Robert S. Laramee, Nick Croft, Rami Malki, Ian Masters, and Chuck Hansen. Visualization of flow past a marine turbine: the information-assisted search for sustainable energy. *Computing and Visualization in Science*, 16(3):89–103, 2014.

[94] Zhenmin Peng and Robert S. Laramee. Vector glyphs for surfaces: A fast and simple glyph placement algorithm for adaptive resolution meshes. In *Proceedings of the Vision, Modeling, and Visualization Conference 2008, VMV 2008, Konstanz, Germany, October 8-10, 2008*, pages 61–70, 2008.

[95] Voicu Popescu, Paul Rosen, and Nicoletta Adamo-Villani. The graph camera. *ACM Transactions on Graphics*, 28(5):1581–1588, December 2009.

[96] Yossi Rubner, Leonidas Guibas, and Carlo Tomasi. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In *Proceedings of the ARPA Image Understanding Workshop 1997*, pages 661–668, 1997.

[97] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 1992*, pages 83–91, 1992.

[98] T. Schultz and G. L. Kindlmann. Superquadric glyphs for symmetric second-order tensors. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1595–1604, November 2010.

[99] T. Schultz, L. Schlaffke, B. Schlkopf, and T. Schmidt-Wilcke. HiFiVE: A Hilbert space embedding of fiber variability estimates for uncertainty modeling and visualization. *Computer Graphics Forum*, 32(3pt1):121–130, 2013.

[100] Ken Shoemake. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the Conference on Graphics Interface*

(GI) 1992, pages 151–156, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[101] Henry Sonnet, Sheelagh Carpendale, and Thomas Strothotte. Integrating expanding annotations with a 3D explosion probe. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI) 2004*, AVI '04, pages 63–70, New York, NY, USA, 2004. ACM.

[102] Richard Swinbank and R. James Purser. Fibonacci grids: A novel approach to global modelling. *Quarterly Journal of the Royal Meteorological Society*, 132(619):1769–1793, 2006.

[103] S. Takahashi, K. Yoshida, K. Shimada, and T. Nishita. Occlusion-free animation of driving routes for car navigation systems. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1141–1148, September 2006.

[104] Jun Tao, Chaoli Wang, Ching-Kuang Shene, and Seung Hyun Kim. A deformation framework for focus+context flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):42–55, January 2014.

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

[106] Wayne Tiller and Eric G Hanson. Offsets of two-dimensional profiles. *IEEE Computer Graphics and Applications*, 4(9):36–46, 1984.

[107] Xin Tong, Chun-Ming Chen, Han-Wei Shen, and Pak Chung Wong. Interactive streamline exploration and manipulation using deformation. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2015*, April 2015.

[108] David S. Tuch. Q-ball imaging. *Magnetic Resonance in Medicine*, 52(6):1358–1372, 2004.

[109] Matthew van Der Zwan, Alexandru Telea, and Tobias Isenberg. Continuous navigation of nested abstraction levels. In *Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis) 2012*, pages 13–17, 2012.

[110] John Viega, Matthew J. Conway, George Williams, and Randy Pausch. 3D magic lenses. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST) 1996*, UIST '96, pages 51–58, New York, NY, USA, 1996. ACM.

[111] James Walker, Zhao Geng, Mark Jones, and Robert S Laramee. Visualization of Large, Time-Dependent, Abstract Data with Integrated Spherical and Parallel Coordinates. In Miriah Meyer and Tino Weinkaufs, editors, *Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis) 2012 - Short Papers*. The Eurographics Association, 2012.

[112] C. Wang, H. Yu, R. W. Grout, K. L. Ma, and J. H. Chen. Analyzing information transfer in time-varying multivariate data. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis) 2011*, pages 99–106, March 2011.

[113] Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. Importance-driven time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1547–1554, 2008.

[114] Huamin Wang. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics*, 34(6):2461–2469, October 2015.

[115] Lujin Wang, Ye Zhao, K. Mueller, and A. Kaufman. The magic volume lens: an interactive focus+context technique for volume rendering. In *Proceedings of IEEE Visualization 2005*, pages 367–374, October 2005.

[116] Michael S. Warren, Alexander Friedland, Daniel E. Holz, Samuel W. Skillman, Paul M. Sutter, Matthew J. Turk, and Risa H. Wechsler. Dark sky simulations collaboration, July 2014.

[117] Frank Weichert, Daniel Bachmann, Bartholomus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380, 2013.

[118] D.M. Weinstein, G. Kindlmann, and E. Lundberg. Tensorlines: Advection-diffusion based propagation through diffusion tensor fields. In *Proceedings of IEEE Visualization 1999*, pages 249–253, 1999.

[119] Rolf Westerteiger, Andreas Gerndt, and Bernd Hamann. Spherical Terrain Rendering using the hierarchical HEALPix grid. In *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*, volume 27, pages 13–23, Dagstuhl, Germany, 2012.

[120] D. Whalen and M. L. Norman. "competition data set and description," in 2008 IEEE Visualization Design Contest, 2008.

[121] C.M. Wittenbrink, A.T. Pang, and S.K. Lodha. Glyphs for visualizing uncertainty in vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):266–279, September 1996.

[122] Koon-Pong Wong, Dagan Feng, S.R. Meikle, and M.J. Fulham. Segmentation of dynamic PET images using cluster analysis. *IEEE Transactions on Nuclear Science*, 49(1):200–207, 2002.

[123] N. Wong, S. Carpendale, and S. Greenberg. Edgelens: an interactive method for managing edge congestion in graphs. In *Proceedings of IEEE Information Visualization (InfoVis) 2003*, pages 51–58, October 2003.

[124] Jonathan Woodring and Han-Wei Shen. Multiscale time activity data exploration via temporal clustering visualization spreadsheet. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):123–137, January 2009.

[125] Jonathan Woodring and Han-Wei Shen. Semi-automatic time-series transfer functions via temporal clustering and sequencing. *Computer Graphics Forum*, 28(3):791–798, 2009.

[126] M. L. Wu and V. Popescu. Multiperspective focus+context visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(5):1555–1567, May 2016.

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

[128] L. Yu, K. Efstathiou, P. Isenberg, and T. Isenberg. Cast: Effective and efficient user interaction for context-aware selection in 3D particle clouds. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):886–895, January 2016.

[129] L. Yu, P. Svetachov, P. Isenberg, M. H. Everts, and T. Isenberg. FI3D: Direct-touch interaction for the exploration of 3D scientific visualization spaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1613–1622, November 2010.

[130] Xin Zhao, Wei Zeng, X.D. Gu, A.E. Kaufman, Wei Xu, and K. Mueller. Conformal magnifier: a focus+context technique with local shape preservation. *IEEE Transactions on Visualization and Computer Graphics*, 18(11):1928–1941, November 2012.

[131] Torre Zuk, Jon Downton, David Gray, Sheelagh Carpendale, and JD Liang. Exploration of uncertainty in bidirectional vector fields. In *Proceedings of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series 2008*, volume 6809, 2008. published online.

[132] D. Zwillinger. *CRC Standard Mathematical Tables and Formulae, 31st Edition*. Advances in Applied Mathematics. CRC Press, 1995.