Distribution-based Summarization for Large Scale Simulation Data Visualization and Analysis

Dissertation

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

By

Ko-Chih Wang, B.S., M.S.

Graduate Program in Department of Computer Science and Engineering

The Ohio State University

2019

Dissertation Committee:

Prof. Han-Wei Shen, Advisor

Prof. Yusu Wang

Prof. Jian Chen

Abstract

The advent of high performance supercomputers enables scientists to perform extreme-scale simulations that generate millions of cells and thousands of time steps. Through exploring and analyzing the simulation outputs, scientists can gain a deeper understanding of the modeled phenomena. When the size of simulation output is small, the common practice is to simply move the data to the machines that perform post analysis. However, as the size of data grows, the limited bandwidth and capacity of networking and storage devices that connect the supercomputers to the analysis machine become a major bottleneck. Therefore, visualizing and analyzing large-scale simulation datasets are posing significant challenges.

This dissertation addresses the big data challenge and suggests distribution-based in-situ techniques. The technique uses the same supercomputer resources to analyze the raw data and generate compact data proxies which use distribution to statistically summarize the raw data. Only the compact data proxies are moved to the post-analysis machine to overcome the bottleneck. Because the distribution-based data representation keeps the statistical data properties, it has the potential to facilitate flexible post-hoc data analysis and enable uncertainty quantification.

We firstly focus on the problem of large data volume rendering on resource-limited post analysis machines. To tackle the limited I/O bandwidth and storage space challenge, distributions are used to summarize the data. When visualizing the data, importance sampling is proposed to draw a small number of samples and minimize the demand of computational power. The error of the proxies is quantified and visually presented to scientists by uncertainty animation. We also tackle the problem of error reduction when approximating the spatial information in distribution-based representations. The error could cause low visualization quality and hinder the data exploration. The basic distribution-based approach is augmented by our proposed spatial distribution which is represented by a three-dimensional Gaussian Mixture Model (GMM). The new representation not only improves the visualization quality but can also be used in various visualization techniques, such as volume rendering, uncertain isosurface, and salient feature exploration. Then, a technique is developed to tackle the problem of large-scale time-varying datasets. This representation stores the time-varying datasets with a lower temporal resolution and utilizes the temporal coherence to reconstruct the data at non-sampled time steps. Each pixel ray at a view at non-sampled time step is decoupled into a value distribution and samples' location information. Our representation utilizes the data coherence to recover the samples' location information and store less data. In addition, similar value distributions from multiple rays are represented by one distribution to save more storage. Finally, a statistical-based super resolution technique is proposed to solve the big data problem caused by a huge parameter space. Simulation runs with a few parameter samples output full resolution data which is used to create the prior knowledge. Data from rest of simulation runs in the parameter space is statistically down-sampled to compact representation in situ to reduce the data size. These compact data representation can be reconstructed to high resolution by combining with the prior knowledge for data analysis.

This is for you, Dad and Mom. Thanks for always being there for me.

And for our Lee Yue Honey Castella.

Acknowledgments

My doctoral journey in The Ohio State University is worthwhile, fulfilling, and also challenging. The experience and achievements help me in getting one more step closer to my dream career. This journey is not easy for me. Without the help from many kind people, I may never finish my doctoral study. I would like to express my most sincere appreciation at the beginning of my dissertation.

First, I am deeply indebted to my advisor, Prof. Han-Wei Shen, for his fundamental role in my doctoral work. Prof. Shen guided and assisted me in every piece of my doctoral study. At the beginning, Prof. Shen gave me the chance to join his research group. He advised and supported me with patience even if I struggled in having significant research process in the first several years. Prof. Shen also pulled me back multiple time when I almost gave up. I also received a lot of help from Prof. Shen in my faculty job search. Without Prof. Shen's patience, support and encouragement, the completion of my Ph.D. degree would not have been possible.

I am also extremely grateful to Naeem Shareef. He spent a lot of his personal time to discuss my research and help me to improve English in my academic papers. Because he is patient in explaining the mistakes in my writing, his help does not only fix English errors in papers but also improve my English skills. This gives me the confidence to step into the academic career. Besides, Naeem Shareef is also one of my best friends in The Ohio State University. I learned a lot of Americans' perspectives and cultures from him.

I must express my deepest appreciation to my mentors in Los Alamos National Laboratory, Jonathan Woodring and Li-Ta Lo. I learned a lot about parallel computing, software development, visualization tools and research skills from them. The experience in LANL give a big boost of my doctoral study.

I would like to thank my labmates, who have discussed research with me, challenged my ideas and pointed out the blind spots of my thinking. They are: Tzu-Hsuan Wei, Chun-Ming Chen, Soumya Dutta, Kewei Lu, Junpeng Wang, Subhashis Hazarika, Cheng Li, Jiayi Xu, Wenbin He, Xin Tong, Ayan Biswas, Xiaotong Liu, Paul Kim, Robert Gross, Ming-Yi Su, Rachel Wang, Harvey Li, Gaun Li and Rui Li.

I would also like to thank my advisor in National Taiwan University, Chieh-Chih (Bob) Wang. His advice and encouragement are the reason I can start my Ph.D. journey. I also would like to thank anyone in my life who supports me to pursue my Ph.D. degree.

Lastly, I am deeply indebted to my family. Thank you, my dad and mom. No matter what decision I made, you always support me and provide whatever you can help me. I am really lucky that I am your son. Thank you, my sister. I am really happy that I can grow up with you together. I am extremely grateful to my wife, Chih-Hsuan Lu, for accompanying me. You always encourage me and listen to me when I feel down. You are always happy for me when I complete milestones. You are the most indispensable spiritual support to me. Your unconditional love and sacrifice are why I can overcome so many difficulties to finish my doctoral study.

Vita

September 8, 1982	Born Tainan City, Taiwan
2004	B.S. Computer Science Tunghai University, Taiwan
2007	M.S. Computer Science National Taiwan University, Taiwan
2007-2008	Alternative Military Service Fifth River Management Office, Taiwan
2008-2010	Research Assistant National Taiwan University, Taiwan
May-August 2017	Research Intern Los Alamos National Lab.
2015-2017	Graduate Teaching Associate The Ohio State University
May-August 2018	Research Intern Los Alamos National Lab.
2013-2019	Graduate Research Associate The Ohio State University

Publications

Research Publications

Ko-Chih Wang, Tzu-Hsuan Wei, Naeem Shareef, Han-Wei Shen "Ray-based Exploration of Large Time-varying Volume Data Using Per-ray Proxy Distributions", IEEE Transactions on Visualization and Computer Graphics, March, 2019 [Minor Revision].

Ko-Chih Wang, Jiayi Xu, Jonathan Woodring, Han-Wei Shen “Statistical Super Resolution for Data Analysis and Visualization of Large Scale Cosmological Simulations”, In IEEE Pacific Visualization (PacificVis) Symposium, 2019.

Ko-Chih Wang, Naeem Shareef, Han-Wei Shen “Image and Distribution Based Volume Rendering for Large Data Sets”, In IEEE Pacific Visualization (PacificVis) Symposium, pages 161–170, 2018.

Ko-Chih Wang, Kewei Lu, Tzu-Hsuan Wei, Naeem Shareef, Han-Wei Shen “Statistical visualization and analysis of large data using a value-based spatial distribution”, In IEEE Pacific Visualization (PacificVis) Symposium, pages 161–170, 2017.

Fields of Study

Major Field: Computer Science and Engineering

Studies in:

Computer Graphics and Visualization	Prof. Han-Wei Shen
Computational Science	Prof. P. Sadayappan
Database	Prof. Arnab Nandi

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Background and Motivation

The computational power of supercomputers grows rapidly in the past decade. According to the list of TOP500 supercomputers, which is released in June 2008, the most powerful supercomputer is RoadRunner in the Department of Energy's Los Alamos National Laboratory where its peak performance just breaks the PFLOPS barrier to reach 1.042 PFLOPS (Peta floating-point operations per second). But according to the updated list of TOP500 supercomputers , which is released on June 2018, the most powerful supercomputer is Summit, which runs in the Department of Energy's Oak Ridge National Laboratory. Its peak performance can reach 187.6 PFLOPS. The computational power of the fastest supercomputer in the world has increased around 180 times in the past ten years. The increasing performance of supercomputers allows scientists to model complex physical phenomena with high-resolution simulations which has millions of spatial grid points and hundreds or even thousands of time steps. Scientists can explore the high-resolution data and have greater understanding about the physical phenomena. To save the supercomputing resources, the datasets are usually moved out from the supercomputer and stored on other devices for data analysis later when the simulation is done. If the size of simulation output is small, common practice is to simply move the data to machines that perform post analysis.

However, as the size of data grows, the limited bandwidth and capacity of networking and storage devices that connect the supercomputers to the analysis machine become a major bottleneck [2, 60, 105]. Handling and analyzing the large-scale datasets are not easy tasks. So, an appropriate approach is crucial.

To overcome the bottleneck, the datasets generated by the simulation can be replaced by a compact data proxy then written to the storage devices. An ideal data proxy should be compact with minimal error and allow scientists to explore the dataset with minimal constraints. However, to create such a general data proxy that can be applied to all types of datasets and analysis tasks is difficult. In practice, scientists often have different needs depending on the analysis tasks and types of datasets. To reach a good trade-off between the size and error of the data proxy, the data proxy should be designed by considering (1) the analysis and visualization techniques which will be used to explore the dataset, (2) the error tolerance and the allowed proxy size budget, and (3) the type of dataset which is analyzed. The application scenarios this proposal focuses on and their challenges are listed below:

- Direct volume rendering is a widely used scientific data visualization technique [14, 47, 73]. With transfer function exploration, scientists can discover salient features and attain a deep understanding of the datasets. However, while using the compact proxies to overcome the aforementioned big data challenge, preserving the ability of exploring occluded features and the quality of data visualization are not trivial tasks. Furthermore, quantizing and visualizing the error in the compact proxies to scientists are also important [7, 17, 59]. This prevents making incorrect data analysis conclusions from the error in the data proxy. Therefore, a compact proxy design that enables both transfer function exploration and error quantification is needed.

- To deeply understand physical phenomenon, modern supercomputers allow scientists to run simulations with very high spatial and temporal resolutions [5,35,69]. To handle such big datasets, a common practice is to reduce the spatial and temporal resolution of datasets. However, this naive approach will introduce large error and could mislead the analysis tasks [56, 80]. For example, when visualizing the data, interpolation approximation is usually used to reconstruct the data. This could reconstruct non-existing values and incorrect feature shapes. Therefore, it is necessary to develop a technique to reconstruct datasets to original spatial and temporal resolutions with less error. This technique should not only have good trade-off between the size and the error but also needs to control the size of data proxy flexibly by the given size budget.

- Simulations usually allow different initial conditions to produce different simulation results. The datasets produced by same simulations are called *ensemble dataset* and each result produced by an initial condition is called a *ensemble member*. Scientists are often interested in studying the relation among different initial conditions [91, 92]. Many data visualization and analysis techniques have been developed for ensemble study to understand the uncertainty of the output datasets and the sensitivity of the input parameters [30, 67]. However, the total data size of an ensemble dataset is often very large. For example, if an initial condition consists of P parameters and each parameter has N possible values, the size of the ensemble dataset will be $P * N$ times than that of one ensemble member. Therefore, an ensemble dataset is often a terabyte- to petabyte-scale data. To design a compact proxy for ensemble study is important.

1.2 Proposed Solutions

We propose a novel technique which we call distribution-based in-situ data summarization to tackle the above challenges of the applications. Our proposed technique has two key components which are in-situ technique and distribution representation.

The in-situ technique processes the data and generates the compact data proxies when the data resides on the supercomputer. Only the data proxies are moved to the post analysis machine for data exploration later. Compared with the traditional data analysis pipeline, which stores raw datasets to the persistent disk and then explores the raw datasets, the in-situ technique has some benefits. The expensive data movement is significantly reduced by only moving the compact data proxies to the post analysis machine and the post analysis computer requires lower storage and less memory footprint.

The distribution-based approach compactly represents a population of data points by summarizing the data using a distribution. The distribution does not only store the data points compactly, but also preserves their statistical characteristics. Compared with traditional data reduction approaches, the distribution-based approach has some benefits. The more flexible post-hoc data analysis approaches can be built on top of the distribution-based representation because the statistical characteristics of data points are preserved. In addition, error is inevitable in the compact data representation, which could mislead the data analysis. The distribution-based representation has the potential to enable uncertainty quantification and show confidence of the visualization to scientists.

In this dissertation, we propose in-situ distribution-based data summarization approaches to tackle different types of large-scale data analysis problems. (1) When experts need to study large volume datasets using volume rendering, we propose a view-dependent distribution-based approach which compactly summarizes the data projected to an image pixel. To accurately preserve the depth cue in the visualization, the data proxy irregularly subdivides each pixel frustum to minimize the information loss in the distribution of each sub-frustum based on the user-given size budget. This representation can visualize the data efficiently on the low-cost post analysis machine and quantify the uncertainty by adapting the user given transfer function. (2) For the datasets with large spatial resolution, we propose a compact data proxy which uses distribution to model data values and their spatial locations separately. The distribution of data values is summarized by a histogram. Because the distribution of data values does not preserve the data samples' location, we use a spatial Gaussian Mixture Model (GMM) to compactly model the spatial location distribution of data samples which fall in the value interval of a bin. The distributions of data values and their spatial location information can be combined by Bayes' rule to reconstruct the data value at arbitrary spatial locations. The error quantification of each reconstructed data value is provided. (3) To tackle the big data problem which is caused by the high temporal resolution, we propose a view-dependent data proxy which can capture the evolution of features at non-sampled time steps when the naive temporal down-sampling is used to handle the dataset. We decouple each ray at non-sampled time steps into a value distribution which summarizes samples along the ray and the samples' location information. We either directly store the samples' location information or utilize data coherence in the temporal domain to represent the samples' location information. In addition, our technique can identify the similar value distributions and location informations to save more storage. (4) For ensemble dataset, we propose a data proxy which utilizes the data coherence in the simulation parameter space to create a compact data proxy. Most of ensemble members are statistically down-sampled. The spatial domain is subdivided into multiple blocks and data in each block is represented by a Gaussian Mixture Model. However, GMM does not preserve the samples' location information in the data block. Therefore, we store a small number of full resolution ensemble members which are called prior knowledge. When we reconstruct a data block to high resolution, a data block's statistical values and surrounding data blocks are used to look up the most similar data block in the prior knowledge. Because the data coherence in the parameter space, the retrieved data block in the prior knowledge can be used to predict the samples' location in the data block.

In summary, this dissertation is organized as follows: In Chapter 2, we discuss the relevant previous research works. Chapter 3 provides the details of the proposed image- and distribution-based large volume rendering technique. Chapter 4 presents the distribution-based representation augmented by value-based spatial distribution in detail. Chapter 5 introduces the technique which compactly stores the non-sampled time steps in the temporal domain and provides high quality evolution of features. The details of the data proxy for ensemble data study are explained in Chapter 6. Finally, we conclude this dissertation in Chapter 7 and discuss several future research directions.

Chapter 2: Background and Related Work

In this chapter, we discuss the previous research works related to this dissertation. These related research works are organized into several categories such as large data representations, in-situ approaches, distribution-based data analysis, image-based approaches and uncertainty visualization. In the beginning, we discuss the techniques which are developed to represent and store large-scale datasets. Section 2.2 covers the research works related to in-situ workflow. In Section 2.3, we summarize the statistical based approaches used on data analysis and visualization. The related works of the image-based approaches are discussed in Section 2.4. Finally, the works which focus on error quantification and visualization are discussed in Section 2.5.

2.1 Data Representation for Large Dataset

Data representation plays an important role in large data analysis. It is designed to reduce the memory footprint, save the I/O time, or efficiently access the data on demand in the task of data analysis. A traditional and straightforward approach to reduce the size of volume data is to down-sample the volume to a smaller resolution. However, this approach suffers from aliasing and inconsistent artifacts, as pointed out in [56,80,104]. The compression technique is one of the major categories to reduce the I/O overhead [11,21,39,40,48,49,55,58,83]. The compression techniques could also produce the inconsistent artifacts in the visualization without the proper error quantification. Multi-resolution strategy loads different level-of-detail data proxies by the need to reduce the requirement of I/O bandwidth and memory footprint. Shen et al. [79], Sicat et al. [80] and Xie et al. [98] used this strategy to handle the large datasets on the commodity computers. Statistically summarizing each low-resolution block preserves more information for the reconstruction to the original data resolution when visualizing the datasets. Dutta et al. [26] irregularly subdivided a volume by minimizing the data distribution entropy in each sub-region to both reduce the proxy size and preserve the representation quality. Wei et al. [96] used stratified sampling to preserve both data distribution of each low-resolution block and samples' location distribution in the spatial domain for the accurate data reconstruction. The simulations which generate time-varying datasets can easily have hundreds or thousands time steps, and simply create huge amount of data. Therefore, many techniques have been proposed to handle the time-varying datasets. Hazarika et al. [41] proposed CoDDA which is a flexible distribution-based framework to handle large-scale multi-variate data. Chen et al. [17] used quadratic Bezier curve to calculate the vector between two sampled time steps and compute the reliable pathlines. Gau et al. [34] used temporal coherence to increase rendering performance. Some approaches focus on specific analysis applications to produce the compact proxies. Sauer et al. [75] proposed a new data structure which combines Eulerian and Lagrangian reference frames to provide efficient data sampling and querying. Yu et al. [106] proposed scalable and hierarchical tree structure to compute the distance field from large datasets in parallel. The state-of-the-art data reduction techniques for scientific data visualization and analysis are summarized by Meyer et al. [63] and Li et al. [53]. They discussed and categorized the approaches in term of different use cases.

2.2 In-situ Techniques

While the simulations run on the modern supercomputers to produce Pera-byte scale datasets, the common practice which moves whole datasets to persistent disk and wait for analysis are infeasible. In-situ workflow is proposed to tackle the problem from the constraint of the I/O bandwidth and the limited disk capacity by processing the raw datasets to produce compact data proxies using the same supercomputer resources. Many works which target on different applications integrate in-situ workflow to solve their own large data problems. Ma et al. [60] and Yu et al. [105] proposed systems which use in-situ workflow to visualizing large time-varying datasets. Ahrens et al. [2, 3] and O'Leary et al. [68] proposed image-based approaches which directly render a set of images in the supercomputer to reduce the data movement. Scientists can interactively explore the dataset via these images. Dutta et al. [23] modeled the dataset from a jet engine simulation by a set of Gaussian Mixture Model to reduce the data size and visualize the jet engine behavior by these GMMs only. Dutta et al. [25] proposed a prediction driven approach on the same application. It requires small number of training set from the domain expert to train the model for the jet engine stall prediction. The model is used to predict the region of the jet engine stall and their technique only outputs the data around the stall region to persistent disk for analysis. Bennett et al. [9] proposed an hybrid approach which combines in-situ and in-transit workflows to enable the analysis on multiple scientific applications. Friesen et al. [33] compared in-situ and in-transit workflows by using two popular analysis scenarios on the cosmological simulation. They pointed out the pros and cons of these two workflows under different simulation configurations.

2.3 Distribution-based Data Analysis

Distribution-based approaches have been proposed to summarize datasets for analysis or keep the statistical characteristics of datasets while reducing the memory footprint. Chen et al. [17] improved pathline computation by interpolating from temporally down-sampled data and model the error with a Gaussian distribution. He et al. [42] used particle filtering to visualize streamlines from datasets with uncertainty. They utilized distribution-based querying to find the regions with certain types of distributions. Chaudhuri et al. [15, 16] and Lee et al. [51] used a histogram based Summed Area Table to access the histogram of any sub-volume of the data set, which can be used to support a variety of data analysis tasks. Their work focused on the efficient access of local distributions rather than a compact representation. Wei et al. presented an efficient algorithm to search similar distributions of local regions [95]. Thompson et al. [84] used a histogram representation to approximate topological structures and fuzzy isosurfaces. Liu et al. [56] used Mixture of Gaussian to summarize datasets and visualize the hidden probability model in the simulation. Athawale et al. [6, 7] proposed statistical isosurface visualization techniques that compute the probability of an iso-value crossing at each voxel and visualize the probability field. Johnson et al. [45] found that the distribution of a local region contains meaningful information which is relevant to many feature queries. Therefore, they proposed a visualization system which allows users to interactively enter customize statistical hypothesis and explore volume datasets. Dutta et al. [24] proposed a framework which can accurately track a user-defined feature. It modeled the feature by a distribution and kept updating the distribution over time.

2.4 Image-based Techniques

Image-based approaches are particularly suited for visualization of large scale data where a proxy is constructed from the original dataset. These approaches mainly focus on both data reduction and fast volume rendering by re-organizing the data along pixel rays. One of the main advantages of the image-based approach is to bound the memory footprint and enable the interactive data exploration [18, 31, 32, 66, 72, 78]. Tikhnova et al. [85–87] proposed image-based frameworks which allow users to change the transfer function and explore volume datasets by storing small number of image slices used to approximate pixel color. Fernandes [28] proposed a volume depth image to model the data of each pixel ray in the spatiotemporal domain for time-varying datasets. Yu et al. [100] proposed an in-situ simulation technique which stores the image-based depth map for isosurfaces. The posterior feature extraction and tracking are enabled from their depth map proxies. Ahrenes et al. and O'Leary et al. [2, 3, 68] created a "Cinema" database from large scale datasets where snapshots are taken from different views along with stored material information. Once the database is constructed, scientists view and analyze the datasets from the stored images. Meyer et al. [64] defined a light field to store images of different materials from the dataset for volume rendering. Another application category of the image-based data structure is to select the proper visualization configuration. Correa and Ma [19] used a ray distribution of visibility to assist the transfer function design in direct volume rendering. For streamline visualization, Lee et al. [50] proposed maximum entropy projection to select the views which have low occlusion and high importance.

2.5 Uncertainty Visualization

The error and uncertainty in datasets can come from multiple different sources. The first category of uncertainty is from the data measurement because of the limit precision of sensors. The second category of uncertainty is from the stochastic model used in the simulation. The third category of the uncertainty is from the error which is generated from the data size reduction. Quantifying and visualizing the uncertainty to scientists are valuable because scientists can deeply understand the physical phenomenon. Therefore, the uncertainty quantification and visualization techniques which focus on different categories of uncertainty have been developed to inform the error to scientists. Box plot [61] is an well-known uncertainty visualization approach which shows the uncertainty by its mean and standard deviation on 2-D plot. Spaghetti plot [4] is a visualization technique which is used to visualize the uncertainty in the ensemble forecast system. Liu et al. [56] generated animation by keeping re-sampling samples from the uncertain data over time to convey the degree of uncertainty at different spots in the dataset. Sakhaee and Entezari [74] proposed an approach to render the volume data with uncertainty. Their approach assumes that each grid point is represented by a distribution and it computes a distribution at a sampling location by a distribution-based interpolation. The transfer function is applied to the interpolated distribution to compute the color at a sampling location for volume rendering. Athawale and Entezari [7] proposed an uncertain level-crossing algorithm to compute the field of the occurrence probability of an isovalue to visualize an uncertainty dataset. Schlegel et al. [76] pointed out the problem of using the traditional linear interpolation to recover the missing data. The traditional linear interpolation often makes an incorrect assumption to visualize the data and mislead the analysis. They proposed a method which uses Gaussian process regression to interpolate uncertain data and gain a continuous uncertainty field for visualization. Thompson et al. [84] proposed an algorithm to compute the uncertain isosurface which describes the probability of a isosurface which passes a grid point.

Chapter 3: Image and Distribution Based Volume Rendering for Large Data Sets

The advent of high performance supercomputers enables scientists to perform extreme scale simulations that generate millions of cells and thousands of time steps. Through exploring and analyzing the simulation outputs, scientists can gain a deeper understanding of the modeled phenomena. When the size of simulation output is small, the common practice is to simply move the data to the machines that perform post analysis. However, as the size of data grows, the limited bandwidth and capacity of networking and storage devices that connect the supercomputers to the analysis machine become a major bottleneck [2, 60, 105]. To ensure high quality analysis of the large scale datasets, the concept of in-situ data analysis has been proposed. In-situ techniques use the same supercomputer resources to analyze and generate compact data proxies without moving the raw datasets [23, 27, 60, 97]. Among the different techniques, the imaged-based approach has emerged as a promising method for in-situ visualization and analysis [2, 85, 87]. In the image-based approaches, scientists pre-select several interesting views based on their prior-knowledge and goals, and produce images for post data analysis. Comparing to the scale of large scale simulations ($> 10^{9\sim15}$), objects defined in image space have a relatively smaller size ($\sim 10^6$) [2, 87], thus offering the potential to overcome the big data challenge.

Although these image-based approaches have demonstrated successful results on different analysis scenarios, they have potential shortcomings. In the context of volume rendering, the existing image-based approaches have very limited or no ability for transfer function exploration. This results in difficulties in discovering and analyzing unfamiliar features that are occluded or far away from the camera. Although there exist approaches that can produce approximated images with modified transfer functions, the error from the approximation is not quantified and communicated to the scientists. In addition, when image-based approaches generate compact proxies from large scale datasets, information loss become inevitable, which complicates the task of data analysis.

To address these problems, this paper presents a novel image-based approach for large scale data analysis and visualization using statistical distributions as the proxy. Our image-based proxy allows transfer function exploration at the post analysis stage, as well as quantification and visualization of the error introduced by the proxy. To enable error quantification, we use distributions to summarize data samples along each ray. Distribution-based approaches have been used to reduce the data size by summarizing data in the object space and also provide the required information for uncertainty quantification [23,56,84,93]. Our technique casts multiple rays from a single image pixel and collect samples at a sufficient sampling rate. The data on the pixel ray can be re-sampled from the distributions and then the image is rendered by applying the user-supplied transfer function to the reconstructed samples. To render an image, the correctness of spatial order among the samples affects the visualization quality. Since the distribution does not keep the location information of samples, distributions that have a larger degree of randomness tend to introduce larger rendering error. Our technique uses Shannon entropy [90] to evaluate the randomness of sample values belonging to each pixel. Distributions collected from more ray segments are used to represent data of a pixel with higher randomness to preserve the visualization quality. Our framework also adapts the proxy size to the target storage bandwidth and post analysis machine's storage by adjusting the randomness tolerance threshold. In the post analysis stage, we propose an approach to integrate both scientist-given transfer function and ray distribution to re-sample the data on the pixel ray, instead of directly re-sampling from ray distributions. This scheme avoids spending time on sampling transparent samples to achieve interactive exploration on the post analysis machine.

The remainder of this chapter is organized as follows. Section 3.1 is an overview of our approach. Section 3.2 describes our proxy generation algorithm. The rendering algorithm on the post analysis is discussed in Section 3.3. Section 3.4 presents our results. Section 3.5 provides a discussion of our approach. Finally, Section 3.6 concludes the approach.

3.1 System Overview

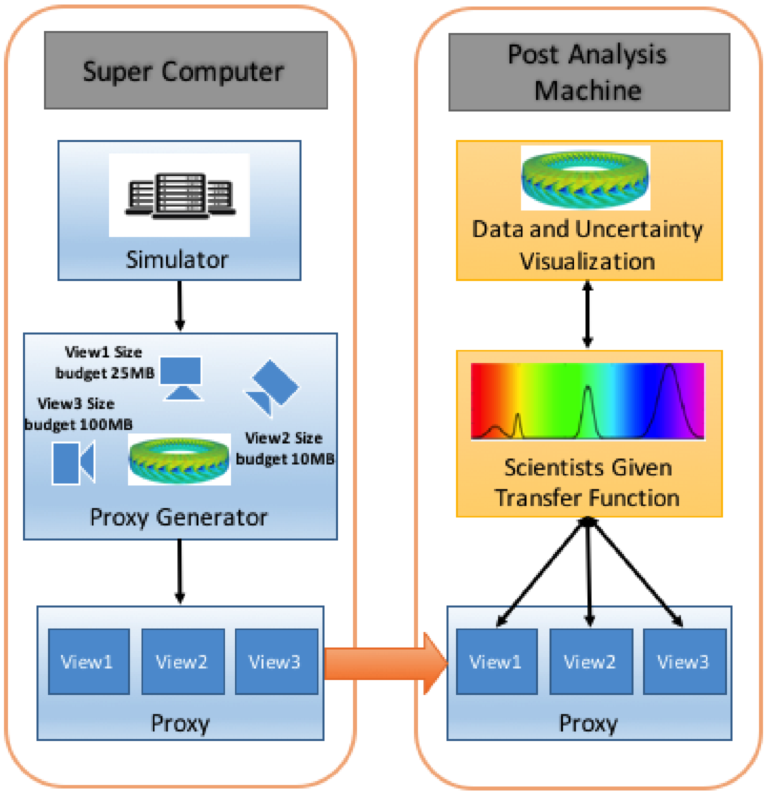


Figure 3.1: An overview of our proposed technique.

Figure 3.1 shows an overview of our proposed framework. Our technique takes the dataset from the simulator to generate distribution- and image-based proxies for each scientists' selected view. The scientist can set the storage size budget for each view proxy. Our framework generates the view proxy by the given size limit, and then moves only the proxies to the post analysis machine. In the post data analysis step, the dataset is visualized directly from the proxies. The scientists can provide different transfer functions to explore the dataset. The uncertainty of the proxies is also quantified and visualized to the scientists.

We implemented both the proxy generation and post-hoc visualization using the data-parallel programming model. To write the data-parallel program, the processed data (either the raw data in the proxy generation or the proxy in the post-hoc visualization) is separated into data units by image pixels. Each independent processor performs the same algorithm on each data unit. The algorithms written by data-parallel programming model are portable on different back-end devices [57]. Our algorithms are implemented on the VTK-m library [65]. VTK-m can run algorithms written by the data-parallel programming model on either GPU (CUDA) or multicore CPU (Intel TBB) by simply changing the compiler configuration without rewriting the code.

3.2 Image-based Distribution Proxy

In this section, we present a detailed description of our image based distribution proxy. For each view sample, an image plane is defined with a target pixel resolution, R. At each pixel, a distribution summarizes the information within the pixel frustum. Since a distribution lacks depth information, it may be beneficial to subdivide the pixel frustum into sub-frustums that are non-overlapping and depth ordered, as shown in Figure 3.2. Each sub-frustum (three are shown in the figure) will be associated with a different distribution that summarizes information within that smaller region. This will provide a better approximation of location information of the samples summarized in a distribution. We present our subdivision algorithm in Section 3.2.1.

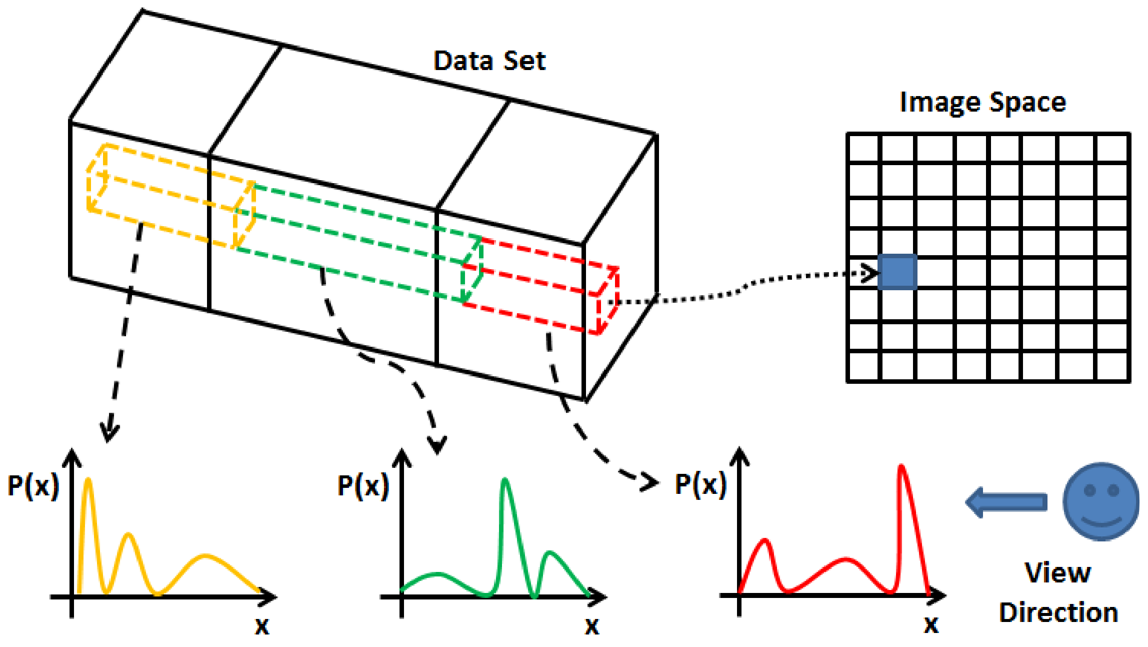


Figure 3.2: An illustration of our proxy where the image plane is shown on the upper right. Each pixel, such as the pixel shown in blue, stores a depth ordered list of sub-frustums (shown on the left). A distribution is associated with each sub-frustum containing a summary of the information within the sub-frustum.

3.2.1 Proxy Construction

A pixel frustum at pixel p is subdivided into K^p sub-frustums and a distribution is computed for each sub-frustum. The total number of sub-frustums in the proxy will be limited by the storage size budget provided by the scientists. Though, the number of subdivisions between different pixel frustums may not be the same. A pixel frustum is subdivided according to the "randomness" of the scalar values in the frustum. A sub-region in a pixel frustum with scalar values in a narrow value range (low randomness) will be appropriate to define a distribution that will incur low error during reconstruction. Whereas a sub-region with high randomness will incur a larger error and would need to be subdivided into smaller sub-regions to alleviate this error. We use a threshold on randomness that represents a tolerance to decide whether or not to further sub-divide a sub-region. Randomness is estimated with Shannon entropy [90] which is shown in Equation 3.1.

$$E(h) = -\sum_{b=1}^{B} h^N(b) * log_2(h^N(b)) \tag{3.1}$$

where h is a histogram, B is the bin count of h and h^N is the normalized histogram from h where $\sum_{b=1}^{B} h'(b) = 1$.

We decided to represent a distribution associated with a sub-frustum by a histogram because the computational cost to construct a histogram from samples is cheaper when compared with other representations, such as Gaussian or Gaussian Mixture Model which need the time consuming EM algorithm to estimate the parameters of the distribution representation. In addition, the histogram would be convenient to compute Shannon entropy. In order to sufficiently sample the information in the sub-frustums, we supersample at the pixel with multiple sampling rays. A histogram is constructed by accumulating samples across the sampling rays within the boundaries of the associated sub-frustum. This will typically include hundreds to thousands of samples that are summarized within a histogram. All of the per-pixel sampling rays will define the same number of samples along each ray. Pixel sub-frustums can be easily delimited with indices assigned to the samples along the rays in depth order. The sample closest to the image plane will be assigned index 0 while the samples further away will be assigned larger indices according to depth. The samples across the per-pixel sampling rays within the same sub-frustum are easily identified by the minimum and maximum indexes of the samples that belong to that sub-frustum.

Algorithm 1 shows the steps to subdivide the pixel frustum into sub-frustums. Samples are collected from all per-pixel rays in lock step according to front-to-back order. At each step, the samples with the same indices across the per-pixel rays are used to update the histogram associated with the current sub-frustum. Once the entropy of the histogram exceeds the entropy threshold, a new sub-frustum is started along with a new histogram.

At Line 10 in Algorithm 1, we re-calculate the entropy of the histogram *hist* with the next samples. Since re-calculating the entropy with Equation 3.1 is expensive, we can use Equation 3.2 to quickly update the entropy when only a single bin changes and given the entropy of the histogram before the update.

$$\begin{aligned} E_{n+1} = \frac{S_n}{S_n+c}(E_n - (-\frac{f_n^b}{S_n} log \frac{f_n^b}{S_n})) \\ -(S_n - f_n^b)(\frac{1}{S_n+c} log \frac{S_n}{S_n+c}) \\ +(-\frac{f_n^b+c}{S_n+c} log \frac{f_n^b+c}{S_n+c}) \end{aligned} \tag{3.2}$$

where E_{n+1} and E_n are the entropy of the updated and old histogram, S_n is sum of frequency of the unupdated histogram, f_n^b is the frequency of the changed bin (b) before updating and c is the change of frequency of the bin. Equation 3.2 is derived from the fundemental Shannon entropy equation [90]. We use it to update the entropy at Line 10 in

Algorithm 1 Proxy construction at a pixel

```
1: ▷ $p$: a pixel
2: ▷ $E$: randomness (entropy) threshold
3: procedure PixelProxyConstruction($p$, $E$)
4:     $i = 0$                                   ▷ sample counter on a per-pixel ray
5:     $e = 0$                                   ▷ entropy of a updated histogram
6:     while MoreSampleOnPixel($p$, $i$) == $True$ do
7:         for each sub-ray in $p$: $r$ do               ▷ r: a sub-ray in pixel p
8:             if MoreSampleOnRay($r$, $i$) == $True$ then
9:                 $v$ = GetSampleOnRay($r$, $i$)
10:                $e$, $hist$ = update($e$, $hist$, $v$)    ▷ entropy and histogram update
11:            end if
12:        end for
13:        if $e \geq E$ then
14:            SaveHistogram($hist$)
15:            $e = 0$
16:            $hist$ = reset($hist$)
17:        end if
18:        $i = i + 1$
19:    end while
20: end procedure
```

Algorithm 1. This entropy updating is a constant time computation. In addition, the table for logarithm computation is pre-computed for Equation 3.2. Thus, the process to subdivide the pixel frustum does not introduce too much overhead when generating the proxy.

3.2.2 Entropy Threshold Selection

In our technique, the scientists can give a proxy size budget for each view according to the storage bandwidth and capacity of the post analysis machine. Or the budget can be decided based on the error tolerance of each selected view. Before generating the proxy for a view, our system computes the entropy threshold based on the given proxy size budget, and then use the entropy threshold to perform the algorithm in Section 3.2.1.

To compute the desired entropy threshold for the given size budget, a straightforward approach is to simply use the algorithm in Section 3.2.1 to test different entropy thresholds and use the corresponding proxy size to pick the threshold that produces the proxy closest to the target size. In order to make the entropy threshold selection more efficient, we made two modifications to the naive approach. Firstly, our technique uses binary search to find the threshold which can generate the proxy whose size falling within the size budget. It first estimates the largest proxy size using the smallest threshold and the smallest size by the largest entropy threshold. Then the binary search is applied to identify the proper entropy threshold. Secondly, the binary search is only applied to the first time step of the dataset. From the second time step, we use the threshold from the previous time step as an initial guess. Then, shifting 1% of the maximal entropy from the initial guess to find the proper threshold. This saves a significant amount of time on the entropy threshold selection step after the second time step.

3.2.3 Proxy Data Structure

Figure 3.3 illustrates the data structure for our proxy at a view sample and for a single time step. The data structure compactly stores the proxy with four arrays. The two arrays shown at the bottom of the Figure 3.3 contain all of the histograms in the proxy. We select to store the histogram in a sparse representation where zero frequency bins are not stored explicitly. Although the sparse histogram representation in general does not always cost less storage than the non-sparse representation, the histogram in our approach only represents a pixel sub-frustum which is a small space region and usually only includes samples with small value interval. This makes most of the histogram bin's frequency zero, so we choose to use the sparse representation. The two blue arrays are the Bin IDs and frequencies of histograms in a sparse representation. Each histogram is associated with its respective pixel and sub-frustum using references from the top two arrays, as shown in the figure. The top array is a list of the pixels on the image plane, where each pixel refers to the first sub-frustum in its list of sub-frustums in depth order. The array below this one contains all per-pixel sub-frustum lists. Each sub-frustum refers to its associated histogram.

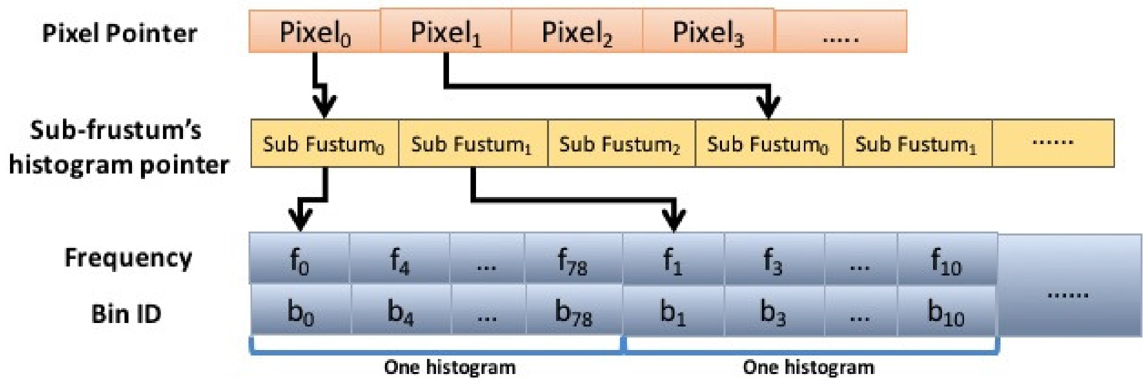


Figure 3.3: The proxy data structure for one view and one time step.

3.3 Proxy Visualization

A basic algorithm to volume render the proxy is to compute a color at each pixel by rendering the associated pixel sub-frustums in front-to-back order. To render a pixel sub-frustum, $s_{f_k^p}$ re-sample locations are used along the pixel ray within this sub-frustum, where $s_{f_k^p}$ is the number of samples reconstructed from the frustum's (f_k^p) distribution. $s_{f_k^p}$ is computed by Equation 3.3

$$s_{f_k^p} = s_p * \left(\frac{FS(f_k^p)}{\sum_{i=1}^{K^p} FS(f_i^p)}\right) \tag{3.3}$$

where s_p is the desired re-sample count of a pixel, $FS(f_k^p)$ is the frequency sum of the k^{th} sub-frustum's histogram on pixel p and the $\sum_{i=1}^{K^p} FS(f_i^p)$ is the frequency sum of all sub-frustums' histograms on pixel p .

The scalar value at a re-sample location can be determined from the histogram associated with a sub-frustum containing the location using Monte Carlo sampling, a well known reconstruction approach. The first drawn sample is applied to the front-most re-sample location and then this continues for re-sample locations further away. Each re-sampled value is applied to the transfer function and composited to determine a color and opacity for the sub-frustum.

It is well known that Monte Carlo sampling reproduces samples well when the number of samples drawn approaches infinity, which is infeasible in practice. In the following sections, we present improvements to the basic algorithm using an importance distribution, which incorporates both scalar and opacity information at each pixel sub-frustum. The importance distribution can better sample the data distribution at each sub-frustum for improved image quality. Also, the importance distribution allows for fewer samples to be used to achieve faster frame rates with only a small loss of image quality.

3.3.1 Importance Sampling

Samples drawn from a sub-frustum's distribution using Monte Carlo sampling are biased towards scalar values with high frequency. If these samples contribute little to the final pixel color according to the opacity transfer function, then drawing these samples from the distribution may be a waste of computation. Scalar values associated with higher opacities represent features of interest to the scientists while scalar values with lower opacity contribute little or not at all to the final pixel color. When the allowed sample count is limited, drawing more samples from scalar values with lower opacities means drawing less samples from scalar values with higher opacities. The feature of interest will be rendered with lower quality. Thus, sampling a data distribution should account for both the frequency of the scalar values as well as the opacity associated with the scalar values.

Figure 3.4 illustrates how using Monte Carlo sampling on the data distribution can lead to poor image quality because very few samples with nonzero opacity are drawn. The red curve represents the data distribution for a pixel sub-frustum. The green curve represents the opacity portion of the transfer function where a region of interest has been defined by the scientist between 0.6 and 0.9 with high opacity. The red dots are 100 samples drawn from the high frequency region in the data. Since these samples are not located in the scientist's region of interest, they will contribute little or not at all to the final rendering. Samples should have been drawn in the scientist's region of interest where the data distribution has lower frequencies in this example.

Our technique uses an importance distribution, which is shown by the blue curve in Figure 3.4. This distribution increases the chance to retrieve samples in the data that have high opacities. Equation 3.4 shows how to compute the importance distribution from the

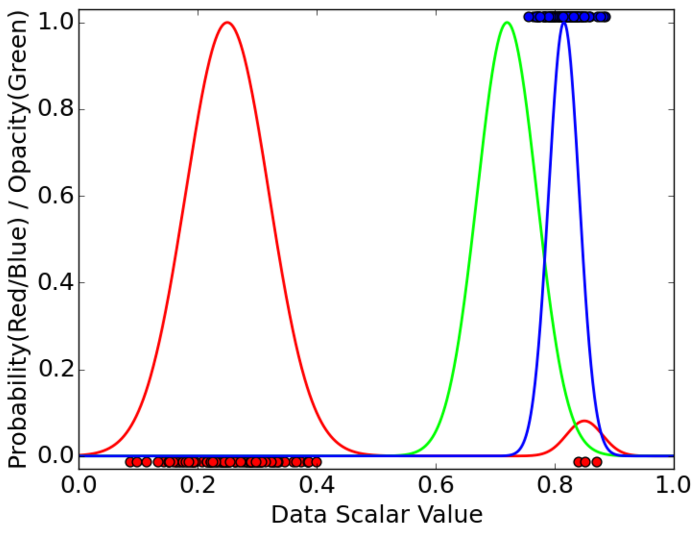


Figure 3.4: The green curve is the opacity function. The red dots are 100 samples drawn from a data distribution (red curve). Most red dots have no significant opacity, so most of the sampling time is wasted. The blue dots are 100 samples drawn from the importance distribution (blue curve), which incorporates both the data distribution (red curve) and the opacity function (green curve) by Equation 3.4. Most blue dots have significant opacity. In order to avoid occlusion, red and blue circles are drawn separately on the top and bottom of the figure. We normalize the highest peaks of the data distribution and importance distribution to 1 in order to visualize them clearly.

given opacity function and a pixel sub-frustum's distribution.

$$h_I(b) = h(b) * O(b) \tag{3.4}$$

where h is the histogram of a sub-frustum and O is the opacity function. $O(b)$ is the opacity for bin b where $O(b) = \int_{b_L}^{b_U} O(v)dv$. b_L and b_U are the value lower and upper bound of bin b. h_I is the importance distribution. The intuition of this equation is that only the scalar values with both high occurrence probability and opacity have higher importance. When sampling on the importance distribution notice that samples are drawn from the region of interest, as shown by the blue dots in Figure 3.4.

3.3.2 Re-sample Count

A larger re-sample count can better reconstruct from the distribution, but will result in longer rendering time. This is because the number of samples drawn from a distribution is propotional to the rendering time. Thus, the re-sample count needs to be manageable for rendering on a low cost platform for effective data analysis. Decreasing the total re-sample count, s^p in Equation 3.3, and using Equation 3.3 to assign each sub-frustum the re-sample count may lead to poor reconstruction and image quality. This is because sub-frustums with higher randomness that contribute more opacity to the final image could have an insufficient re-sample count. For an improved trade off between rendering speed and image quality, our approach gives a sub-frustum with a larger random importance distribution a higher re-sample count than a sub-frustum with a smaller random importance distribution. The following equation shows how to calculate the re-sample count for a pixel sub-frustum.

$$s'_{f_k^p} = s_{f_k^p} * (\frac{E(h_I)}{E^{max}})^c \tag{3.5}$$

where $s_{f_k^p}$ is the re-sample count for the sub-frustum f_k^p from Equation 3.3, $E(h_I)$ is the entropy of the importance distribution, E^{max} is the maximal value of entropy and c is used to adjust the final re-sample count. If c is 0, $s'_{f_k^p}$ and $s_{f_k^p}$ are the same. c can be increased to reduce the total re-sample count and achieve faster rendering speed. When c increases, the re-sample count of the sub-frustum with less random importance distribution will drop much faster than that of the sub-frustum with a larger random importance distribution.

Although Shannon entropy is the standard way to evaluate the randomness of a distribution, entropy computation is time-consuming. To efficiently determine the re-sample count, we use the number of bins with non-zero frequency in the histogram to approximate randomness. Equation 3.5 can be rewritten in the following way

$$s'_{f_k^p} = s_{f_k^p} * (\frac{nzb}{B})^c \tag{3.6}$$

where *nzb* is the number of bins with non-zero frequency in the importance distribution and B is the total number of bins. The scientist can simply adjust c to control the trade-off between rendering speed and image quality.

3.3.3 Opacity Function Modulation

Color blending samples drawn from the importance distribution with the adjusted re-sample count and then applying the samples to the transfer function directly without modulation can result in incorrect images. Some sub-frustums will be too opaque or too transparent. Hence, the opacity function should be modulated before its application to the samples so that each sub-frustum has the same opacity as the sub-frustum rendered by the Monte Carlo sampling algorithm. Equation 3.7 is used to estimate the opacity of a sub-frustum rendered by the Monte Carlo sampling algorithm when the samples are well reconstructed by Monte Carlo sampling with a sufficient re-sample count.

$$A_{f_k^p} = 1 - \prod_{b=1}^{B} (1 - O(b))^{s_{f_k^p} * h^N(b)} \tag{3.7}$$

where O is the original opacity function, h^N is the sub-frustum's histogram after normalization where $\sum_{b=1}^{B} h^N(b) = 1$. $s_{f_k^p} * h^N(b)$ is the expected re-sample count for the scalar values in bin b's value range. Equation 3.8 shows the opacity of the sub-frustum when we use the adjusted re-sample count and draw samples from the importance distribution.

$$A'_{f_k^p} = 1 - \prod_{b=1}^{B} (1 - O'(b))^{s'_{f_k^p} * h_I^N(b)} \tag{3.8}$$

where $s'_{f_k^p}$ is the adjusted re-sample count in Section 3.3.2 and h_I^N is the normalized importance distribution where $\sum_{b=1}^{B} h_I^N(b) = 1$. So, $s'_{f_k^p} * h_I^N(b)$ is the expected re-sample count for scalar values in bin b's value range from the importance distribution and the adjusted re-sample count. In order to make $A_{f_k^p} = A'_{f_k^p}$, we adjust the opacity function by Equation 3.9.

$$O'(v_b) = 1 - (1 - O(v_b))^{(\ell/\ell')} \tag{3.9}$$

where $\ell = s_{f_k^p} * h^N(b)$ and $\ell' = s'_{f_k^p} * h_I^N(b)$ and v_b is a scalar value in bin b's value range. Algorithm 2 summarizes the rendering procedure introduced in this section.

Algorithm 2 Proxy rendering

```
1:  ▷ $p$: a pixel which contains histograms of sub-frustums
2:  ▷ $s^p$: given re-sample count of $p$
3:  ▷ $c$: parameter to adjust re-sample count
4:  ▷ $T^c$ and $T^o$: color and opacity function
5:  procedure ProxyRendering($p$, $s^p$, $c$, $T^c$, $T^o$)
6:      $color$ = (0,0,0,0)
7:      $pxlFS = PixelFreqSum(p)$            ▷ compute freqency sum of all histograms on p
8:      $K^p = GetNumOfHist(p)$
9:      for $k = 1 \to K^p$ do
10:         $h = GetHist(p,k)$                ▷ get the k-th histogram on p
11:         $h_I = ImpDistr(h, T^o)$          ▷ compute the importance distribution
12:         $s = (s^p * \frac{FreqSum(h)}{pxlFS}) * (\frac{Nzb(h_I)}{Bins(h_I)})^c$   ▷ adjust the re-sample count
13:         $T_k^o = AdjustOpacityFunc(T^o, s^p, s)$   ▷ modulate the opacity function
14:         for $i = 1 \to s$ do
15:             $v = ReSample(h_I)$
16:             $colorV = T^c(v)$
17:             $opacityV = T_k^o(v)$
18:             $color = blend(color, colorV, opacityV)$
19:         end for
20:         $CheckEarlyRayTermination(color)$
21:     end for
22:     return $color$
23: end procedure
```

3.3.4 Uncertainty Quantification and Visualization

Since our proxy reduces the size of a large dataset according to the scientist's given size budget, information loss is inevitable. The major information loss in the proxy comes from the loss of spatial information in the pixel sub-frustums, especially the loss of the samples' depth order. This is because each pixel sub-frustum is represented by a distribution of data, and the distribution does not store the samples' location information. This could give the imprecise depth cue of features. To avoid providing misleading information, our technique can show the uncertainty from the information loss to the scientists. We have two methods to quantify and convey the uncertainty to the scientists. The first method is to use probabilistic animations generated from our proxy rendering scheme. Our renderer keeps generating a new frame from the proxy of the same view and time step in an animation, and each new frame re-samples from the distribution over again. When the animation shows pixels with high color variability, this indicates distributions associated with these pixel have high randomness. Thus, the image should have less confidence on these pixels. This uncertainty visualization keeps the scientists informed of the error.

The second method to provide visualization of uncertainty in the result is to use static images. To compute the randomness of a sub-frustum, we first compute the importance distribution which considers the opacity function given by the scientist. Then, instead of computing the distribution's randomness at the scalar value domain directly, our technique transforms the importance distribution to YUV color space from the color portion of the transfer function. YUV color space is a well-known color space which is closer to human's perception. Y, U and V channel are all divided into U discrete intervals. This makes the YUV color space define a histogram with U^3 bins. In our system, the default U is set to 10. The randomness of the sub-frustum is computed by the entropy of the histogram in

Algorithm 3 Static uncertainty visualization

```
1: ▷ $p$: a pixel which contains histograms of sub-frustums
2: ▷ $s^p$: given re-sample count of $p$
3: ▷ $T^c$ and $T^o$: color and opacity function for data rendering
4: ▷ $T^c_U$: color map for uncertainty visualization
5: procedure StaticUncertaintyVisualization($p$, $s^p$, $T^c$, $T^o$, $T^c_U$)
6:     $color = (0,0,0,0)$
7:     $pxlFS = PixelFreqSum(p)$
8:     $K^p = GetNumOfHist(p)$
9:     for $k = 1 \rightarrow K^p$ do
10:        $h = GetHist(p,k)$
11:        $h_I = ImpDistr(h,T^o)$
12:        $h_I^{YUV} = TransformToYUVSpace(h_I,T^c)$        ▷ map scalar to YUV space
13:        $e = ComputeEntropy(h_I^{YUV})$
14:        $colorU = T^c_U(e)$                                ▷ look up color by e
15:        $opacityU = ExpectedOpacity(h,T^o,(s^p * \frac{FreqSum(h)}{pxlFS}))$
16:                ▷ compute the opacity of a sub-frustum in data rendering image
17:        $color = blend(color,colorU,opacityU)$
18:        $CheckEarlyRayTermination(color)$
19:    end for
20:    return $color$
21: end procedure
```

YUV space, then a color map is used to look up a color to represent the randomness of the sub-frustum. The color of each sub-frustum's randomness on a pixel is blended in the front-to-back order and the opacity of the sub-frustum's randomness is defined by the expected opacity of the sub-frustum in the image of data rendering, which is computed from Equation 3.7. Algorithm 3 shows the procedure to visualize the uncertainty of a pixel when a transfer function is given. With the method, the resulting visualization shows the impact of the loss of samples' depth information under the current transfer function. This means even if two sub-frustums' distributions have the same shape but represent different scalar value ranges, our technique displays higher uncertainty on the distribution whose range has higher color variety. Also, if samples or sub-frustums contribute more opacity in the data rendering image, we emphasize the uncertainty they generate in the uncertainty visualization. Our technique computes the randomness of each sub-frustum's distribution to quantify the uncertainty. Figure 3.5 shows a static uncertainty visualization example. In our system, users can switch between the volume image and the static uncertainty image by pressing a key to intuitively compare and identify regions with low and high confidence.

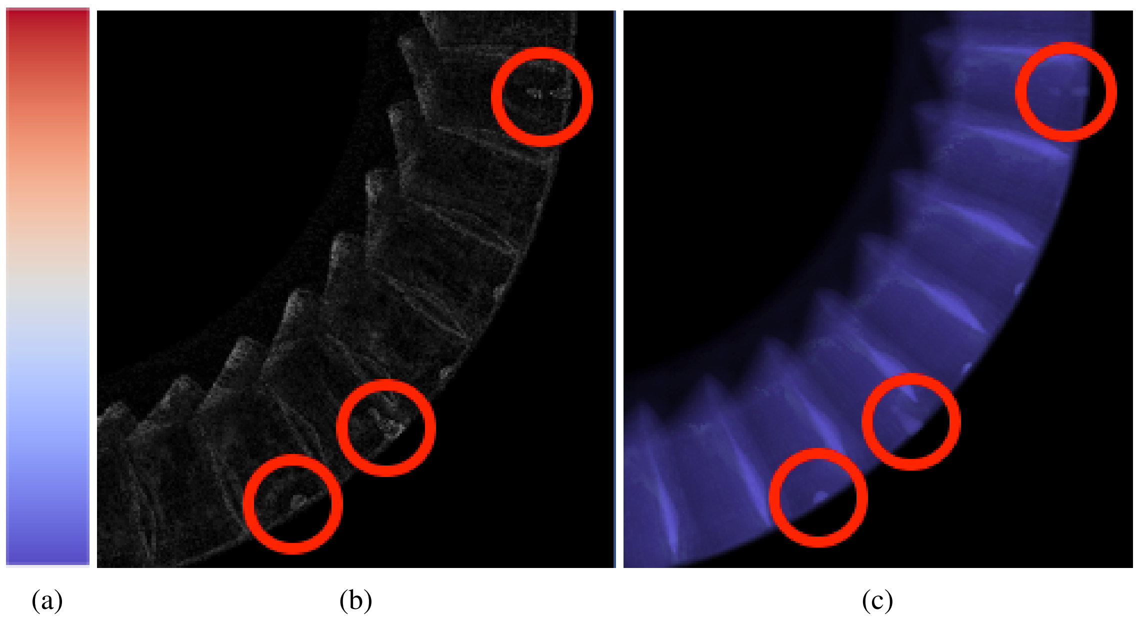


Figure 3.5: Static uncertainty visualization on the right bottom part of the 1st time step of Turbine dataset. (a) is the "Cool2Warn" color map which is used to visualize the uncertainty. The color closer to blue indicates low uncertainty and the color closer to white or red indicates higher uncertainty. (b) is a grayscale image calculated from the difference between images rendered by the raw dataset and our proxy, and a given transfer function. (c) is the uncertainty visualization from our technique. This image is calculated by Algorithm 3. We highlight the regions which has high error in (b) and the corresponding regions in (c).

3.4 Evaluation

We show results from our experiments utilizing four datasets to evaluate the performance of proxy generation and the quality of visualization from the proxy. The experiments were carried out on a machine with two 14-core Intel (Broadwell) Xeon E5-2680 v4 processors, 128 GB DDR3 Memory, one NVIDIA Tesla K80 GPU card. The post data analysis machine used to visualize the proxy is a desktop computer with an Intel 8-Core i-7-4770 3.40GHz CPU, 16 GB main memory, and an NVIDIA GeForce GTX 660 video card with 2 GB of memory. The Isabel dataset is a pressure field of Hurricane Isabel from the IEEE Visualization 2004 Contest. The raw data resolution is 500x500x100 with 48 time steps. We up-scaled its spatial resolution to 2200x2200x445 to perform the experiment (the total data size is 385 GB). The Plume dataset is a simulation of Solar Plume for thermal down flow on the surface layer of the Sun and was provided by the National Center for Atmospheric Research. We used the vector magnitude field in our experiments. The raw data resolution is 126*126*512 with 29 time steps. We up-scaled its spatial resolution to 756x756x3072 to perform the experiment (the total data size is 189 GB). The Combustion dataset was provided by Sandia National Laboratories where we used the mixture fraction field. The dataset has a resolution of 480x720x120 with 50 time steps. We scale up its spatial resolution to 1800x2700x450 to perform the experiment (the total data size is 407 GB). The Turbine dataset is a turbine engine compressor simulation, which was used in [23]. The original Turbine dataset is a curvilinear grid data, and its Pressure variable is re-sampled to a regular grid. The resolution of the regular grid Turbine dataset is 2036x2036x352 with 50 time steps(the raw size is 271 GB).

3.4.1 Proxy Size and Proxy Generation Time

This section presents proxy sizes and generation times with our approach. In the evaluation, we take six views from each dataset to produce the proxies. The six views are set up to look at the centers of six faces of the volume cube. All proxies in this section have a pixel resolution of 1024x1024. To precisely calculate the distribution for the proxies, we cast 16 sub-rays into the volume from each pixel. Each distribution used to represent a pixel sub-frustum is a histogram with 128 bins. We set 50MB as the size budget to generate a proxy for one view and one time step. Each dataset will generate the proxy with 3%-6% of the raw dataset size and covers the six given views and all time steps,. The total proxy size of each dataset is reported in Table 3.1.

The time reported in Table 3.1 is the average time to produce a proxy of one view and one time step. The average proxy generation time is computed as "(total proxy generation time + total entropy selection time) / (total views * total time steps)". The total generation time includes the entropy threshold selection time (Section 3.2.2) and the time for generating the proxy using the selected entropy threshold (Algorithm 1). The average of entropy selection time for the first time step is only calculated the average time to produce proxies in the first time step. The time in the right-most column in Table 3.1 is the average time to select the entropy thresholds after the second time step. Because the entropy thresholds of the consecutive time steps are similar and our technique uses the threshold from the previous time step as the initial guess and just slightly adjusts it, only the first time step spends longer time searching the proper entropy threshold.

Table 3.1: The proxy size and generation time.

	Size			Time (Average)		
	Size Budget (Per view/time step proxy)	Raw Data (Total)	Proxy (Total)	Proxy Generation	Entropy Selection (1st time step)	Entropy Selection (From 2nd time step)
Isabel	50MB	385GB	14.2GB	19.6s	42.3s	11.4s
Plume	50MB	189GB	8.9GB	23.2s	125.2s	17.7s
Combustion	50MB	407GB	15.2GB	27.6s	112.2s	16.7s
Turbine	50MB	271GB	15.3GB	44s	203s	28s

3.4.2 Image Quality and Rendering Time

We evaluate the quality of rendering images in this section. We choose one view from the proxies generated in Section 3.4.1 to evaluate. The view of each dataset is shown in Figure 3.6, 3.7, 3.8 and 3.9. In this experiment, each dataset has their own color portion of the transfer function. Turbine and Plume use regular "Jet" color map as their color map.

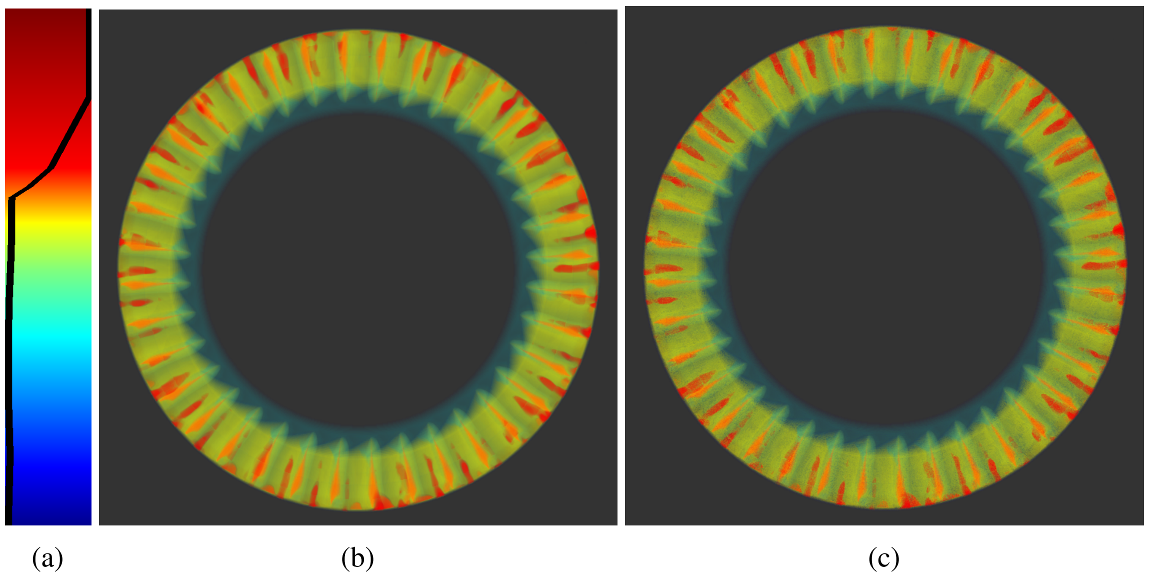


Figure 3.6: The 12th time step of Turbine dataset. (a) is the transfer function. The higher scalar value is at the top. The black curve is the opacity function and the right hand side indicates the highr opacity. (b) is the image rendered from the raw data. (c) is the image rendered from our proxy.

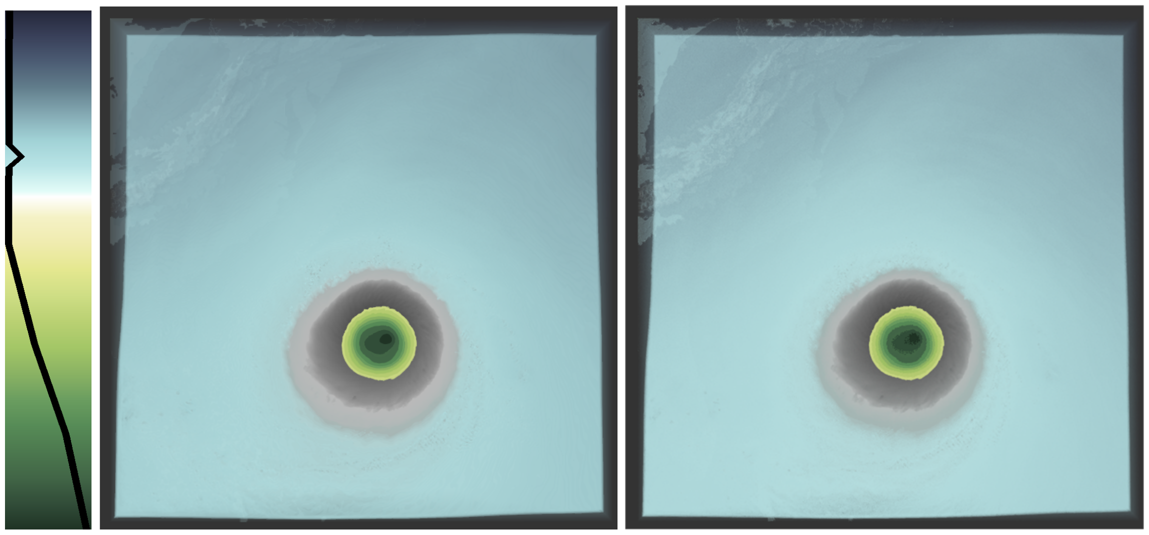

(a) (b) (c)

Figure 3.7: The 25th time step of Isabel dataset. (a) is the transfer function. The higher scalar value is at the top. The black curve is the opacity function and the right hand side indicates the highr opacity. (b) is the image rendered from the raw data. (c) is the image rendered from our proxy.

Isabel uses the color map “Asymmetric Blue/Green Divergent”. Combustion uses the color map “Asymmetric Blue/Orange Divergent”. The last two color maps are from the color map web page [1] of Data Science at Scale in Los Alamos National Laboratory. We design the opacity portion of the transfer function by the principle which gives one or two scalar value intervals higher opacity as the value of interest and give other scalar values low opacity as the context. This is one of the common transfer function design principles when exploring the datasets. The actually transfer function for each dataset is shown in Figure 3.6, 3.7, 3.8 and 3.9.

To evaluate the image quality, we calculate the peak signal-to-noise ratio (PSNR) [43] between the image rendered from our proxy and the ground truth image which is rendered from the raw dataset. The higher PSNR indicates that the tested image is more similar to

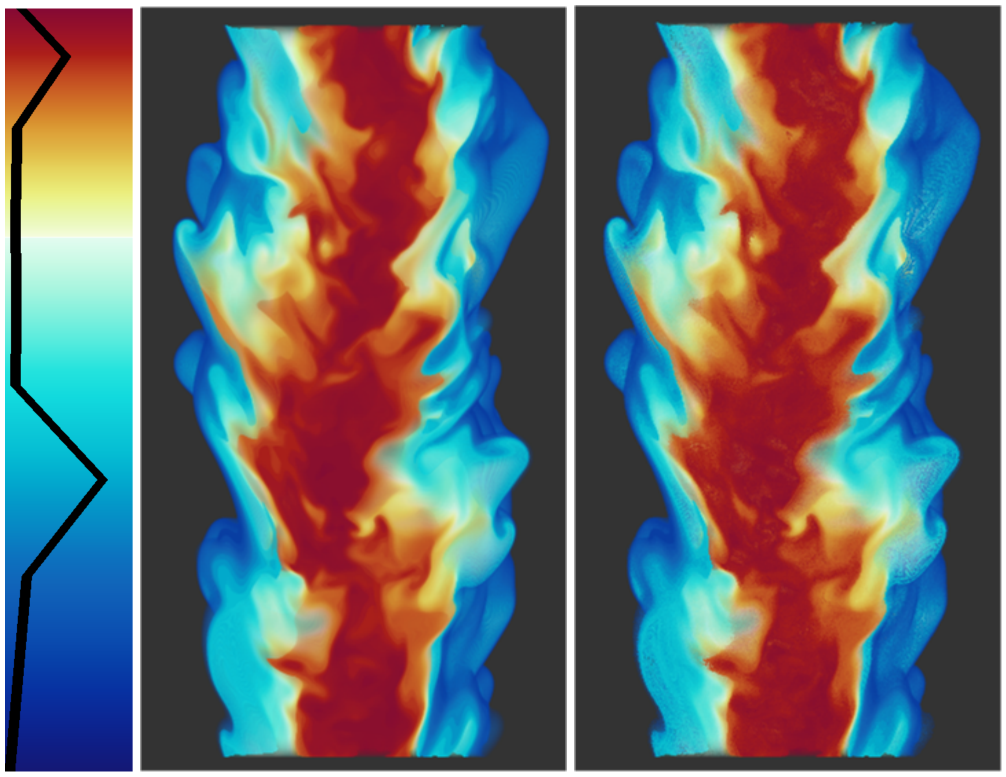
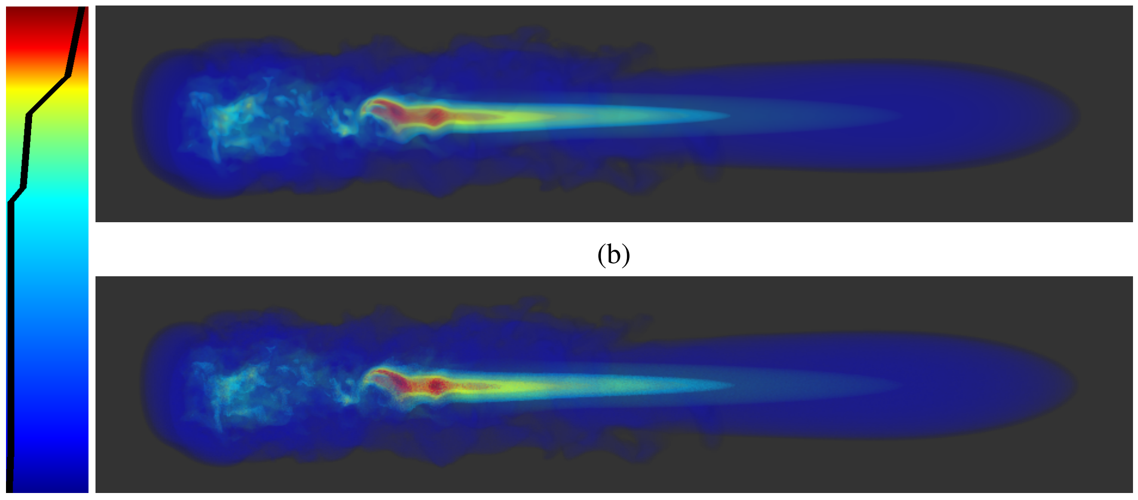


Figure 3.8: The 35th time step of Combustion dataset. (a) is the transfer function. The higher scalar value is at the top. The black curve is the opacity function and the right hand side indicates the highr opacity. (b) is the image rendered from the raw data. (c) is the image rendered from our proxy.

the ground truth image. The average PSNR of all time steps of a dataset is calculated and reported. Turbine, Isabel, Combustion and Plume have 37.07, 35.97, 33.29 and 43.71 db PSNR, respectively. Figure 3.6, 3.7, 3.8 and 3.9 also visually show that the image rendered by our proxy is similar to the ground truth images. The average rendering times of all time steps using the transfer function in Figure 3.6, 3.7, 3.8 and 3.9 to render are also reported. The average rendering time of Turbine, Isabel, Combustion and Plume are 0.19, 0.12, 0.078 and 0.075 second, respectively.


(a)

(b)

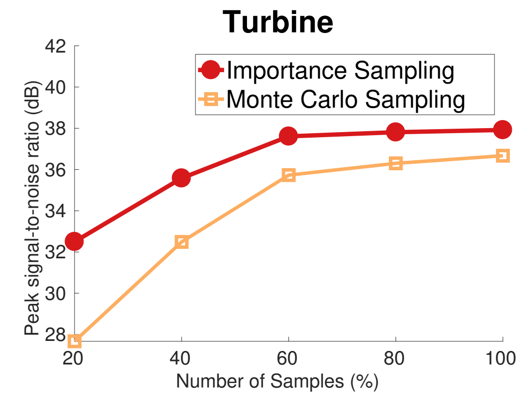
(c)

Figure 3.9: The 10th time step of Plume dataset. (a) is the transfer function. The higher scalar value is at the top. The black curve is the opacity function and the right hand side indicates the highr opacity. (b) is the image rendered from the raw data. (c) is the image rendered from our proxy.

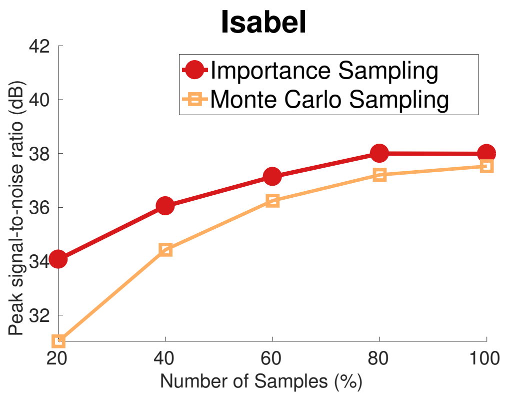
3.4.3 Importance Sampling

Information in the importance distribution can help to reduce the number of samples drawn by drawing more samples that have a high frequency in the data space and in the regions of interest and less samples otherwise. Figure 3.10 shows PSNR comparisons between importance sampling and Monte Carlo sampling using different data sets and using different numbers of samples. The proxies used in these experiments are the same as the proxies in Section 3.4.2. The PSNR is the average PSNR of all time steps. The plots show that the quality of images rendered by importance sampling decreases more slowly than the quality of images rendered by the traditional Monte Carlo sampling. Hence, using importance sampling can keep the image quality when scientists decrease the number of re-sample count to gain fast rendering and interaction speed on the post analysis machines.

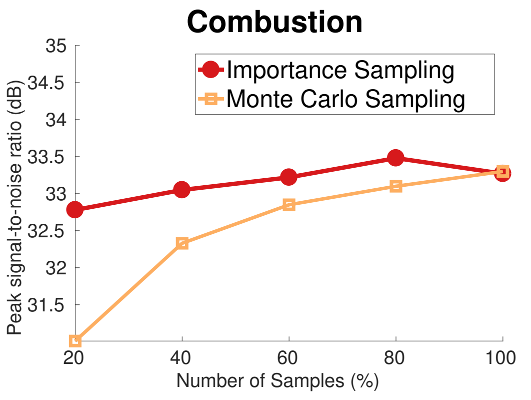
Note that 100% number of samples in Figure 3.10 indicates the re-sample count is the sames as the re-sample count used in Section 3.4.2.



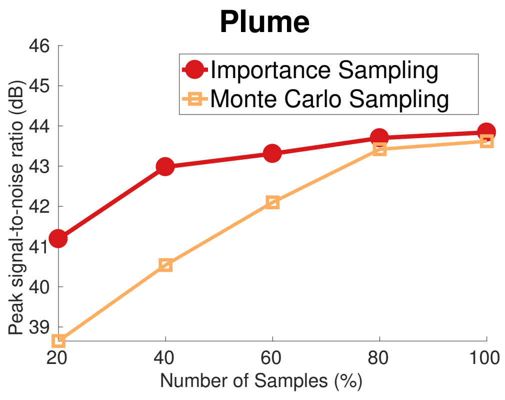
(a)



(b)



(c)



(d)

Figure 3.10: Image quality comparison between Importance sampling and basic Monte Carlo sampling for varying numbers of re-sample count.

3.5 Discussion

3.5.1 Transfer Function

The rendering quality is not only affected by the proxies but also affected by the transfer function in use. Hence, it is worth to study the impact from transfer functions. We firstly study the opacity portion of the transfer function. We use the same datasets, color portion of the transfer function and proxies in Section 3.4.2, but give different opacity functions to run the experiment. Table 3.2 shows the average PSNR of all time steps from all datasets with different opacity functions. The opacity function 1 (OF1) is the sames as the opacity function used in Section 3.4.2. The opacity function 2 (OF2) is designed to give all scalar values high opacity (0.85). It means only the samples close to the surface of the volume cube contribute to the final color of the image. The opacity function 3 (OF3) is designed to give all scalar values low opacity (0.05). It means most samples in the data contribute to the final color of the image. Table 3.2 shows that the PSNR could be either much better or worse than the PSNR in OF1 when the scalar values have high opacity (OF2). Some datasets have worse PSNR because the pixel color can be very different if the depth order of samples is incorrect. Some datasets, e.g. Plume, have better PSNR because the samples close to the surface of the volume cube have similar color and they occlude all other samples. Hence, the image quality highly depends on the data and the color map design when the opacity function gives samples high opacity. Comparing with OF2, OF3 has constantly similar or better PSNR than OF1. This is because the sample does not occlude too much color from the samples behind it, the neighboring samples along the pixel ray will contribute similar intensity of color to the image. So, even if our proxy loses the samples' depth order in the local region defined by a pixel sub-frustum, the incorrect samples order does not have huge impact in this case. We also evaluate the image quality on different transfer functions.

Table 3.2: Quality (PSNR) of images rendered by different transfer function. Reported numbers of OF1, OF2 and OF3 are the average PSNR of all time steps of a dataset. Reported numbers of RTF are the mean and variance from PSNRs of 10 random transfer functions.

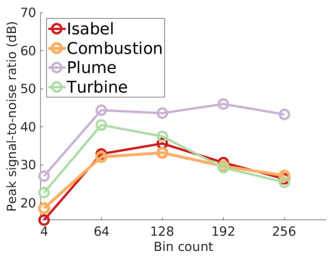
	Image Quality			
Dataset	OF1	OF2	OF3	RTF
Turbine	37.07	17.14	34.08	30.24 (0.19)
Isabel	35.97	25.28	37.55	36.69 (0.38)
Combustion	33.29	32.08	35.53	39.05 (0.51)
Plume	43.71	51.09	46.43	48.98 (0.71)

We randomly pick 10 different transfer functions for each dataset to evaluate the image quality. The color portion of a random transfer function is chosen from the pre-defined color maps in VTK-m library [65]. To determine the opacity portion of a transfer function, we uniformly pre-define 11 scalar values to randomly select the opacity values and use the linear interpolation to compute the full opacity function. We report the mean and variance of PSNRs from 10 random transfer functions in Table 3.2.

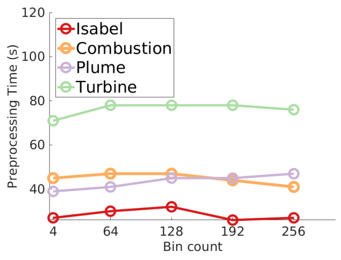
3.5.2 Bin Count of Histograms

The bin count of histograms directly affects the image quality, preprocessing time and rendering time. Figure 3.11 shows the image quality, preprocessing time and rendering time when the bin count of the histogram changes. This test is carried out by generating the proxies at the views in Figure 3.6, 3.7, 3.8 and 3.9 for the four datasets.

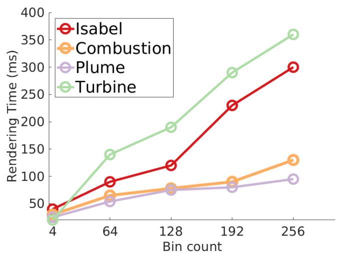
In Figure 3.11(a), we generate proxies with different bin counts by giving each time step proxy a 50MB budget. Transfer functions in Figure 3.6, 3.7, 3.8 and 3.9 are used to generate images and the PSNR of a dataset and a bin count is the average PSNR of all time steps. In the figure, we observe that images have best PSNR when the bin counts are 64 or 128. If the bin count is smaller, the PSNR decreases. This is because each bin represents a wider scalar



(a)



(b)



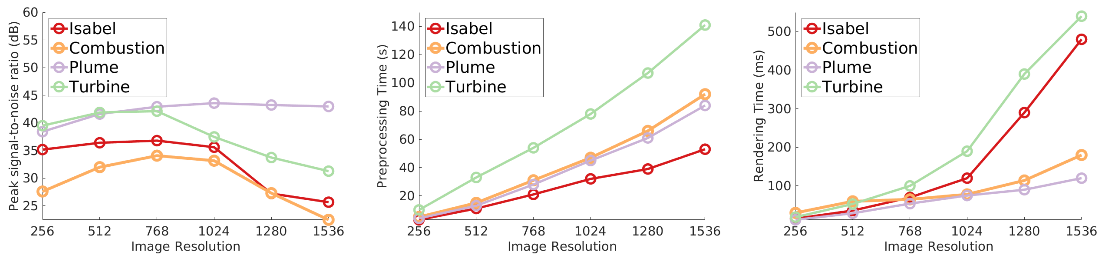
(c)

Figure 3.11: The impact of different histogram's bin counts on the image quality, proxy generation time and rendering time.

value range which may map to quite different colors or opacity values. If the bin count is very large, the PSNR also reduces. This is because each pixel sub-frustum's histogram costs more storage and the proxy loses more information of samples' depth order if the same size budget of the proxy is given. According to this test, the bin count selection is a trade-off between the accuracy of samples' depth-order information and re-sample scalar values.

We also evaluate the proxy generation time and the rendering time when the histogram's bin count changes. Figure 3.11(b) shows that the proxy generation time does not have significant difference when the bin count increases. This is because the proxy generation time is mainly controlled by the number of samples taken from the raw dataset to generate the proxy. Computing the sub-frustum's histograms with different bin counts from the same number of samples does not significantly change the proxy generation time. However, Figure 3.11(c) shows that the rendering time increases when the larger histogram's bin count is used. This is because the time complexity of sampling a value from a histogram is $O(B)$ where B is the bin count. The re-sample process takes a longer time if the histogram's bin count increases.

3.5.3 Image Resolution



(a) (b) (c)

Figure 3.12: The impact of different image resolutions on the image quality, proxy generation time and rendering time. In this test, the image width and height are the same. The X-axis in each subfigure is the image width and height.

In this subsection, we discuss the impact of different image resolutions on the image quality, preprocessing time and rendering time. We fixed the histogram's bin count to 128 and varied the image resolution. The other test settings are the same as that in Section 3.5.2. Figure 3.12(a) shows that the image quality decreases when the image resolution increases. This is because each pixel ray has less size budget if the image resolution increases and a proxy budget is fixed. When each pixel ray has less size budget, each histogram will represent a longer sub-frustum and has more losses in the samples' depth order information. However, the image quality also slightly reduces when a smaller image resolution is used. In this case, although each pixel ray has more size budget and the proxy loses less samples' depth order information, each pixel sub-frustum covers more sub-rays and increases the randomness of a sub-frustum's histogram. This also slightly increases the uncertainty and error in the re-sample process. Figure 3.12(b) and (c) show the preprocessing time and rendering time if different image resolutions are used. Both the preprocessing and the rendering time increase when the larger image resolutions are used because more pixels have to been processed.

3.5.4 Size Budget

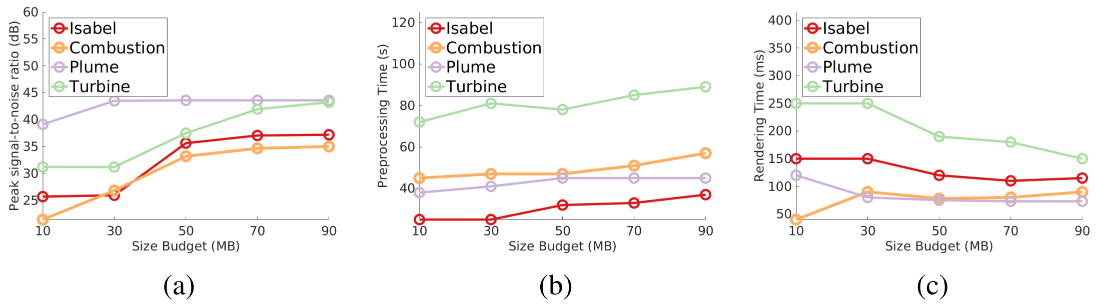


Figure 3.13: The impact of different size budgets on the image quality, proxy generation time and rendering time.

We also varied the size budget to evaluate the impact on the image quality, proprocessing time and rendering time. We use 10MB, 30MB, 50MB, 70MB and 90MB as the size budget for one view and one time step to carry out this test. The histogram' bin count and image resolution are fixed to 128 and 1024x1024, respectively. The other test settings are the same as that in Section 3.5.2. Figure 3.13(a) shows that PSNRs of all datasetes are monotonically increasing while a larger size budget is given. One interesting observation in this figure is that the left most two points on both Turbine's and Isabel's curves are the same. This means that the image quality does not increase even if the size budget increases from 10MB to 30MB. We examine these proxies and find that all pixels are only represented by one histogram. This is because the proxy size for one time step and one view exceeds the giving size budget even if each pixel is only represented by one histogram. So, the actual proxy size will exceed the giving size budget if the size budget is too small. For example, the actual proxy size of the Isabel's proxy generated by 10MB size budget is 38.4MB because this is the minimal storage cost when each pixel is only represented by one histogram. We can also observe that the image quality improves if we give 50MB size budget because this size budget is greater than 38.4MB and the proxy can represent a ray by multiple histograms to reduce the samples' depth order loss. In addition, we can also observe that the right most three points on Plume's curve are almost the same. In this case, because 50MB budget is enough to divide a pixel's frustum into fine-grained sub-frustums, increasing the size budget does not significantly improve the image quality. Figure 3.13(b) shows that the preprocessing time slightly increases while a larger size budget is given. When the larger size budget is given, each ray produces more sub-frustum histograms and the process is slowed down a little bit. Figure 3.13(c) shows that, in general, the rendering time decreases while more size budget is given. Because the entropy threshold selection selects a smaller entropy threshold when a larger budget is given. This makes each histogram has less non-zero bins. To draw a sample from a histogram with less non-zero bins spends less time because only the non-zero bins have to be visited. Therefore, the rendering time is smaller because less time is needed to draw a sample from a histogram. However, we can also observe an exception on the Combustion's curve if we give only 10MB size budget. We examine this case and discover that this results from the early ray termination technique in the rendering pipeline. Early ray termination is a popular technique to finish the rendering process of a pixel and save the computation if the pixel is already fully opaque. In this case, due to a small size budget is given and the proxy loses a lot of samples' depth order information, high opacity samples which are far away from the view point in the raw data could be drawn early and terminate the pixel rendering process.

3.5.5 Parameter Selection

In our approach, we have to select several parameters to produce the proxy. The first one is view point selection for the proxy generation. Usually, the scientists may have prior-knowledge to select preferred views to observe the datasets. However, if prior-knowledge is not available, one alternative approach is to use the geodesic grid [15] to sample the views around the datasets. This provides an uniform view coverage around the dataset. The other alternative is to take a few time steps and apply the view selection techniques [12, 44]. The view selection techniques select salient views by analyzing the dataset to generate proxies.

The histogram's bin count is the other important parameter which has to be selected before generating proxies. As the test in Section 3.5.2, the bin count selection is a trade-off between the accuracy of samples' depth-order information and re-sampled scalar value. Hence, to select the histogram's bin count, the transfer functions which will be applied should be taken into account. The rule of thumb to select the bin count is that the value interval of each bin should map to as similar as possible colors and opacities in the desired transfer function designs and also the histogram uses as less as possible bin count. This way, we can minimize both the uncertainty of sample color drawn within a bin and the samples' depth information loss. In practice, scientists often have prior-knowledge about the dataset and the scalar value of interest. Scientists often map the values of interest to more different colors. Hence, scientist can give different scalar value intervals different bin counts. For example, the range of interesting scalar values should receive more bin count than other value ranges. Thus, our technique can utilize the size budget more efficiently and generate images with better quality by the scientists' prior-knowledge. But, if the scientist does not know the value interval of interest, we can uniformly give entire value range the same bin count as how we run the experiments in Section 3.4.

3.6 Conclusion

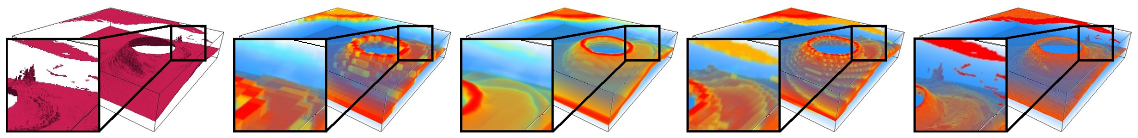
This paper presented an image and distribution-based representation for large scale data analysis, which allows transfer function exploration and uncertainty quantification. Distributions are used to compactly store the data on each pixel ray and each ray is irregularly subdivided to minimize the randomness of each distribution and preserve the visualization quality. Our approach generates the proxy by the scientist's selected views and the storage size budget that the post analysis machine's storage bandwidth and capacity can afford. The analysis and visualization are carried out on the post analysis machine by accessing these compact proxies only. The uncertainty of the volume image is visualized by either the static images or flickering animation to scientists.

Chapter 4: Statistical Visualization and Analysis of Large Data Using a Value-based Spatial Distribution

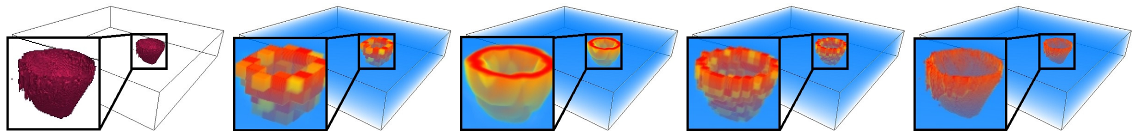
The computational power of modern supercomputers allow scientists to model physical phenomena with high-resolution simulations. Data-driven analysis and visualization techniques help scientists understand the dataset with greater depth and create more precise prediction models. However, analyzing such large-scale scientific simulation data is challenging due to the incompatibility between memory limitations, I/O capacities, and high computational power [8,54]. Reducing the size of the dataset is a viable option for visualization and analysis on commodity hardware, but information loss in the proxy representation must be controlled. Approaches to construct a proxy include lossy compression [29,36,102], subsampling [81], and statistical based data representations [7,24,56,84]. The latter approach is increasingly becoming more popular not only for data reduction but also due to the ability to retain important statistical features.

Many statistically based data representations partition the data domain into non-overlapping block regions, where the scalar field within each block is represented by a statistical summary of the scalar field, e.g. the mean or standard deviation of the scalar values within the block. Storing a distribution of the scalar values, such as a histogram, better represents the scalar field than simple summary metrics, but still suffers from limited accuracy. Though recent work on uncertain volume rendering and isosurface extraction [6,7,42,56,84] has tried to address this issue, they have not tackled the inherent problem of the loss of spatial information in the distribution. Thus, estimating the value at a position within a block using its corresponding distribution will incur inevitable approximation errors that are difficult to overcome.

We present a novel distribution based representation that incorporates spatial information at a small additional storage cost. The dataset is partitioned into blocks, where the data in each block is used to construct two types of distributions. One is a value distribution on the scalar values. The other is a spatial distributions where the locations of samples within the block are collected and stored as a multi-dimensional distribution for each value sub-range. Each multi-dimensional distribution is stored using a Gaussian Mixture Model (GMM) and hereafter referred to as a Spatial GMM. Whereas current approaches that use GMMs map data values to probabilities, our Spatial GMM maps the locations of the data points in different value ranges to probabilities. An adaptive scheme is employed to determine the number of Gaussian components that are required for each Spatial GMM, which makes the total size to store Spatial GMMs smaller than using a fixed number of Gaussian components. Given an arbitrary location, we utilize our representation to infer the probability for a value to reside at this location using Bayes' rule, which combines known information (the value distribution) and additional evidences (the Spatial GMMs) from a given condition. Equipped with this spatial information, our approach produces lower variance, and hence lower uncertainty, in the results of statistical based analysis and visualizations. Figure 4.1 shows an example of using our approach to provide clearer and more accurate uncertain isosurface rendering when compared with recently proposed distribution-based data representations.



(a) True isosurface (b) Block hist. (c) Block hist. w/ INTRPL (d) Block GMM (e) Our approach



(f) True isosurface (g) Block hist. (h) Block hist. w/ INTRPL (i) Block GMM (j) Our approach

Figure 4.1: Our *value-based spatial distribution* representation includes spatial information (called a Spatial GMM) that is compact, which is lacking in current distribution based representations. We illustrate our approach with volume rendering of the probability field for the location of an isosurface of the Isabel dataset (500x500x100). Figures (a) and (f) show the ground truth isosurface rendering using +90Pa and -900 Pa, respectively. Using a hot to cold transfer function, where red indicates a high probability for the isosurface and blue indicates a low probability, the rest of the columns show volume renderings of our approach (the last column) compared with current approaches. Equipped with added spatial information, our representation is able to better identify with higher certainty the location of the isosurface including small details, which are missed in the other representations. The block sizes used to compute the renderings in the last four columns are 12^3, 12^3, 6^3 and 16^3, respectively. In our approach, the value histogram costs 3.5MB and the Spatial GMM costs 3.26MB.

The rest of the chapter is organized as follows. Section 4.1 provides an overview of our approach. Our representation is presented in Section 4.2. Section 4.3 discusses the use of Bayes' rule to integrate the Spatial GMM and the block distribution. Section 4.4 presents quantitative experiments using Root Mean Square Error (RMSE) to compare our representation with other distribution-based representations. We demonstrate in Section 4.5 that our approach provides a marked improvement in visual quality when applied to a variety of visualization and analysis tasks. Section 4.6 presents timings to construct and evaluate our representation as well influence of parameters and the scalability of our approach. Section 4.7 concludes the paper.

4.1 Overview

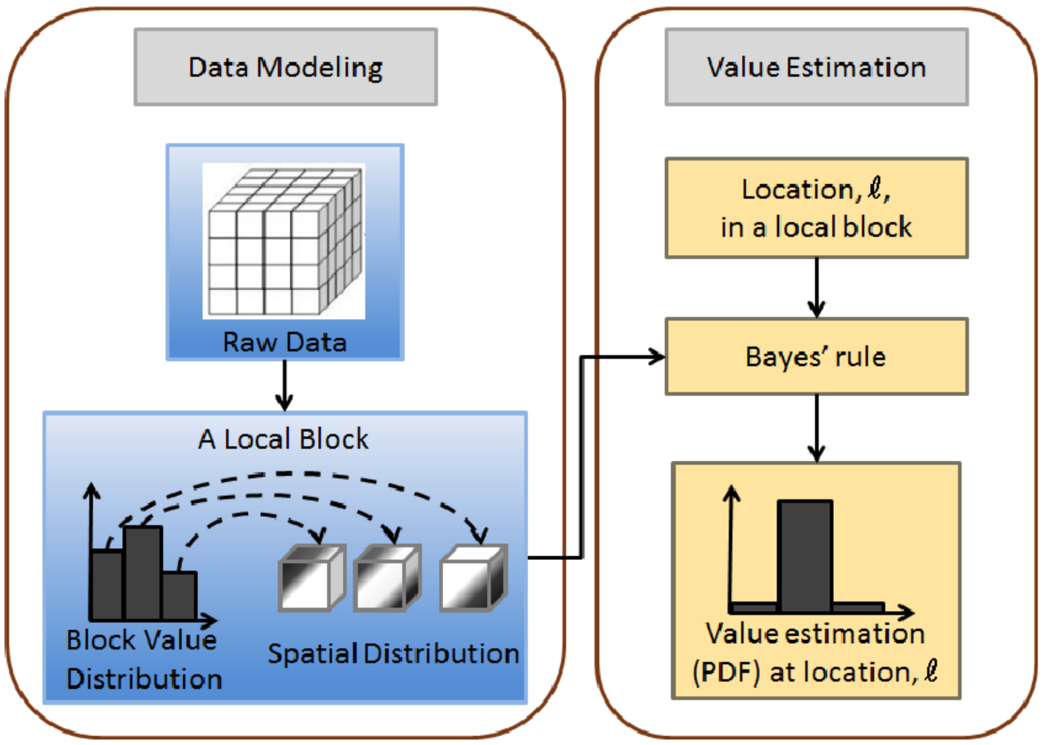


Figure 4.2: Overview of our approach. In a pre-processing step (shown on the left), the dataset is subdivided into blocks and both a value distribution and spatial distributions (Spatial GMMs) are computed for each block. To estimate a value at a given location, Bayes' rule is used to integrate the value and spatial distributions to obtain the probability density function of values at this location (shown on the right).

Our representation combines two kinds of distributions, the value distribution and spatial distributions. Figure 4.2 illustrates our representation and approach where our augmented representation is used to approximate a data value at a given location. The representation is constructed in a pre-processing step where the dataset is first subdivided into blocks. For each block, a value distribution is computed from the values of the samples in the block. This is a histogram consisting of B bins, where each bin represents a data value range as an interval. If b_i denotes the i^{th} bin, then the bin interval for b_i represents a data value range $[Lower_{b_i}, Upper_{b_i}]$. If M denotes the number of grid sample points in the block that have a value within the interval $[Lower_{b_i}, Upper_{b_i}]$ and N denotes the total number of grid sample points of the block, then the i^{th} bin has a probability of M/N. In addition, at each bin a distribution of spatial locations is computed from the grid sample locations in the block, which provides the probability of the locations for the data values in the bin interval. Each spatial distribution modeled with a GMM, called a Spatial GMM, whose dimensionality is determined by the spatial dimensions of the data. For example, a 3D dataset will define Spatial GMMs with three dimensions. Details of the spatial distribution are introduced in Section 4.2. To determine a probability density function (PDF) to use for value estimation (see Section 4.3), we use Bayes' rule by integrating information from the value distribution and the Spatial GMMs.

4.2 Spatial Distribution

In this section, we describe the Spatial GMM associated with each bin in the value distribution and show how to use the Gaussian Mixture Model to compactly represent the distribution. The number of Gaussian components to use for a Spatial GMM should vary from one Spatial GMM to another in order to accurately represent coherence of the data values across the block represented by the bin interval. Fewer components should be used when data values are distributed coherently in space. Figure 4.3 illustrates the process of computing the spatial GMM on a local data block. Details of the data structure are presented at the end of this section.

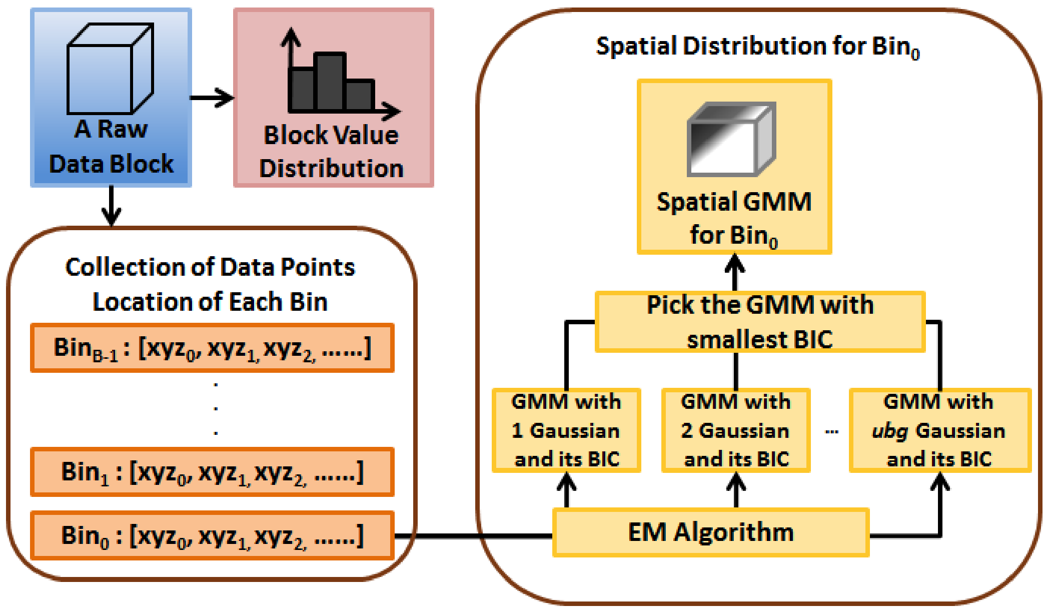

A Raw Data Block
Block Value Distribution
Collection of Data Points Location of Each Bin
Bin_{B-1} : [xyz_0, xyz_1, xyz_2, ...]
Bin_1 : [xyz_0, xyz_1, xyz_2, ...]
Bin_0 : [xyz_0, xyz_1, xyz_2, ...]
Spatial Distribution for Bin_0
Spatial GMM for Bin_0
Pick the GMM with smallest BIC
GMM with 1 Gaussian and its BIC
GMM with 2 Gaussian and its BIC
...
GMM with *ubg* Gaussian and its BIC
EM Algorithm

Figure 4.3: This diagram shows the steps used to compute the spatial GMM for a raw data block (shown in blue). Besides the computation of the value distribution, the raw data in the block is used to construct the Spatial GMM. First, the locations of the data samples are collected into the corresponding bin interval according to the data value at that location (shown in the bottom left). Then, a Spatial GMM is constructed (shown on the right) for each bin interval using the locations in the interval (illustrated here for Bin_0).

4.2.1 Spatial Distribution Per Bin Interval

The spatial distribution associated with a bin describes the probabilities of the locations for the values in the bin interval. The spatial distribution of a particular bin b_i can be modeled from the coordinates of the sample grid points whose scalar values belong to interval represented by bin b_i by using the Kernel Density Estimation (KDE) [38]:

$$S_{b_i}(x) = \frac{1}{N}\sum_{j=1}^{N} K_h(x - x_j) \tag{4.1}$$

where x is an arbitrary location in the block. Equation 4.1 computes S_{b_i}, which is the spatial distribution of bin b_i. $S_{b_i}(x)$ describes how likely x has a value within the interval b_i. Value N is the total number of grid points that have values within the bin interval b_i, x_j is the j^{th} grid point, and K_h is a kernel function.

4.2.2 Spatial Gaussian Mixture Model

A representation constructed directly from Equation 4.1 would incur a high storage cost to store information for N grid points. Instead, we can use the multivariate Spatial Gaussian Mixture Model [10], which compactly captures spatial coherence amongst similar data values, such as those in the same bin interval. The GMM is a widely used probabilistic model [24, 56, 101] because it can fit complex data well with a few number of parameters.

As opposed to the block GMM [24, 56], which maps a data value in the block to a probability, the Spatial GMM maps a spatial location to a probability. The Spatial GMM is a multivariate GMM where the dimensionality is determined by the spatial dimensions of the dataset. A Spatial GMM for a location x is expressed in Equation 4.2

$$SGmm_{b_i}(x) = \sum_{j=1}^{K} w_j * \mathcal{N}(x|\mu_j, \Sigma_j) \tag{4.2}$$

where $SGmm_{b_i}(x)$ is the Spatial GMM of bin b_i, K is the number of Gaussian components, and w_j, μ_j and Σ_j are the weight, mean and covariance matrix for the j^{th} Gaussian component. $\mathcal{N}(x|\mu_j,\Sigma_j)$ is the probability density at x of the Gaussian distribution, which is defined by μ_j and Σ_j. For a three dimensional dataset, μ_j are the location coordinates and Σ_j is a 3x3 covariance matrix. $\sum_{j=1}^{K} w_j$ has to be equal to 1.

To obtain the weights, means and covariance matrices, of the Spatial GMM, we use the Expectation-Maximization (EM) algorithm [10] where the locations of the grid points whose values are within the interval of bin b_i are collected as the input training samples to find the parameters of $SGMM_{b_i}$. The EM algorithm iteratively maximizes the likelihood function described in Equation 4.3

$$\text{argmax}_\theta SGmm_\theta(X) = \sum_{m=1}^{M} \log[\sum_{k=1}^{K} w_k \mathcal{N}(x_m|\mu_k,\Sigma_k)] \tag{4.3}$$

where θ represents the parameters of the Spatial GMM. The EM algorithm iteratively adjusts and finds the parameters of the Spatial GMM by maximizing the likelihood of input samples $X = \{x_1, x_2, ..., x_M\}$. M is the number of grid points whose values are within the bin interval and x_m is the location coordinate of these input samples. By maximizing Equation 4.3, the difference between S_{b_i} and $SGmm_{b_i}$ is minimized.

The EM algorithm may suffer from large computational overhead due to the number of iterations needed for the algorithm to converge. Though, the convergence rate of the algorithm depends on the initial parameters [46, 103], where good initial parameters result in fewer iterations needed for convergence. In general, finding good initial parameters is difficult. However in our case, we observe that the parameters $SGmm_{b_i}$ can be a good initial guess for parameters $SGmm_{b_{i+1}}$ because the locations of the samples belonging to neighboring bins b_i and b_{i+1} are usually close to each other in scientific datasets. We can leverage this property to help the EM algorithm to quickly estimate the parameters and save on training time.

4.2.3 Varying Number of Gaussian Components

An advantage of using the Gaussian Mixture Model is the ability to approximate any arbitrary target distribution. In general, using more Gaussian components can fit the target distribution better. Using too many Gaussian components may not significantly improve the approximation quality, but it will increase storage overhead costs and sometimes causes over-fitting.

In order to decide on a suitable number of Gaussian components to be used in a Spatial GMM, we use Bayesian Information Criterion (BIC) [70] to choose the number of Gaussian components. Given a model and data, i.e. a Spatial GMM and the input training samples, BIC will reward with a high likelihood over training samples and penalizes with a higher number of Gaussian components. In our implementation, an upper bound on the number of Gaussian components, called *ubg*, is given manually before training the Spatial GMMs. We iteratively check the BIC scores by changing the number of Gaussian components used for training a Spatial GMM. The number of Gaussian components with the smallest BIC is then selected for that Spatial GMM because lower BIC scores mean a better result for fitting. Hence, each Spatial GMM can have a different number of Gaussian components, which is determined by the complexity of the input data. This scheme avoids using too many Gaussian components without gaining significant quality improvement compared to using a fixed number of Gaussian components.

4.2.4 Data Structure

The data structure for the Spatial GMM is illustrated in Figure 4.4. Each Gaussian component stores a weight, a mean (location), and a covariance matrix. Because the covariance matrix is symmetric, only the upper triangle of the matrix is stored. Each bin consists of the adjusted probability of the histogram, the Gaussian component count of the GMM, and the Gaussian components.

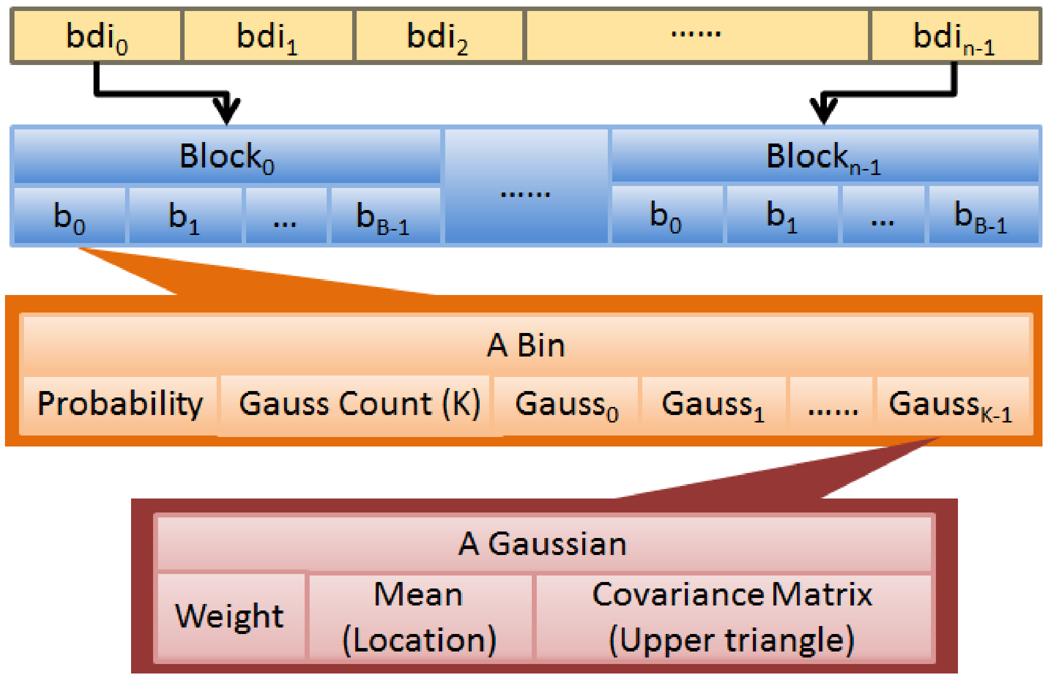


Figure 4.4: Data structure for our data representation using Spatial GMMs. Each block is indexed with a table of the starting locations of each block (shown at the top). Each block (shown in blue) stores bins (bin b_0 is shown in orange for $Block_0$). Each bin stores K Gaussians of the GMM for the bin. The data structure of a Gaussian component is shown in red.

Note that we store the adjusted probability instead of the original probability of the histogram in our data structure because this can preserve the correctness of the PDF computation, as described in the next section. Ideally, we would like the Spatial GMM to only model the space in a block region, but the Spatial GMM model is an approximate distribution representation and the domain is defined in infinite space. The Spatial GMM could give probabilities to coordinates outside the block and introduce bias in the PDF computation. Hence, we store the adjusted probability, $H'(b_i)$, to correct this, as shown in Equation 4.4

$$H'(b_i) = \frac{H(b_i)}{\int_{\Omega} SGmm_{b_i}(l)dl'} \tag{4.4}$$

where $H(b_i)$ is the original probability in the histogram, Ω is the spatial region of the block and $\int_{\Omega} SGmm_{b_i}(l)dl'$ is the accumulated probability over Ω of $SGmm_{b_i}$. In Section 4.3.1, we will show how to compute the PDF using $H'(b_i)$ and $SGmm_{b_i}$ in order to avoid the bias.

Different bins may have differing numbers of Gaussian components (see Section 4.2.3). So, a bin with zero probability requires storing zero Gaussian components. A homogeneous block, in which only one bin has probability, will not need the Spatial GMM. Sine the number of Gaussian components may differ per bin, the data size of each block varies as well, which leads to an irregular storage footprint. To easily access the data within blocks, an additional index table is stored where each index points to the starting location of the data in a block.

4.3 Probability of Data Values From Block and Spatial Distributions

This section describes how to to estimate a value at given a location within a block by combining information in the value distribution and the Spatial GMMs. Using Bayes' rule, we compute a probability density function (PDF) that provides probabilities for each possible value.

4.3.1 PDF Computation by Bayes' Rule

Bayes' rule is a popular theorem that is widely used in classification problems. It tells us how to rationally augment the known information with additional evidences from a given condition. In our scenario, the block value distribution is the known information and the additional evidences are the probabilities from each Spatial GMM at a given location. Bayes' rule is written as the random variable form

$$P(I = b_i | L = \ell) = \frac{f_L(\ell | I = b_i) P(I = b_i)}{f_L(\ell)} \tag{4.5}$$

$$f_L(\ell) = \sum_{k=0}^{B-1} f_L(\ell | I = b_k) P(I = b_k) \tag{4.6}$$

where I is a random variable that represents a bin index, b_i is the i^{th} bin index, L is a random variable that represents a location in the spatial space of the block, and ℓ is the location in question. Given a data value within the interval of bin b_i, $f_L(\ell | I = b_i)$ represents the probability that the value of b_i is at the location ℓ. $P(I = b_i)$ is the probability of bin b_i of the value distribution. $f_L(\ell)$ sums up the numerator term in Equation 4.5 over all bins b_i for normalizing $P(I = b_i | L = \ell)$.

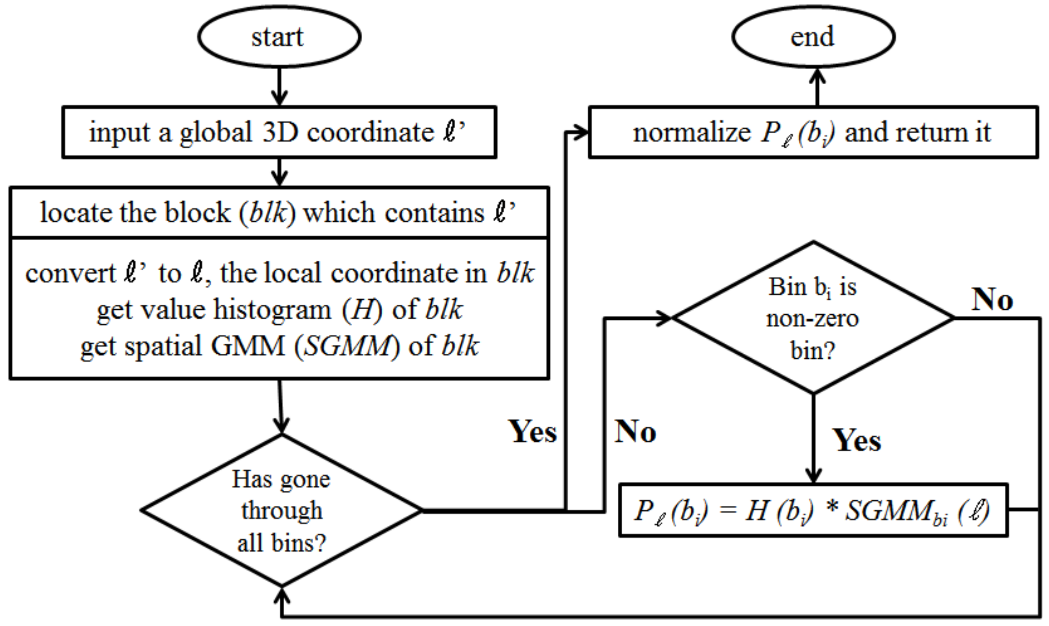

start
end
input a global 3D coordinate ℓ'
normalize $P_\ell(b_i)$ and return it
locate the block (*blk*) which contains ℓ'
convert ℓ' to ℓ, the local coordinate in *blk*
get value histogram (*H*) of *blk*
get spatial GMM (*SGMM*) of *blk*
Bin b_i is non-zero bin?
No
Yes
No
Yes
Has gone through all bins?
$P_\ell(b_i) = H(b_i) * SGMM_{bi}(\ell)$

Figure 4.5: This flow chart shows the steps to compute the PDF consisting of probabilities associated with possible data values at a given a location ℓ. The input of this algorithm is a 3D location ℓ.

The function $f_L(\ell|I = b_i)$ comes from the normalized Spatial GMMs, $\frac{SGmm_{b_i}(\ell)}{\int_\Omega SGmm_{b_i}(l)dl'}$, which ignores the probability outside the block region. $P(I = b_i)$ comes from the value distribution. Equation 4.5 can be rewritten as Equation 4.7

$$\begin{aligned} P_\ell(b_i) &= \frac{\frac{SGmm_{b_i}(\ell)}{\int_\Omega SGmm_{b_i}(l)dl'} * H(b_i)}{\sum_{k=0}^{B-1} \frac{SGmm_{b_i}(\ell)}{\int_\Omega SGmm_{b_i}(l)dl'} * H(b_k)} \\ &= \frac{SGmm_{b_i}(\ell) * H'(b_i)}{\sum_{k=0}^{B-1} SGmm_{b_k}(\ell) * H'(b_k)} \end{aligned} \tag{4.7}$$

where $P_\ell(b_i)$ is the probability that the data sample at location ℓ has a value that belongs to the value sub-range of bin b_i, $H(b_i)$ is the probability associated with bin b_i in the value distribution, and P_ℓ is a PDF, which can be used to estimate the probabilities of various data values at location ℓ. The denominator normalizes $\sum_{k=0}^{B-1} P_\ell(b_k)$ to 1. $H'(b_i)$ is the adjusted probability of the histogram, which is mentioned in Section 4.2.4. The flow chart shown in Figure 4.5 illustrates how to compute the PDF at a given location ℓ.

4.4 Quantitative Evaluation

We show results from our experiments utilizing four datasets to analyze the storage cost of our representation and the quality of the value estimation. We used an Intel 8-Core i-7-4770 3.40GHz CPU with 16 GBs of main memory and an NVIDIA GeForce GTX 660 video card with 2 GBs of video memory in our experiments. The Plume dataset was provided by the National Center for Atmospheric Research and is a simulation of Solar Plume for thermal down flow on the surface layer of the Sun. The data resolution is 504x504x2048 (the raw size is 1984 MB) and where we used the vector magnitude field in our experiments. The Isabel dataset is a pressure field of Hurricane Isabel from the IEEE Visualization 2004 Contest with a resolution of 500x500x100 (the raw size is 95 MB). The Combustion dataset was provided by Sandia National Laboratories and the mixture fraction field was used. The dataset has resolutions of 480x720x120 (the raw size is 158 MB). The Turbine dataset is a turbine engine compressor simulation which was used in [24]. The original Turbine dataset is a curvilinear grid data, and its Pressure variable is re-sampled to a regular grid. The resolution of the regular grid Turbine dataset is 2545x2545x440 (the raw size is 10871MB).

4.4.1 Representation Quality

We illustrate the improved accuracy of our representation using the Spatial GMM over two popular distribution-based representations: the block histogram, also called Hixel in [84], and the block GMM [24, 56]. We also compare our representation with histogram based trilinear interpolation (described in [71]) applied to the block histogram. Since a fundamental flaw of the block histogram is loss of spatial information, interpolation may compensate for this to some extent.

To evaluate the quality of data value estimation at given locations, we use the Root Mean Square Error (RMSE) metric, which encodes both bias and variance [89] as shown in equation 4.8

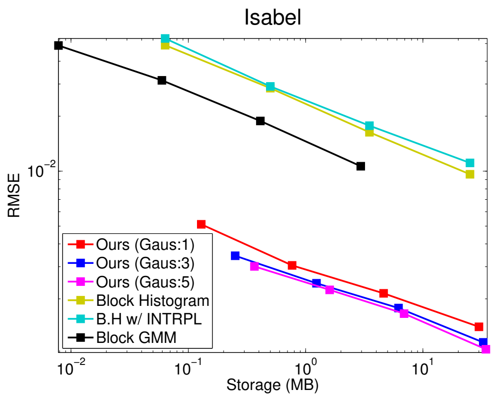
$$RMSE(P_\ell, x_\ell) = \sqrt{Bias(P_\ell, x_\ell)^2 + Variance(P_\ell)} \tag{4.8}$$

where the variable ℓ is a given location, P_ℓ is a PDF that indicates the possible values at location ℓ and their associated occurrence probability, and x_ℓ is the true value from the raw data. P_ℓ with a lower RMSE indicates that the estimate of a data value is closer to the true value and has smaller uncertainty.

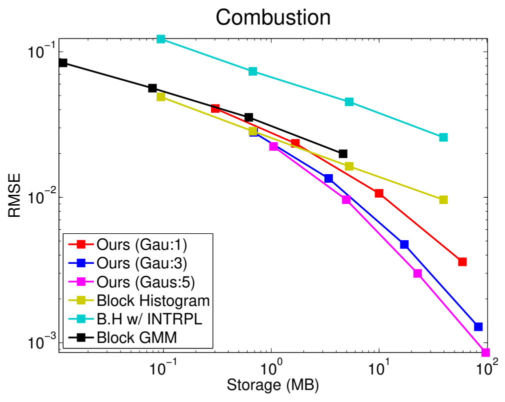
We use Equation 4.7 to compute the PDF P_ℓ using our representation. The block histogram and block GMM representations directly use the corresponding block distribution of location ℓ as the basis to estimate the data value at location ℓ. P_ℓ is the interpolated histogram at ℓ when we used the block histogram with interpolation approach. The RMSE is calculated directly from the PDF at a location ℓ and the true value, as shown in equation 4.9.

$$RMSE(P_\ell, x_\ell) = \sqrt{\int_{-\infty}^{\infty} P_\ell(v) * (v - x_\ell)^2 dv} \tag{4.9}$$

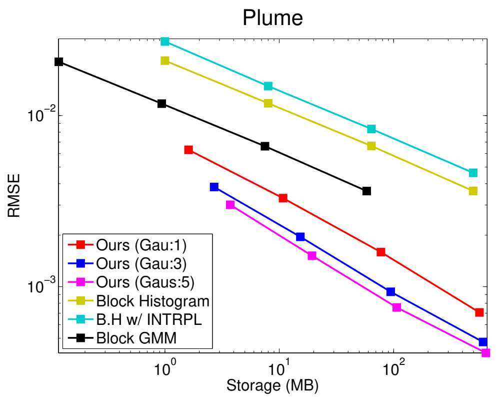
Figure 4.6 shows the improved estimation accuracy of using our Spatial GMM representation (bottom three curves) when compared with the the block histogram and block GMM distribution-based representations (top three curves). Each curve has four points that represent the block sizes used in the experiment. Each point is a plot of the storage cost vs. RMSE using a particular block size. The RMSE in Figure 4.6 is the average of the RMSE values at all grid points in the dataset. The histograms for all representations used 128 bins and the block GMM used 5 Gaussian components (as was used in [24]). The curves representing the block histogram with and without interpolation have the same storage cost



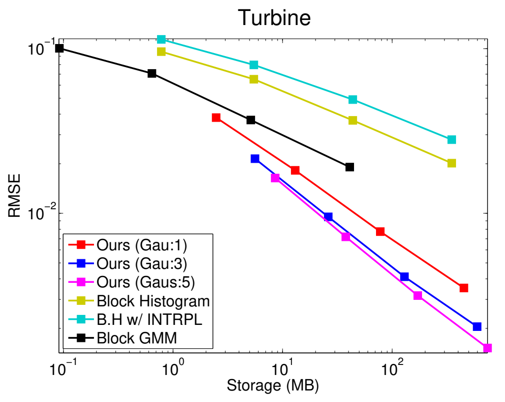
(a) Isabel RMSE



(b) Combustion RMSE



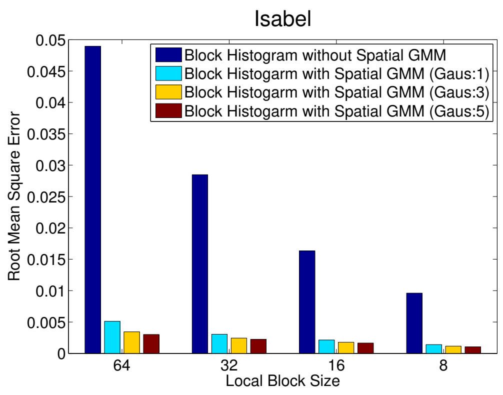
(c) Plume RMSE



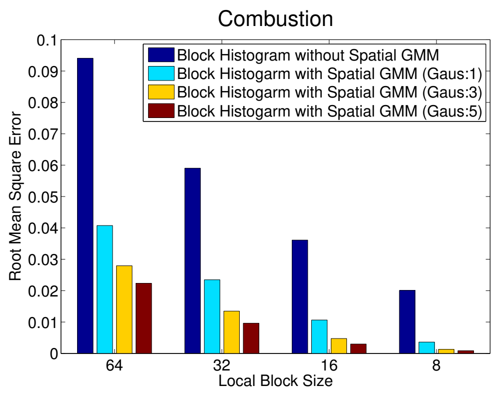
(d) Turbine RMSE

Figure 4.6: The trade off between accuracy (RMSE) and storage cost when comparing our spatial GMM approach using different block sizes and current approaches. Block sizes from left to right for (a),(b) and (c) are 64^3, 32^3, 16^3 and 8^3. Block sizes from left to right for (d) are 128^3, 64^3, 32^3 and 16^3. The bottom three curves show this trade off using our approach with upper bounds of 1, 3 and 5 on the number of Gaussian components.

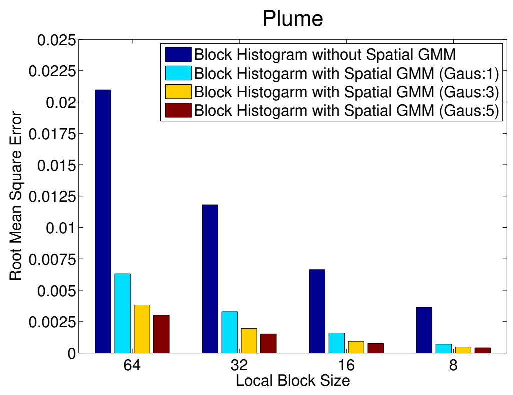
since the same block histograms were used in both cases. Given the same block size, block GMMs use less storage because the block histogram is a more compact representation. Even though our representation requires additional storage cost to incorporate spatial information, it significantly improves the RMSE. The bottom three curves show the estimation accuracy and storage cost of our Spatial GMM using up to 1, 3, and 5 Gaussian components for the spatial distributions. The spatial GMMs achieve lower RMSE when compared with all the other representations using just 1 Gaussian component, as illustrated by the red curve. In addition, given the same storage cost as the other approaches, our representation has smaller RMSE with different numbers of Gaussian components set as the upper bound. This trend can be observed consistently in all test data sets, which indicates that our approach gives better trade off between the quality and storage cost.



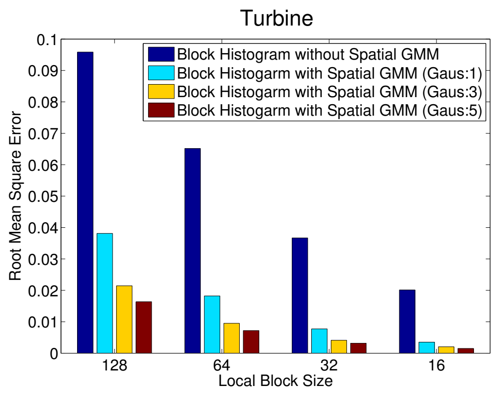
(a)



(b)



(c)



(d)

Figure 4.7: Comparison of RMSE when block histogram with our spatial GMM and without our spatial GMM are used under different block sizes.

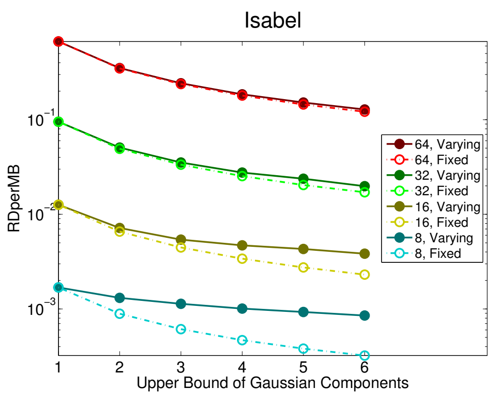
We also performed experiments to study the benefit of using the spatial distribution for different block sizes. We compared the RMSE of using only a block histogram compared with using a block histogram with our Spatial GMM, as shown in Figure 4.7. It is clear that by combining our Spatial GMM with the block histogram, we achieve a smaller RMSE when compared to only using a block histogram. Another observation is that when the block size is larger, the difference in RMSE with without our Spatial GMM is larger. The reason being that when a larger block size is used, more spatial information is lost, and thus we can benefit more from the spatial information found in the Spatial GMM.

4.4.2 Varying vs. Fixed Number of Gaussian Components

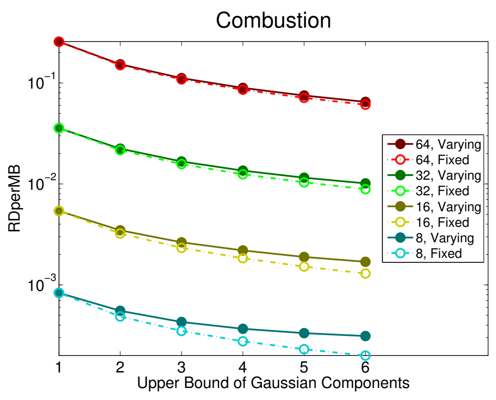
In this experiment, we show the benefit of varying the number of Gaussian components for the different spatial GMMs (see Section 4.2.3) rather than using the same or fixed number of Gaussian components for all of the Spatial GMMs. Using an upper bound on the number of Gaussian components (*ubg*) of 4 and compared with the fixed number of Gaussian components scheme, varying number of Gaussian components saves 29%, 28%, 16% and 13% Gaussian components on the Plume, Isabel, Combustion and Turbine datasets, respectively. We also compare these two schemes by varying different parameters such as block size and the upper bound on the number of Gaussian components. Both alternatives are compared with the $RDperMB(g)$ metric shown in Equation 4.10

$$RDperMB(g) = \frac{RMSE(0) - RMSE(g)}{Size(g) - Size(0)} \tag{4.10}$$

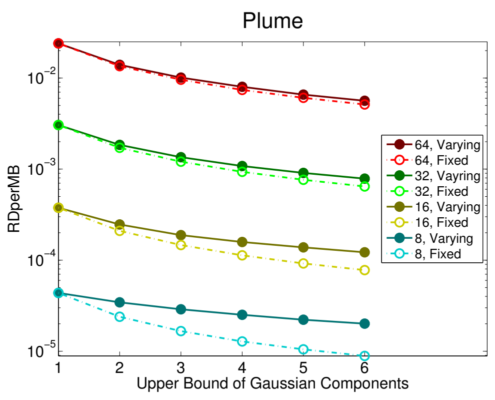
where $RDperMB(g)$ stands for ”RMSE Decrease per MBytes” and g represents the Gaussian components of a Spatial GMM, which is a fixed value when using a fixed number of Gaussian components and an upper bound when using a varying number of Gaussian components. $Size(g)$ is the storage cost using the Spatial GMM. The value $g = 0$ corresponds to using the block histogram only, i.e. no spatial information. Note that a higher $RDperMB(g)$ value corresponds to better storage utilization in order to further decrease RMSE.



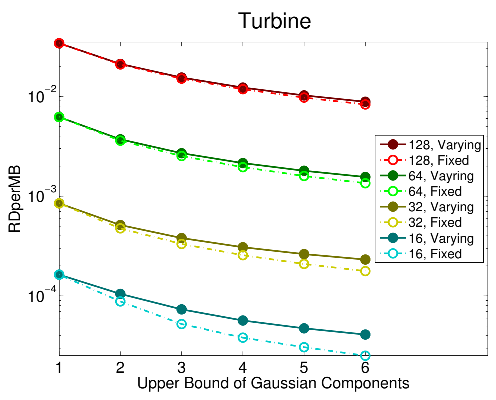
(a)



(b)



(c)



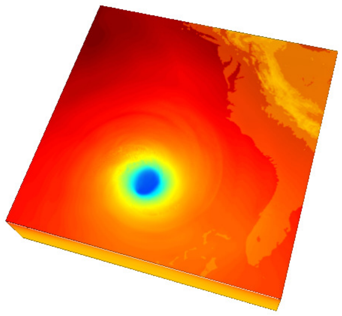
(d)

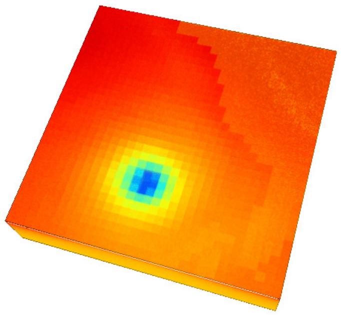
Figure 4.8: Comparison of varying and fixed number of Gaussian components schemes. The curves with same color use the same local block size. The solid and dotted lines indicate the representations generated from the varying and fixed number of Gaussian components schemes respectively.

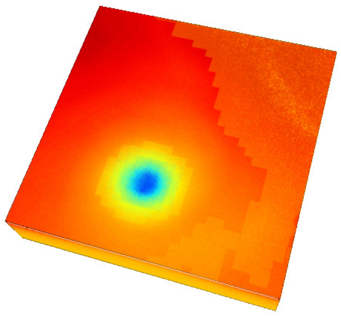
Figure 4.8 shows the decrease in RMSE per MBytes using both schemes. Four pairs of curves are shown for each dataset where the solid curves represent results from using a varying number of Gaussian components and the dotted curves represents the fixed number scheme. Each pair of curves, from the top to the bottom, was computed using the same block size. Varying the number of Gaussian components provides the same or larger decrease in $RDperMB(g)$, as shown by the the solid curves always being above the dotted curves. This is observed for all block sizes and is more pronounced for smaller block sizes. In addition, this is also observed when using a larger upper bound on the number of Gaussian components, indicated by the horizontal axis in the plots. In many cases, the spatial information in a smaller local block is more easily modeled so that using more Gaussian components may not be a good trade off. Varying the number of Gaussian components allows for fewer components and the use of more components on a need only basis.

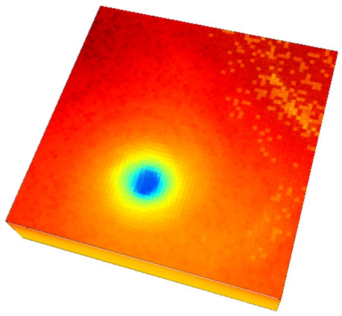
4.5 Visualization and Analysis

This section compares our approach to the block histogram and block GMM approaches [24, 56] on three data visualization and analysis applications. The first application is volume rendering of a reconstructed scalar field generated by sampling values from a distribution based representation. The second application is uncertain isosurface generation by computing the level crossing probability [7] at each voxel. The third application is local distribution-based feature matching. For each of these three applications, we first construct the appropriate field from a dataset using user provided parameters for the visualization task and a user defined region to construct the field and its resolution. Thus, scalar, probability, and similarity fields are constructed for the first, second, and third applications, respectively. Once constructed, we render the field with Paraview and perform viewpoint and transfer function exploration. We use the entire region of the dataset to construct each field. We use the same datasets used in our experiments in Section 4.4. We used the same spatial resolution as the original raw datasets to construct the fields for the Combustion, Isabel and Plume datasets. The Turbine dataset uses 1272x1272x220 as the spatial resolution. In this experiment, we set the local block size to 16^3 for the Combustion, Isabel and Plume datasets and 32^3 for Turbine dataset for our approach. We set the number of bins for the histograms to 128 and the upper bound of the number of Gaussian components for each Spatial GMM to 4. The datasets Plume, Isabel, Combustion and Turbines have averages of 51.03, 16.04, 10.36 and 31.38 Gaussian components per block, respectively. For the block histogram approaches, the histogram has 128 bins. We used 5 Gaussian components for each GMM in the block GMM approach. We ensured that all three representations, the block histograms with interpolation, the block histogram without interpolation, and block GMMs all had close storage size as our representation by adjusting the local block size.


(a) Rendering from raw data


(b) Block histogram (7.93MB)


(c) Block histogram w/ interpolation (7.93MB)


(d) Block GMM: (7.03MB)

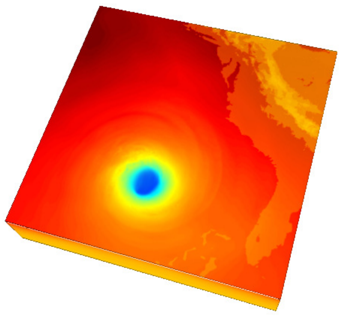
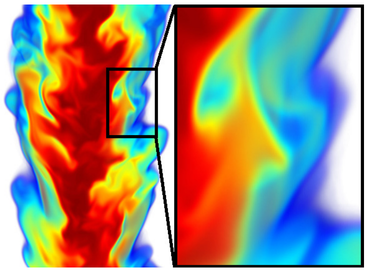

(e) Our approach (6.76MB)

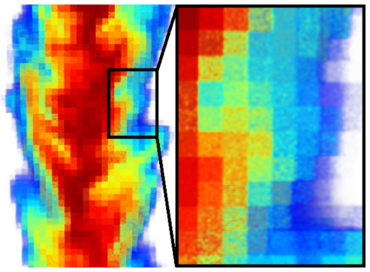
Figure 4.9: Visual comparison of volume rendering in Pressure variable of Isabel dataset. The samples are drawn from the PDFs, which are calculated at all grid points of the raw data, using Monte Carlo sampling. The block size of (b),(c),(d) and (e) are 12^3, 12^3, 6^3 and 16^3, respectively. In (e), the value histogram costs 3.5MB and the Spatial GMM costs 3.26MB.

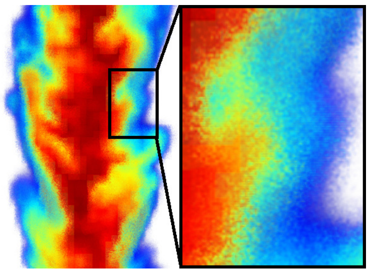
4.5.1 Volume Rendering

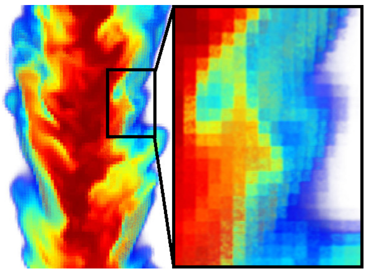
Our first experiment is to use each of the distribution-based representations for a direct volume render volume rendering task. We reconstruct the scalar field by drawing samples from the computed PDF of our representation, and from the histogram, interpolated histogram and GMM of other these approaches.

Figures 4.9 - 4.12 show renderings of each dataset (Isabel (Figure 4.9), Combustion (Figure 4.10), Plume (Figure 4.11) and Turbine (Figure 4.12)) using the four approaches.


(a) Rendering from raw data


(b) Block histogram (20.73MB)


(c) Block histogram w/ interpolation (20.73MB)


(d) Block GMM: (19.34MB)

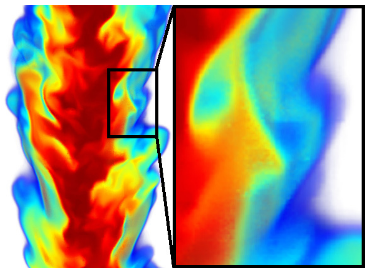
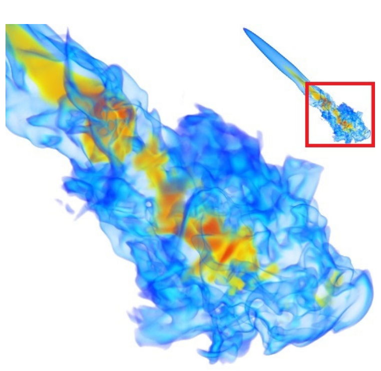
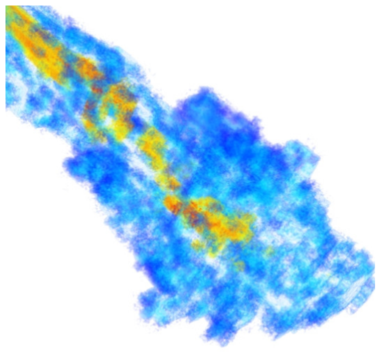
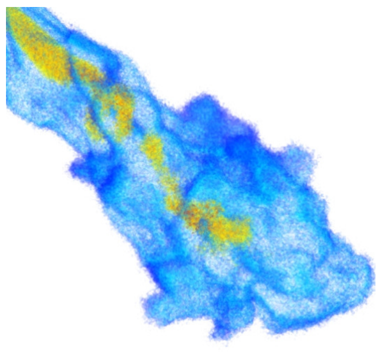

(e) Our approach (20.63MB)

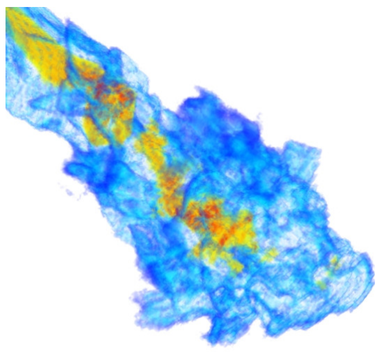
Figure 4.10: Visual comparison of volume rendering in combustion dataset. The samples are drawn from the PDFs, which are calculated at all grid points of the raw data, using Monte Carlo sampling. The block size of (b),(c),(d) and (e) are 10^3, 10^3, 5^3 and 16^3, respectively. In (e), the value histogram costs 5.27MB and the Spatial GMM costs 15.36MB.

Renderings computed with the block histogram without interpolations (shown in (b)) clearly shows blocky artifacts and lost detail. Renderings using the block histogram with interpolations (shown in (c)) alleviate the blocky artifacts, but details are blurred. Renderings computed with the block GMM (shown in (d)) show clear improvement from the block histogram only approaches. However, the approach also suffers from blocky artifacts and noise. Our approach (shown in (e)) is a marked improvement over the other approaches. Though our results do not exactly match the ground truth renderings (shown in (a)), it is much closer with a little noise, as illustrated in the zoomed in views.


(a) Rendering from raw data


(b) Block histogram (120.16MB)


(c) Block histogram w/ interpolation (120.16MB)


(d) Block GMM (141.39MB)

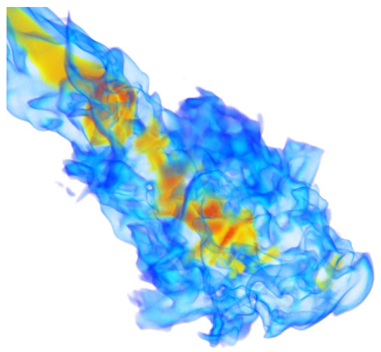
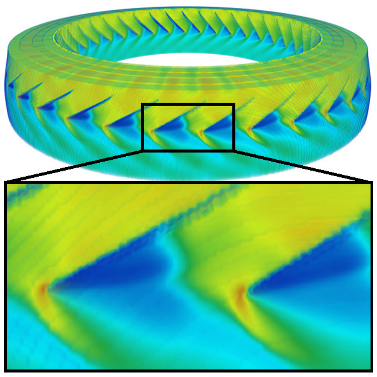
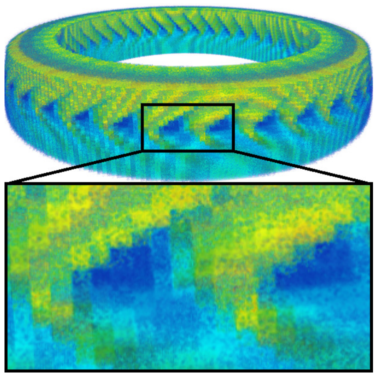

(e) Our approach (103.17MB)

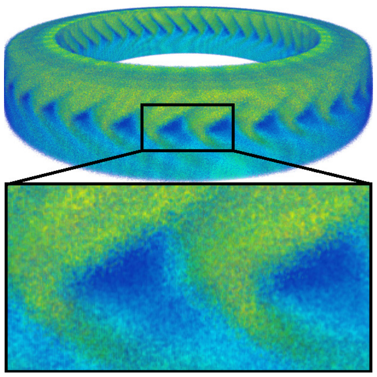
Figure 4.11: Visual comparison of volume rendering in Plume dataset. The thumbnail at right upper of (a) is the global view of Plume. We zoom in to the red box region in this visual comparison. We zoom in and render part of the dataset. The samples are drawn from the PDFs, which are calculated at all grid points of the raw data, using Monte Carlo sampling. The block size of (b),(c),(d) and (e) are 13^3, 13^3, 6^3 and 16^3, respectively. In (e), the value histogram costs 64MB and the Spatial GMM costs 39.17MB.

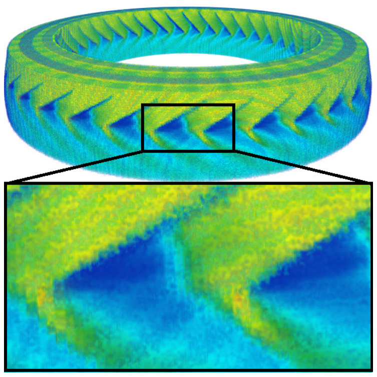
4.5.2 Uncertain Isosurface

Our next experiment is to show a visual comparison using uncertain isosurface rendering between the approaches using a statistical isosurface visualization technique with uncertain realizations [6, 7, 84]. We use the uncertain isosurface technique proposed by Athawale et


(a) Rendering from raw data


(b) Block histogram (131.4MB)


(c) Block histogram w/ interpolation (131.4MB)


(d) Block GMM (163.71MB)

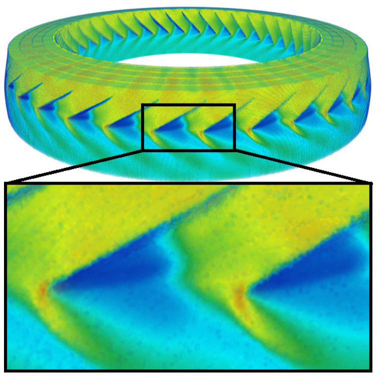
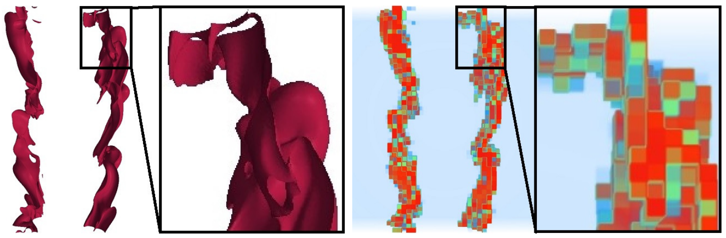

(e) Our approach (151.54MB)

Figure 4.12: Visual comparison of volume rendering in Turbine dataset. The samples are drawn from the PDFs, which are calculated at all grid points of the raw data, using Monte Carlo sampling. The block size of (b),(c),(d) and (e) are 22^3, 22^3, 10^3 and 32^3, respectively. In (e), the value histogram costs 43.75MB and the Spatial GMM costs 107.79MB.

al. [6] which computes the probabilities of the isovalue crossing in each cell and renders the probability field. We use Equation 4.11 to compute uncertain isosurface

$$P_{crossing}(c) = 1 - \prod_{k=0}^{7} \int_{-\infty}^{c} P_{\ell_k}(v)dv - \prod_{k=0}^{7} \int_{c}^{\infty} P_{\ell_k}(v)dv \tag{4.11}$$

where c is an isovalue and $P_{\ell_0}...P_{\ell_7}$ are the PDFs for the data values at the eight grid points on the cell.


(a) True isosurface (b) Block histogram (20.73MB)

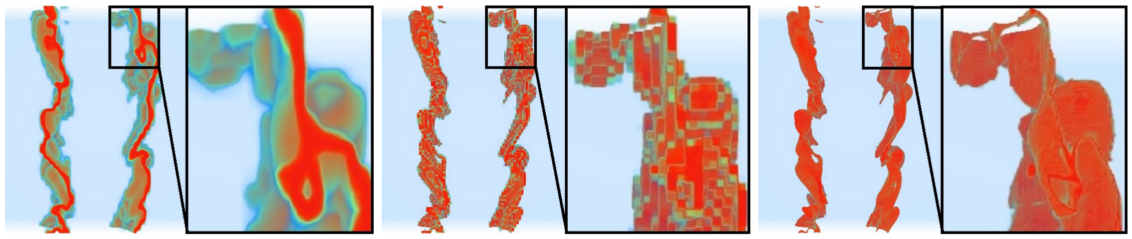
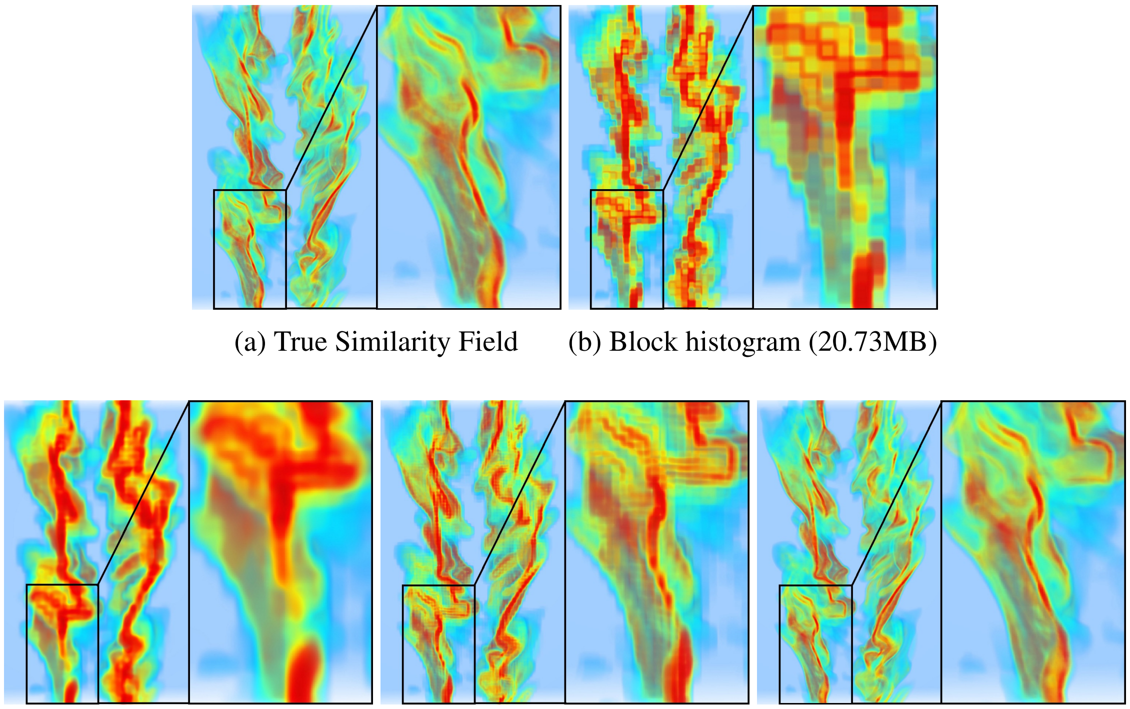

(c) Block histogram w/ interpolation (20.73MB) (d) Block GMM (19.34MB) (e) Our approach (20.63MB)

Figure 4.13: Uncertain isosurface (isovalue = 0.15) visualization of the Combustion dataset. (a) is the true isosurface from the raw data. The sizes of the distribution-based representations were similar in the results shown in (b),(c),(d), and (e). The color orange indicates locations with higher possibility for the location of the isosurface. The color blue indicates locations with less possibility for the isosurface. The settings used here are the same as used for the results in Figure 4.10.

Figure 4.1 shows renderings using the four approaches on the Isabel dataset. Figure 4.1 (a) and (f) show isosurface renderings generated from the original raw data to represent the ground truth. As was seen in the volume rendering results, renderings computed using the block histogram (Figure 4.1 (b) and (g)) and the block GMM (shown in Figure 4.1 (d) and (i)) approaches clearly suffer from blocky artifacts. Interpolation applied to the block histogram (Figure 4.1 (c) and (h)) greatly alleviates these blocky patterns, but the result only shows a rough uncertain isosurface. Our approach is able to show the isosurfaces much more clearly and accurately (Figure 4.1 (e) and (j)). Rendering using the Combustion dataset are shown in Figure 4.13. Notice the two holes in the zoomed in region for the ground truth rendering in (a). This feature is completely or nearly absent in the renderings using the other approaches (Figure 4.13 (b) (c) and (d)) due to higher uncertainty from these representations. On the contrary, this feature is preserved using our approach (Figure 4.13) (e)).

4.5.3 Local Distribution-based Feature Matching



(a) True Similarity Field (b) Block histogram (20.73MB)

(c) Block histogram w/ interpolation (20.73MB) (d) Block GMM (19.34MB) (e) Our approach (20.63MB)

Figure 4.14: Images of the distribution similarity field. (a) is the similarity field from the raw data. (b)(c)(d) and (e) are the similarity field from other distribution representation with similar storage size. Red color indicates high similarity. Blue color indicates low similarity. The data representations here are the setting in Figure 4.10.

One application of local distribution-based features [16, 45, 95] is to identify locations where local histograms are similar to a target feature defined by the user [16, 95]. Given a target distribution and a neighborhood size, the local distribution at each voxel in the raw data domain is computed from its neighborhood, and the L1-norm distance measure is applied to compute the similarity between the local distribution and the target distribution. Figure 4.14 shows renderings of the similarity fields from the search results on the Combustion dataset. The neighborhood size is set to 5x5x5 and the target distribution is selected from a region with pure fuel mass in the mixture fraction variable of Combustion data. Figure 4.14 (a) shows a ground truth rendering of the search result applied to the raw data. Results when using the block histogram (figures 4.14 (b) and (c)) and the block GMM (Figure 4.14 (d)) are unsatisfactory due to areas that are overestimated in the region around the pure fuel mass as a result of using an improper PDF with large uncertainty to estimate a value at each voxel. As the zoomed in results show, our approach (Figure 4.14 (e)) is extremely close to the ground truth.

4.6 Discussion

Preprocessing Time: We discuss and report the preprocessing time per block of our approach for the four datasets used in our experiments. We measure the preprocessing time to compute our representation with the same settings reported in Section 4.5 and computed on the same machine reported in Section 4.4. In the preprocessing step, we subdivide the data into blocks, compute the block histogram and Spatial GMMs using our adaptive scheme described in Section 4.2.3. The program is written in *Python* 3.3 using the *mixture* package in the *scikit-learn* library [13] to train the Spatial GMM. Since preprocessing time is affected by data complexity, more complex data requires more time to run the EM algorithm in training the Spatial GMMs due to more iterations needed for the algorithm to converge. For example, our algorithm took the most time to pre-process each block on average on the Combustion dataset since it has the most complexity among all of our test datasets. Computation time for each test dataset was 0.97 seconds to pre-process each block on average on the Combustion dataset, 0.51 second on the Plume dataset, 0.55 second on the Isabel dataset, and 0.69 second on the Turbine dataset.

PDF Computation Time: The time to compute the PDF depends on the number of Gaussian components. Hence, the time complexity of computing a PDF at a given location is $O(G)$, where G is the average number of Gaussian components per block and bin. We implemented the PDF computation using the NVIDIA CUDA Thrust Library and used the same machine as was used for the experiments conducted in Section 4.4 to compute run times here. We report both the total and average computation times on each dataset. Also, we use the same parameter settings that were used in Section 4.5. Since the resolutions of reconstructed fields are different, the total number of computed PDFs are also different. The total time to compute the PDF for the Combustion dataset took 3.79 seconds ($9.15 * 10^{-8}$

second in average), 1.12 seconds ($4.48 * 10^{-8}$ second in average) for the Isabel dataset, 20.99 seconds ($3.91 * 10^{-8}$ second in average) for the Plume dataset, and 291.08 seconds ($8.17 * 10^{-7}$ second in average) for the Turbine dataset. Note that the times reported here only include the time to compute the PDF computation.

We also report the total time to construct the scalar field in Section 4.5, which includes PDF computation time and value sampling time from the distribution. The time measurements for Combustion, Isabel, Plume and Turbine are 5.19, 1.89, 32.81 and 351.67 seconds, respectively. After the field is constructed and the two stage visualization pipeline (see Section 4.5) is used, both viewpoint and transfer function exploration can be computed in real-time.

Influence of Parameters: Three important parameters are the number of bins, the upper bound of the Gaussian components, and block size. An obvious trade off is that by increasing the bin number, you will increase the size of the histogram and the number of Spatial GMMs needed, but improve precision in the value domain. The influence of the upper bound of Gaussian components of the Spatial GMM can be seen in our experiments. In Figure 4.7, the RMSE computed over the datasets is decreasing for all block sizes when the upper bound of Gaussian components of the Spatial GMM increases. The reason is that with more Gaussian components used in the Spatial GMM there can be a better approximation of the Spatial distribution. But the "RMSE Decreasing per MByte" keeps decreasing while the upper bound of Gaussian components increases, as shown in Figure 4.8. This means that the per unit storage cost for the Spatial GMM results in decreasing quality improvement when the upper bound of Gaussian components is increasing. As mentioned previously, the PDF computation time is proportional to the total number of Gaussian components. Thus, increasing the upper bound of Gaussian components helps to improve quality but is affected by diminishing returns and increases PDF computation time. Also, an infinitely small block size makes the spatial distribution useless because no spatial information is lost. In Figure 4.8, we observe that the curves showing larger block sizes are always on the top of the curves showing smaller block sizes. This means that the per unit storage cost for the Spatial GMM improves quality more. But, Figure 4.7 also shows that the smaller block size always has better RMSE. Hence, when a larger block size is used, we can gain more by using the Spatial GMMs, but the smaller block size can provide better overall RMSE.

Scalability: Since computation of the distributions is done in a separate pre-processing step and distributions are localized to individual blocks, our approach scales well and easily leverages parallel processing. Pre-processing tasks over a single block are independent of pre-processing tasks in other blocks and only requires the data within that local block. Thus, the memory requirement for pre-processing each block only depends on the local block size and does not increase with raw data size. The computation time for pre-processing is proportional to the number of data points within a local block and the number of blocks of the dataset. Although the time to process each Spatial GMM in a local block is difficult to estimate, as it depends on the parameters of EM algorithm and properties of the data, the computation time of a Spatial GMM increases in general with more data points in a local block, and vice versa. The pre-processing time over the entire dataset grows just linearly with the number of blocks. Computing the pre-processing is suitable on multi-node supercomputers to bound the overall pre-process time for large datasets. Since no communication is needed between processes, no specially designed parallel algorithm is needed to apply our algorithm for pre-processing to multi-node supercomputers.

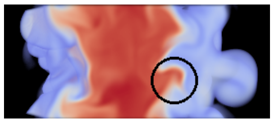
4.7 Conclusion

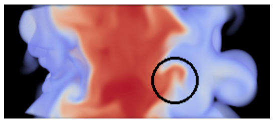
This paper presented a novel distribution-based representation for large-scale data sets, which allows high-quality statistical based analyses and visualization. Our proposed Spatial GMM representation compactly stores spatial information, which is missing in current distribution-based representations. In order to keep the storage overhead small, an adaptive scheme is used to determine the number of Gaussian components needed for each Spatial GMM. We qualitatively compared our representation with existing distribution representations. Our approach is able to compute the probability density function of values at any location, which represents the possible values and its associated occurrence probabilities, with less bias and variance at each voxel and provides superior visual results in the three visual analysis applications in our experiments.

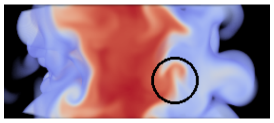
Chapter 5: Ray-based Exploration of Large Time-Varying Volume Data Using Proxy Per-Ray Distributions

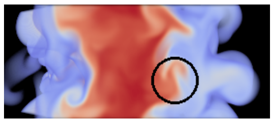
The advance of large scale supercomputers enables scientists to model complex physical phenomena with high-resolution simulations which have millions of spatial grid points and hundreds or even thousands of time steps. Scientists can understand the modeled phenomena in great detail by observing and analyzing the evolution of features in the simulation output over time. When the size of simulation output is small, common practice is simply to move the data to machines that perform post analysis. However, as the size of data grows, the limited bandwidth and capacity of networking and storage devices that connect the supercomputers to the analysis machine become a major bottleneck of the data analysis [2, 24, 94]. Furthermore, most post analysis and visualization machines do not have sufficient storage space to hold the entire simulation output. Rather they can handle only a small portion of the original dataset.

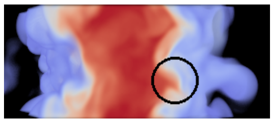
One common way to transfer large time-varying data sets to the analysis machine is to reduce the sampling rate in the temporal domain by storing volume datasets at every tenth or hundredth time step [17, 60, 105]. The drawback is that the scientist can easily lose track of the evolution of features between the two sampled time steps. Feature evolution between sampled time steps can be inferred by domain knowledge and approximated by linear interpolation or stored by compact data proxies. Though, smaller temporal sampling

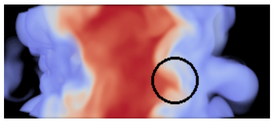

(a) True Volume (time step 45)

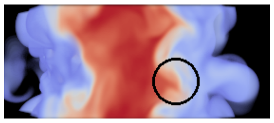

(b) True Volume (time step 46)

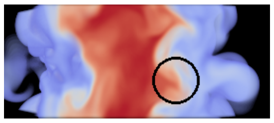

(c) True Volume (time step 47)

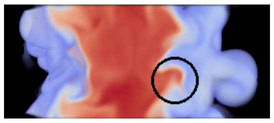

(d) True Volume (time step 48)

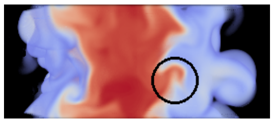

(e) Interpolation (time step 45)

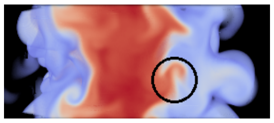

(f) Interpolation (time step 46)


(g) Interpolation (time step 47)


(h) Interpolation (time step 48)


(i) Our approach (time step 45)


(j) Our approach (time step 46)


(k) Our approach (time step 47)

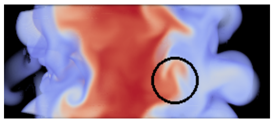

(l) Our approach (time step 48)

Figure 5.1: Our approach mixes naive interpolation approximation, ray histogram and depth information to produce the animation of time-varying data sets. We illustrate our approach with the mixture fraction field of Combustion time varying dataset. We sampled this dataset every ten time steps in temporal domain. In this figure, we have the sampling time steps at 40 and 50. The time steps at 45, 46, 47 and 48 are shown. The images in the first row are the ground truth rendered from the raw volumes of these four time steps. The images in the second row are rendered based on the interpolation approach which interpolates data from time steps 40 and 50. The images in third row are rendered by our approach. In the black circle, our approach shows the feature evolution of the red material, which is very close to the ground truth. However, we cannot observe similar feature evolution in the second row. This may mislead the scientists' analysis.

rates may not be sufficient for the use of domain knowledge to correctly predict features between sampled time steps. Interpolation approximation is prone to large errors leading to incorrect analysis, which is illustrated in Figure 5.1. The pure fuel region (red material) is clearly apparent as it moves over time in the ground truth rendering from the raw data, but is barely seen in the renderings in the second row. Another way to visualize feature evolution at non-sampled time steps is to create compact data proxies to replace the non-sampled time step volumes. These include compression [39, 48, 55] and spatial subsampling-based techniques [24, 80, 93]. The drawback is that these approaches do not take into account image space properties and ignore information from the sampled time steps. Thus, it is difficult to arrive at a good trade-off between storage and image quality. The image-based approach [2,85,87] is promising in that salient views are selected and data proxies are created based on the image space at the skipped time steps to compensate for lost information. The approach has demonstrated successful results in many analysis applications to overcome the big data challenge. Though, their potential shortcomings either have limited ability for transfer function exploration or imprecise assumptions regarding depth information. This may result in difficulties in identifying the salient features. In addition, existing image-based approaches do not incorporate information from sampled time steps to achieve a good trade-off between storage and image quality. Therefore, it is still an open problem to develop techniques to explore large scale time-varying data in post analysis machines, while not losing any critical information between the sampled time steps.

In this paper, we present a ray-based approach for time-varying volume analysis. The goal is to better preserve temporal data fidelity between sampled time steps with a small storage cost. With our algorithm, scientists can select salient views to generate compact ray-based proxies with the ability to allow for arbitrary transfer function modification to compensate for the lack of information from non-sampled time steps. To produce the compact ray-based data representation, we decouple a pixel ray into the samples' *value distribution* and *bin-wise depth informations*, and store them separately. The samples' distribution along a ray from the raw data at non-sampled time steps is stored as a histogram, which captures the statistical characteristics of the samples. The bin-wise depth informations are the location information of data samples of each histogram bin. It provides the clues to place the samples of a ray histogram along the ray. By decoupling the rays into histograms and bin-wise depth informations, the histograms and bin-wise depth informations can be encoded to save on storage cost. Ray histograms and bin-wise depth informations are encoded into codebooks, where similar ray histograms and bin-wise depth informations are identified and represented by a single ray histogram and a bin-wise depth information, respectively. In addition, we use data coherence in the temporal domain to further define a more compact representation. The data along a pixel ray at a non-sampled time step and a corresponding pixel ray at a sampled time step are often similar due to trends in how data values change in the time sequence. One simple example is that sample values on both pixel rays are monotonically increasing even if they have different corresponding sample values along the rays. In this example, the depth information along the pixel ray at the sampled time step can be used instead of the bin-wise depth information of the pixel ray at the non-sampled time step, which saves on the cost to store bin-wise depth informations in this case. Also, some pixel rays could pass regions where the data values are not changing, changing very little, changing smoothly over time. In this case, naive interpolation of values from the sampled time steps is sufficient to reconstruct a pixel ray at a non-sampled time step and we can completely avoid storing a ray histogram and bin-wise depth informations. When constructing the data proxy in a preprocessing step, our approach analyzes temporal coherence in the time-varying data to choose the most appropriate representation for scalar data along pixel rays at non-sampled time steps. Our view-dependent proxy allows for fast rendering of the time-varying data from view samples used to create the proxy as well as rendering from nearby view angles to improve scientists understanding of evolving features over time.

5.1 Overview

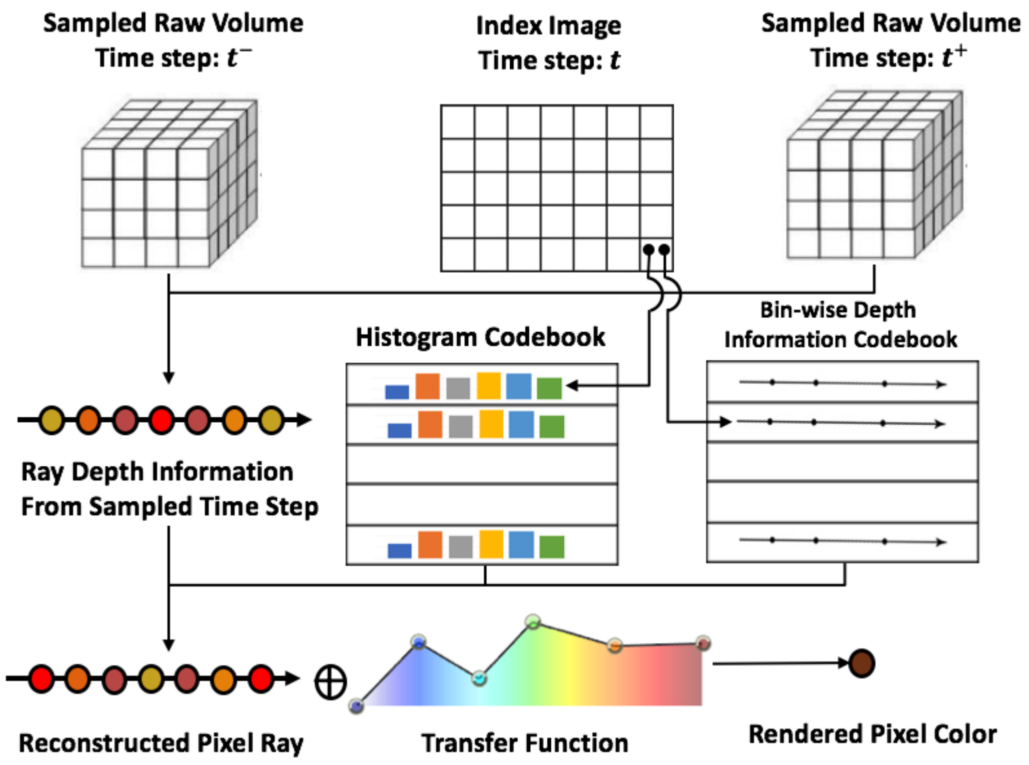


Figure 5.2: The proxy data structure stores sampled volume time steps and a collection of index images at each non-sampled time step, which are located between two neighboring sampled time steps (t^- and t^+). The samples defined on a pixel ray are represented by a ray histogram and bin-wise depth informations except when temporal coherence is detected and corresponding pixel rays in neighboring sampled time steps provide information to describe the samples on the pixel ray. Samples along index image pixel rays are quickly reconstructed for fast volume rendering in the context of transfer function changes. Ray histogram and bin-wise depth informations are stored in codebooks.

Our proxy data structure stores only a subset of the volumes in the time series of a time-varying volume dataset, called sampled time steps. Each volume in the rest of the time series, called a non-sampled time step, is replaced by a collection of index images. The scalar values of the samples along a pixel ray defined at an index image are summarized with a histogram, called a ray histogram. The location information of these samples is similarly summarized as bin-wise depth information. We call this compact pixel ray representation the decoupled ray representation. The ray histograms and bin-wise information across pixel rays are stored in codebooks for further storage savings. Though, temporal coherence in the time-varying volume data allows for two alternative pixel ray representations that provide a better tradeoff between storage cost and sample reconstruction quality. Our proxy data structure is illustrated in Figure 5.2. Each pixel ray at an index image is represented by one of three possible ray representations: the interpolation ray, depth profile ray, or decoupled ray. The interpolation ray representation (Section 5.4) incurs the least storage cost among the three options because it approximates the scalar function along a pixel ray using scalar information from rays at the same image coordinate on the neighboring two sampled time steps. The depth profile ray representation (Section 5.3) combines a histogram constructed from data along the pixel ray and depth profile information found at the neighboring sampled time steps. The storage cost is a ray histogram and a pointer. The decoupled ray representation (Section 5.2) is the most expensive among the three representations because bin-wise depth information incurs a larger storage cost than depth profile information. During data proxy construction, one of these three ray representations is chosen for each pixel ray such that the chosen representation provides the least storage cost and quality that is greater than a user-defined threshold, T_r. This quality metric, we call Q_r, is defined to be between 0 and 1 where higher values denote better reconstruction quality, as shown in Equation 5.1.

$$Q_r = 1 - \left(\sqrt{\frac{\sum_{i=0}^{L-1}(r_{gt}(i) - r_{re}(i))^2}{L}} / V_{range}\right) \tag{5.1}$$

The data value range is V_{range} and L is the number of samples on the ray. The sample values at the i^{th} sample along the ray are $r_{re}(i)$ and $r_{gt}(i)$ from the reconstructed time step and the raw volume, respectively. The quality of the approximation of the samples along the pixel ray is measured using Root Mean Square Error (RMSE), $\sqrt{\frac{\sum_{i=0}^{L-1}(r_{gt}(i)-r_{re}(i))^2}{L}}$. The choice of which ray representation to use for the pixel ray is as follows. The interpolation ray representation is selected if the quality, Q_r, is greater than T_r. If not, then the depth profile ray representation is selected if its quality is greater than T_r. Otherwise, the pixel ray is represented with the decoupled ray representation. In this case, if the number of rays that do not satisfy the quality threshold T_r is greater than $e\%$ of the number of rays at the non-sampled time step, then $e\%$ of the rays with the lowest reconstruction quality are selected to compute the ray histogram. All of the ray histograms in the data proxy are encoded into a codebook by removing "redundant" ray histograms that are too similar to each other. Bin-wise depth information is encoded similarly to gain further storage savings. The codebooks are described in Section 5.5.

5.2 Ray-based Proxy Representation

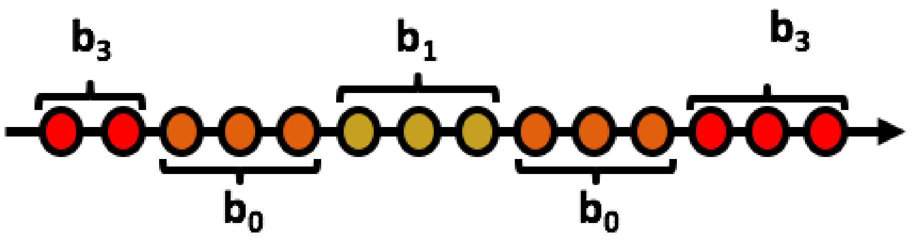
Each non-sampled volume time step is replaced with a proxy consisting of index images defined at pre-chosen view samples. At each pixel ray of an index image, a ray is cast into the non-sampled volume and sampled in depth order. The samples are encoded in a statistical summary, called a *ray histogram*, and their locations are preserved in a *bin-wise depth* data structure. In this section, we describe how to compute these two data structures, which together is called the *decoupled ray representation* because sample values are stored separately from their locations. Section 5.5 describes how the ray histograms and bin-wise depth information are stored into codebooks for further savings. Sections 5.3 and 5.4 present two alternative pixel ray representations to the *decoupled ray representation* that are more compact by leveraging temporal coherence.

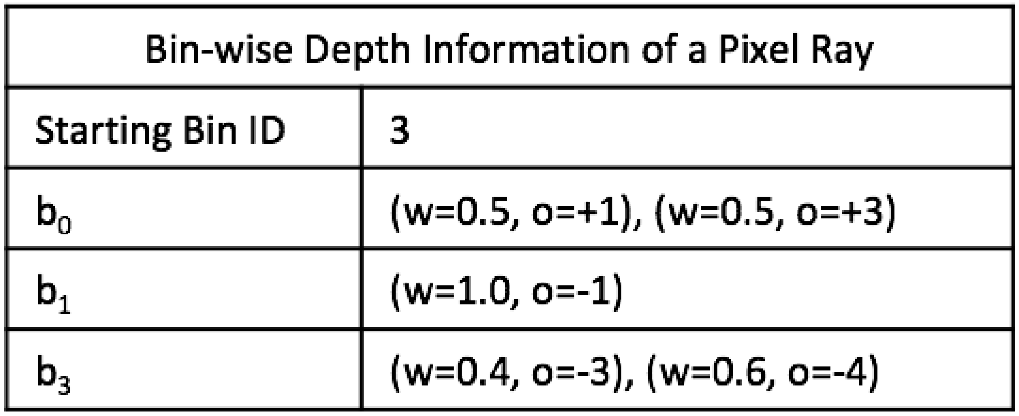
5.2.1 Ray Histogram

The *decoupled ray representation* summarizes the sampled data values along the pixel ray in a histogram consisting of B bins, where each bin represents a value interval in the data range. The value intervals may be defined to equally subdivide the data range or defined by scientists to represent value ranges based on material properties. If b_i denotes the i^{th} bin in the histogram, the value interval of bin b_i represents the data value range $[Lower_{b_i}, Upper_{b_i}]$. The probability mass for the i^{th} bin is M/N, where M is the number of sample values within the data interval $[Lower_{b_i}, Upper_{b_i}]$ and N is the total number of samples along the pixel ray.

5.2.2 Bin-wise Depth Information

The ray histogram lacks location information for the pixel ray samples, which is needed for quality rendering. We compactly represent this information by identifying *runs* of consecutive samples along the ray in depth order such that all samples in the same run map to the same ray histogram bin. This is illustrated in Figure 5.3 where depth information of the samples along a ray is modeled as a list of five runs. The first run consists of two samples whose data values map to bin b_3, the second run consists of three samples whose data values map to bin b_0, etc. Samples from different runs may map to the same bin, such





Bin-wise Depth Information of a Pixel Ray	
Starting Bin ID	3
b_0	(w=0.5, o=+1), (w=0.5, o=+3)
b_1	(w=1.0, o=-1)
b_3	(w=0.4, o=-3), (w=0.6, o=-4)

Figure 5.3: To preserve location information, samples are collected along a pixel ray bin-wise into runs in depth order. In the example above, each circle represents a sample and samples with the same color map to the same histogram bin, i.e. material. A sample run is a group of consecutive samples that map to the same bin. These runs are stored in a table indexed by bin ID and connected via a linked list in depth order. Each run stores the fraction of samples found in the run over all runs in the bin, w, and a link, o, that refers to the bin containing the next run in the linked list.

as the first and last runs denoted by b_3 in the figure. We define a table to hold all the runs on the pixel ray such that each run is stored with its bin, also shown in Figure 5.3. The table is indexed by bin IDs where each table entry stores a depth ordered list of runs that map to the bin for those bins with non-zero probability. We store with each run the percentage of the sample count in the run with respect to the sample count of all runs that map to the same bin to use as a weighting factor. Equation 5.2 defines this value for the c^{th} run in the list of runs stored at bin ID b. The value $SmpCount_b(i)$ is the sample count of the i^{th} run in the list at bin ID b and N is the length of the bin's list. In the example in Figure 5.3, bin b_3 has two runs along the ray with weights 0.4 and 0.6, respectively.

$$w_b(c) = \frac{SmpCount_b(c)}{\sum_{i=0}^{N-1} SmpCount_b(i)} \tag{5.2}$$

To preserve the ordering of the runs along the pixel ray in the table, we encode the list of runs as a linked list by storing with each run a link value that is a relative bin ID to the next run. This link value is defined in Equation 5.3 for the c^{th} run at bin ID b, where $NextBinID(c)$ is the bin ID of the run after run c along the pixel ray and $BinID(c)$ is the bin ID of run c. In Figure 5.3, the link value for the first b_3 run is 0 - 3 = -3. We use -1 to represent the next bin ID after the last run. For example, the link value of the second b_3 run is -1 - (3) = -4. The table also stores the bin ID of the first run, called the starting bin ID. Thus, the runs may be visited in depth order, where in our example the first pixel ray run is found in the first run at bin ID 3, then the next run is found as the first run at bin ID = 3 + (-3) = 0, then the run at bin ID = 0 + 1 = 1, then the second run at bin ID = 1 + (-1) = 0, and finally the second run at bin ID = 0 + 3 = 3.

$$o_b(c) = NextBinID(c) - BinID(c) \tag{5.3}$$

5.2.3 Ray Reconstruction

A pixel color resulted from volume rendering can be computed from the ray histogram and bin-wise depth information. The sample runs are processed in front-to-back order by traversing the linked list of runs. Color and opacity are computed for each run using the transfer function where these values are blended together for the final pixel value. As shown in Equation 5.4, the number of samples in the c^{th} run at bin b_i, $SmpCount_{b_i}(c)$, is computed from the weight, w, of the run, the probability mass value of the bin containing the run found in the ray histogram, H, and the expected number of samples along the entire ray, S.

$$SmpCount_{b_i}(c) = S * H(b_i) * w(c) \tag{5.4}$$

Algorithm 4 Ray reconstruction and rendering with the ray histogram and bin-wise depth information

```
▷ $rh$: ray histogram; $dptInfos$: bin-wise depth informations
▷ $stBinID$: starting Bin ID of the ray
▷ $S$: expected number of samples on the reconstructed ray
procedure RayReconstruction($rh$, $depInfors$, $stBinID$, $S$)
    $color$ = (0,0,0)
    $currBin$ = stBinID
    $currComps$ = [0,0,...,0]

    while $currBin \neq -1$ do
        $di$ = dptInfos($currBin$)
        $compSmps$ = $S$*$rh[currBin]$*$di.w[currComps[currBin]]$
        $rgba$ = tf($currBin$)
        $alpha$ = 1.0 - pow((1.0 - $rgba.a$), $compSmps$)
        $color$ += $clr.rgb$ * $alpha$
        $currBin$ += $di.o[currComps[currBin]]$
        $currComps[currBin]$ += 1
    end while

    return $color$
end procedure
```

The steps to compute the pixel color are shown in Algorithm 4. Line 7 initializes an array of indices with length equal to the number of bins in the histogram. Each entry records the number of runs that have been processed in the bin, which is used to index the next bin to process. Line 10 retrieves the next run to process. Next, the sample count of this run is computed in line 11. Accumulated color, denoted by variable $color$, for the pixel is blended in lines 13 - 14 after color and opacity is determined for the run in line 12 using the transfer function. The location of the next run to process, *currBin*, is computed in lines 15 - 16 using the link o. The variable $currComps[currBin]$ is updated to refer to the next run in the bin.

5.3 Depth Profile Ray Representation

As mentioned in the previous section, a ray histogram lacks location information of the samples, which is needed for rendering. We leverage temporal coherence that may exist in time-varying datasets and present a cheaper alternative to approximate location information than using bin-wise depth information described in Section 5.2.2. A *depth profile* of the scalar function defined on a pixel ray represents how the scalar values change along a pixel ray in depth order. We observe that many pixel rays may share the same *depth profile* due to temporal coherence. For example, this is seen in the Isabel dataset (Figure 5.8) where scalar values along many pixel rays are monotonically increasing due to lower altitudes having higher pressure. The *depth profile ray* representation combines the ray histogram of the samples along a pixel ray in the non-sampled time step and the depth profile of a pixel ray in the nearest sampled time step, where the latter requires storing only a reference to the identified pixel ray. Thus, the *depth profile ray* representation requires less storage than decoupled ray representation.

5.3.1 Detection and Search

To determine whether a pixel ray r_p at image coordinate p at a non-sampled time step should be represented in the proxy with a depth profile ray, the pixel ray r_p is cast into the non-sampled volume and sampled at L locations along the ray. This pixel ray represents the ground truth and is used to evaluate a candidate depth profile ray. Next, a small pixel region is defined in the nearest sampled time step around image coordinate p. A ray is cast at each pixel in this region where samples are reconstructed from the volume at L locations. Each sample is a two-tuple (*value*, *location*) holding a scalar value and its location along the pixel ray. A search for the best depth profile among these pixel rays is done to find a candidate depth profile to use for representing pixel ray r_p. Before the search, a list of the L scalar values of the samples along pixel ray r_p is constructed by sorting these scalar values and ignoring their locations. For each candidate pixel ray in the search region, its L (*value*, *location*) tuples are sorted by their scalar values. A pixel ray with L approximated samples is constructed by taking the sorted list of scalar values from pixel ray r_p and creating a corresponding list of (*value*, *location*) tuples, where the first component of each tuple holds a scalar value from the sorted list of scalar values. The location value for the second component of each two-tuple is copied from the location value of a two-tuple in the candidate pixel ray sample list whose scalar value's rank in its sort is the same rank as the scalar value in this two-tuple. Thus, the approximated samples in this pixel ray hold the scalar values from pixel ray r_p using the depth profile defined in the candidate ray. The metric Q_r is computed using Equation 5.1, where $r_{gt}(i)$ is the i^{th} sample value along the ground truth pixel ray p_r and $r_{re}(i)$ is the corresponding sample value on the pixel ray holding the approximated samples. If the value of Q_r is above the user-defined threshold, T_r, then pixel ray r_p can be represented with a depth profile ray. In the proxy, storage for the pixel ray will be a ray histogram and a reference to the candidate pixel ray found in the search.

5.3.2 Ray Reconstruction

If a pixel ray at image coordinate p in the proxy is represented with a depth profile ray, we describe how to reconstruct the ray to be used to calculate pixel color. First, a ray is cast at the pixel found in the search region around image coordinate p in the nearest sampled time step and S samples are reconstructed from the volume, where each sample is a (*value*, *location*) tuple, and then the samples are sorted by their scalar values. Next, S scalar values are drawn from the ray histogram and placed into a sorted list. A list of (*value*, *location*) tuples is created from this sorted list of scalar values where each scalar value is stored in the first component of a tuple. The location of a tuple is copied from the location in a tuple from the ray cast into the nearest sampled time step whose scalar value's rank is the same as the rank of this tuple's scalar value. This list of tuples is sorted according to location and the pixel ray is rendered in front-to-back order.

Algorithm 5 Depth profile ray reconstruction

```
 1: ▷ $dr$: depth information ray; $rh$: ray histogram;
 2: ▷ $S$: number of samples on reconstructed ray
 3: procedure RayReconstruction($dr$, $rh$, $S$)
 4:     $reconRay$ = createRayBuffer($S$)
 5:
 6:     $drt$ = buildTuples($dr$, ($value$,$location$))
 7:     $smps$ = takeSamples($rh$, $S$)
 8:     $drt$ = sortTuples($drt$, $value$)
 9:     $smps$ = sort($smps$)
10:
11:     for $i := 0$ to $(S-1)$ do
12:         $reconRay[drt[i].location] = smps[i]$
13:     end for
14:
15:     return $reconRay$
16: end procedure
```

Algorithm 5 illustrates these steps in pseudo code. Line 4 allocates *S* spaces for the ray that will hold the reconstructed sample values from the ray histogram and associated depth order. Line 6 retrieves the list of (*value*, *location*) tuples from the pixel ray at the nearest sampled time step that is referenced by the depth profile ray. Line 7 draws *S* samples from the ray histogram. Line 8 sorts the array of tuples, *drt*, by their samples values. Line 9 sorts the list of sample values drawn from the ray histogram. The steps in Lines 11-13 associate a location from the depth profile *drt* with a scalar value from *smps* such that scalar values from both sorted lists have the same rank.

5.4 Interpolation Ray Representation

Time-varying datasets possess temporal coherence in spatial regions where scalar values either do not change or change smoothly over time. In this case, simply interpolating corresponding sample values from the two nearest sampled time steps will suffice to approximate sample values along a pixel ray at the non-sampled time step t from the data proxy. To determine whether the pixel ray at pixel p cast into the non-sampled time step t will be represented with the interpolation ray representation, we first cast a pixel ray from pixel p into the non-sampled volume at time step t and define L samples. The samples along this pixel ray represent the ground truth. Next, two pixel rays are cast from the same pixel location p at the two nearest neighbor sampled time steps, V_{t^-} and V_{t^+}, and L samples corresponding to the ground truth samples are collected along both of these rays. The corresponding sample pair at location ℓ along the pixel rays at the sampled time steps are linearly interpolated over time to time step t using Equation 5.5

$$R_t(\ell) = R_{t^-}(\ell) * (1-a) + R_{t^+}(\ell) * a \tag{5.5}$$

where $R_t(\ell)$ is the sample value at location ℓ along the pixel ray at time step t where $a = (t-t^-)/(t^+ - t^-)$. The quality of the interpolation over all samples along the pixel ray, Q_r, is computed using the Root Mean Square Error (RMSE) metric as shown in Equation 5.1. If the value of Q_r is above a user-defined threshold, T_r, then the data proxy will simply store a single flag at pixel p of time step t to indicate that linear interpolation (equation 5.5) should be used to reconstruct the pixel ray samples during rendering.

5.5 Ray Histogram and Bin-wise Depth Information Codebooks

The ray histogram and bin-wise depth data structures presented in Section 5.2 may be compressed into codebooks for further storage savings in the proxy. A codebook for the ray histograms will replace similar histograms with a single representative ray histogram. Equivalently, a codebook for bin-wise depth information will store depth information by identifying similarity as well. After the codebooks are constructed, representative ray histograms and bin-wise depth information are found in the codebooks using a bitmap index data structure for efficient search. The following sections describe how to construct the codebooks and then search for representatives.

5.5.1 Ray Histogram Codebook

Ray histograms are defined in the decoupled ray and depth profile ray representations. The proxy can encode these histograms into a codebook where each ray histogram is either stored in the codebook or indexes a representative histogram. We observe that similar ray histograms are often found within a neighborhood of pixel rays and across the temporal domain. Groups of similar ray histograms can be replaced with a single representative ray histogram where it is expected that sizable groups would be identified since the ray histogram is a statistical summary of scalar values lacking the constraint of associated depth information.

Ray Histogram Similarity

To group similar ray histograms, we define a similarity metric, $HistD(H_i,H_j)$, that indicates the difference between two ray histograms H_i and H_j. Two ray histograms are not similar if corresponding bins with non-zero probability do not match. This avoids introducing non-existent scalar values that may map to non-existent colors from the transfer function during rendering. In this case, $HistD(H_i, H_j)$ is assigned to infinity. Two ray histograms are similar when the shapes of their distributions are close, which is measured by the L_1-*norm* distance metric as shown in Equation 5.6

$$L_1(H_i, H_j) = \sum_{k=0}^{B-1} \left| H_i(b_k) - H_j(b_k) \right| \tag{5.6}$$

where $H_i(b_k)$ and $H_j(b_k)$ are the probability mass at bin b_k of both histograms. The similarity metric $HistD(H_i, H_j)$ is shown in Equation 5.7

$$HistD(H_i, H_j) = \begin{cases} L_1(H_i, H_j) & if\ nz(H_i) = nz(H_j) \\ \infty & Otherwise \end{cases} \tag{5.7}$$

where $nz(H_i)$ and $nz(H_j)$ are the sets of bins with non-zero probability in both histograms.

Codebook and Bitmap Indexing Table Construction

The ray histogram codebook is constructed incrementally by encoding each ray histogram into the codebook one at a time, as illustrated in Figure 5.4. To encode a ray histogram H_n, a ray histogram H_e is found in the codebook that minimizes $HistD(H_e, H_n)$. If the similarity $HistD(H_e, H_n)$ is less than a pre-defined threshold T_h, then H_n will index H_e. Otherwise histogram H_n is inserted into the codebook. Instead of using a linear search to find H_e that compares H_n with every ray histogram in the current codebook, we reduce the search space using bitmap indexing, which is an efficient indexing data structure that is supported by fast bit-wise operators implemented in computer hardware. It is used widely in visualization applications [82, 95] for fast value range queries.

Bitmap Indexing Table: Equation 5.7 illustrates that in order for two histograms to be similar both must have the exact same bins with zero probability and a small enough

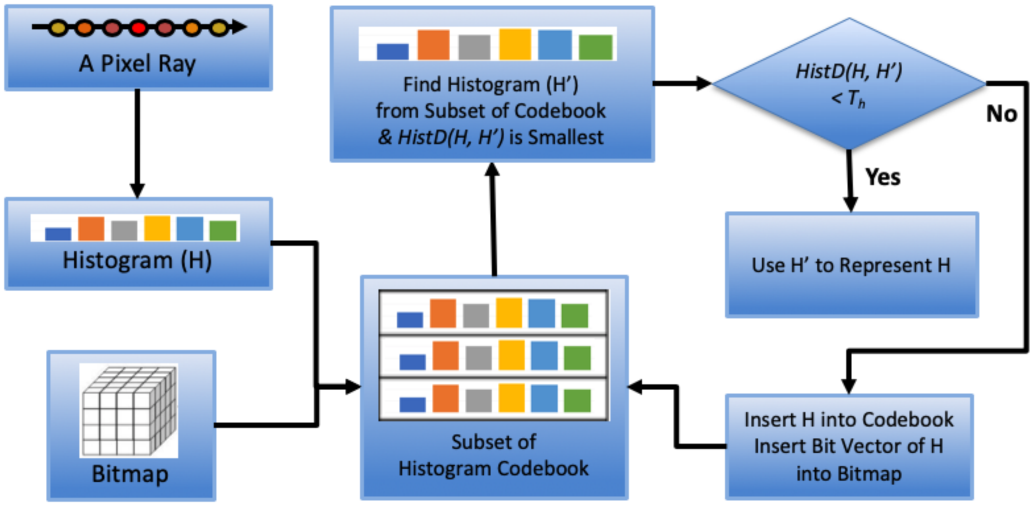


Figure 5.4: The ray histogram codebook is constructed by individually inserting each ray histogram. If a ray histogram is similar using a pre-defined threshold to a histogram already contained in the codebook, then it will index the representative histogram. Otherwise the ray histogram is inserted into the codebook.

difference in the probabilities of corresponding non-zero bins. Given a ray histogram H_n, the bitmap indexing data structure can quickly identify similar histograms to H_n in a codebook that satisfy both of these criteria using bit-wise operations. Consider a simple example where each ray histogram has only a single bin. The table on the left in Table 5.1 shows a codebook containing six histograms with IDs 0 to 5 and probability mass indicated in the second column. A bitmap indexing table is shown on the right where table entries hold a binary value, i.e 0 or 1. Each row matches an ID in the codebook. The first column, p_0, encodes histograms with zero probability in their bins as a bit vector. Here, the bit vector (0, 0, 1, 0, 0) indicates that only the histogram with ID = 2 has zero probability in its bin. To determine the histograms with non-zero bin probabilities we just compute the negation of this vector, i.e. $\neg(0, 0, 1, 0, 0) = (1, 1, 0, 1, 1)$. The probability range (0, 1] is partitioned into five subranges (0, 0.2], (0.2, 0.4], (0.4, 0.6], (0.6, 0.8], and (0.8, 1.0],

Table 5.1: 2D bitmap indexing example

Ray Histogram Database		Bitmap Indexing					
ID	Bin Prob	p_0	p_1	p_2	p_3	p_4	p_5
		0	(0,0.2]	(0.2,0.4]	(0.4,0.6]	(0.6,0.8]	(0.8,1.0]
0	0.7	0	0	0	0	1	0
1	0.6	0	0	0	1	0	0
2	0	1	0	0	0	0	0
3	0.2	0	1	0	0	0	0
4	0.5	0	0	0	1	0	0
5	0.9	0	0	0	0	0	1

which are defined in the columns p_1, p_2, p_3, p_4, and p_5 in the table. The bit vectors in these columns indicate the histograms that have a bin probability within the indicated subrange. For example, the ray histograms with IDs 1 and 4 have bin probabilities in the subrange (0.4, 0.6]. To query histograms having bin probabilities in a wider subrange, bit vectors in overlapping subranges can be combined using the bitwise *OR* operator. For example, histograms with bin probabilities in the subrange (.2, .8] are identified by first retrieving bit vectors in columns p_2, p_3, and p_4. The bit vector resulting in the calculation (0, 0, 0, 0, 0, 0) $\bigvee$ (0, 1, 0, 0, 1, 0) $\bigvee$ (1, 0, 0, 0, 0, 0) = (1, 1, 0, 0, 1, 0) indicates that ray histograms with IDs 0, 1, and 4 have bin probabilities in the subrange (.2, .8].

Table Creation: Since we require ray histograms to have more than one bin, we use a 3D bitmap indexing table, as shown in Figure 5.5(a). The three dimensions of the table are the ray histograms in the codebook, the bins for each histogram, and the probability range partitioned into sub ranges. The entries of the table are assigned using Equation 5.8.

$$bmp_{(c,i,p)} = \begin{cases} 1 & if\ p = 0,\ H_c(b_i) = 0 \\ 1 & if\ p \neq 0,\ f(H_c(b_i)) = p \\ 0 & Otherwise \end{cases} \tag{5.8}$$

where $bmp_{(c,i,p)}$ is the bit value stored at the table entry with index (c,i,p), c is the ID of a ray histogram in the codebook, i indexes a particular bin in the histogram, and p indexes a probability interval. The function $f(pr) = \lceil pr * I \rceil$ maps a probability pr to a corresponding probability interval, where I is the number of probability intervals. The time to search for a similar histogram using the bitmap indexing data structure is sensitive to the number of probability intervals. Using too few will result in increased search time because fewer histograms will be filtered out. On the other hand, too many probability intervals will increase the number of bit-wise vector operations. We define the size of each probability interval to be $T_h/2$ as a good trade off when defining the 3D bitmap indexing table. Thus, the number of probability intervals is $T_h/2$ + 1 in order to include zero probability bins. If a ray histogram H_n is not similar to any histograms currently in the codebook, then an entry is added for H_n in the 3D bitmap indexing table by computing a 2D bitmap using Equation 5.8 and inserted it at the bottom of the table.

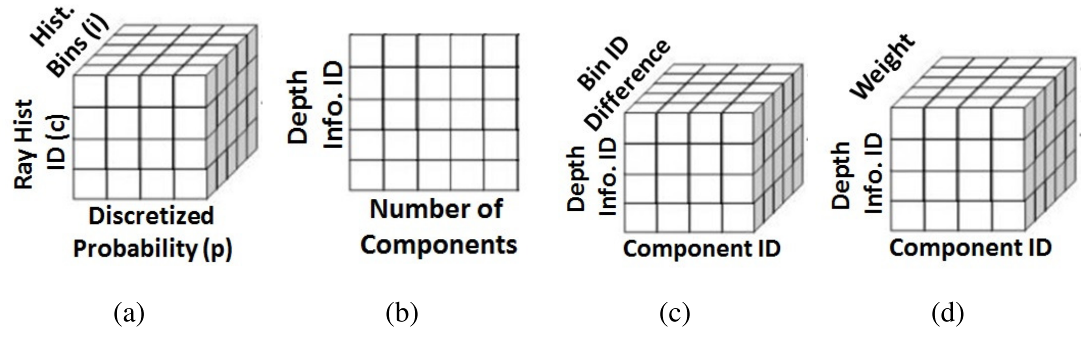


Figure 5.5: The ray histogram codebook defines a single 3D bitmap index table as shown in (a). The Bin-wise depth information codebook utilizes three bitmap indexing tables. A 2D bitmap indexing table, shown in (b), is used for searching given a component count. Two 3D bitmap indexing tables are used for searching based upon matching bin ID differences, shown in (c), and similar weight vectors, shown in (d).

Search Space Reduction: Given a ray histogram H_n, an efficient search for the most similar ray histogram in the codebook must only consider ray histograms that have the same set of bins with zero and non-zero probabilities to H_n. This calculation can be done quickly using the 3D bitmap indexing table as shown in Equation 5.9

$$C_1 = \Big(\bigwedge_{b \in H_n(b)=0} bmp_{(:,b,0)}\Big) \wedge \Big(\bigwedge_{b \in H_n(b)\neq 0} \neg bmp_{(:,b,0)}\Big) \tag{5.9}$$

where $\bigwedge$ and $\wedge$ are bitwise *AND* operators and $\neg$ is the bitwise *NOT* operator for bit vectors. Let $bmp_{(:,i,p)}$ be a bit vector for some ray histograms given bin ID i and probability interval p, where ":" is a wild card. The first term computes a bit vector encoding ray histograms having the same collection of zero probability bins as H_n. The second term performs a similar calculation for the non-zero probability bins. The resulting bit vector, C_1, encodes the ray histograms in the codebook with matching zero and non-zero probability bins with H_n.

In addition to the ray histograms identified in C_1, we also identify ray histograms that have similar probability bin values with H_n in corresponding non-zero probability bins. We observe that if the difference between any non-zero probability bin in H_n and a corresponding non-zero probability bin in a ray histogram H_e is larger than threshold T_h, then $HistD(H_n, H_e)$ must be larger than T_h and H_e can be avoided in the search. To identify ray histograms that satisfy this check with the threshold, we use Equation 5.10

$$C_2 = \bigwedge_{\forall b'} \Big(\bigvee_{p=f(H_n(b')-T_h)}^{f(H_n(b')+T_h)} bmp_{(:,b',p)}\Big) \tag{5.10}$$

where b' represents the set of bins containing non-zero probabilities in H_n, $\bigvee$ is the bitwise *OR* operator, and function f was defined in Equation 5.8. The term $\bigvee_{p=f(H_n(b')-T_h)}^{f(H_n(b')+T_h)} bmp_{(:,b',p)}$ computes a bit vector encoding ray histograms in the codebook whose differences between the non-zero probability bin value in bin b' and the corresponding bin value in H_n are within the subrange $[f(H_n(b') - T_h), f(H_n(b') + T_h]$. These bit vectors are combined with the *AND* operator to identify only those ray histograms that satisfy this condition over all non-zero probability bins into the final bit vector C_2.

The final bit vector is computed as $C_1 \wedge C_2$. These calculations result in a considerable reduction in the search space even though it may not filter out all the ray histograms H_e that fail to satisfy the check $HistD(H_n, H_e) \leq \mathrm{T}_h$. The ray histograms encoded in the final bit vector are compared with the ray histogram H_n to find the most similar histogram, H_e^s, using Equations 5.6 and 5.7 and the check $HistD(H_n, H_e) \leq \mathrm{T}_h$. If ray histogram H_e^s exists in the codebook, then H_n will index this histogram. Otherwise, ray histogram H_n is inserted into the codebook.

5.5.2 Bin-wise Depth Information Codebook

Bin-wise depth information described in Section 5.2.2 also incurs a considerable overhead storage cost in the proxy. We show how to efficiently encode this information into a codebook, which is similarly built as the ray histogram codebook by incrementally adding bin-wise depth information. Bin-wise depth information for each pixel ray is defined at the bins with non-zero probability values in the ray histogram.

Bin-wise Depth Information Similarity

We define a similarity metric to compare bin-wise depth information using three conditions: (1) The numbers of runs are the same, (2) The bin ID differences of all runs are the same, and (3) The L_1-*norm* distance of the weight vectors is smaller than a pre-defined threshold, T_d. The L_1-*norm* distance calculation on weight vectors is shown in Equation 5.11

$$L_1^{weight}(w_i, w_j) = \sum_{c=0}^{C-1} \left| w_i(c) - w_j(c) \right| \tag{5.11}$$

where w_i and w_j are the weight vectors found when comparing bin-wise depth information.

Bin-wise Depth Information Codebook Construction

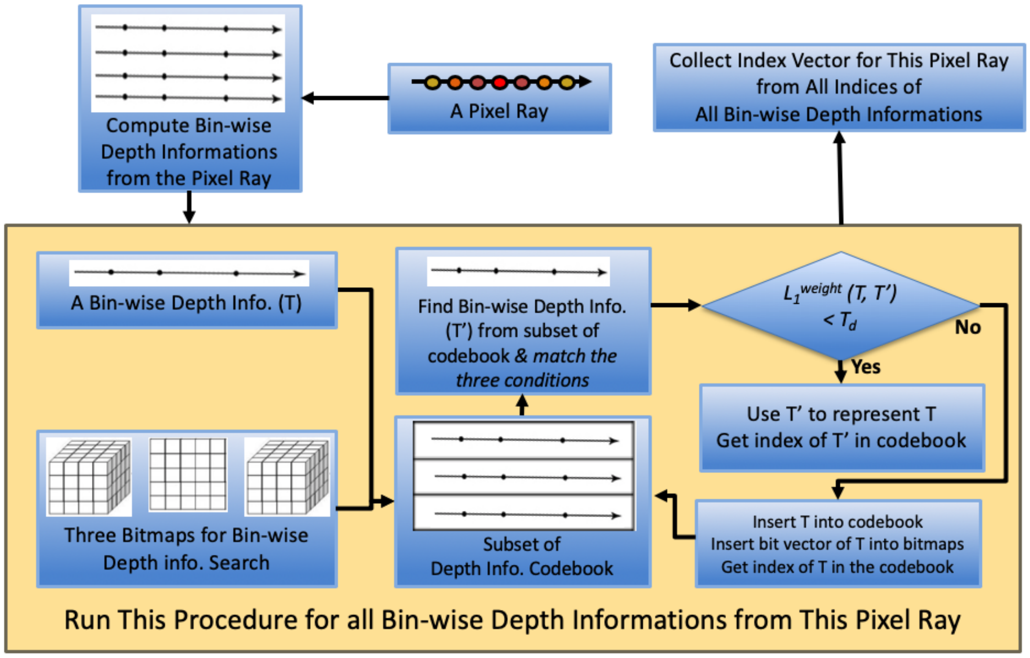
To build the codebook, the bin-wise depth information to be inserted into the codebook, DI_n, is compared with all entries currently in the codebook that satisfy the first and second conditions for similarity outlined in the previous section. These entries are filtered using the third condition, which must satisfy $L_1^{weight}(w_i, w_j) \leq \mathrm{T}_d$.

To avoid a costly linear search over all bin-wise depth information in the codebook, we again use bitmap indexing to reduce the search space. We use three bitmaps as shown in Figures 5.5 (b) - (d). A 2-dimensional bitmap indexing table (see Figure 5.5(b)) is used to quickly identify bin-wise depth information that have the same number of runs as DI_n. The bit assignment for this 2D bitmap index table is shown in Equation 5.12

$$compBmp_{(d,c)} = \begin{cases} 1 & if\ Comp(DI_d) = c \\ 0 & Otherwise \end{cases} \tag{5.12}$$

where d indexes the bin-wise depth information DI_d in the codebook and c is the number of runs. The function $Comp$ returns the number of runs of the given bin-wise depth information. Let $C_3 = comBmp_{(:,Comp(DI_n))}$ be the bit vector that encodes bin-wise depth information in the codebook that meets the first similarity condition. A 3-dimensional bitmap indexing table (see Figure 5.5(c)) is used to compute a bit vector C_4 that encodes bin-wise depth information in the codebook whose bin ID differences of all runs are the same as DI_n. Another 3-dimensional bitmap indexing table (see Figure 5.5(d)) is used to compute a bit vector C_5 that encodes bin-wise depth information in the codebook whose L_1^{weight} of DI_n is possibly less than T_d. The final bit vector is computed as $C_3 \wedge C_4 \wedge C_5$. The entry e

with the smallest $L_1^{weight}(w_e, w_n)$ that satisfies $L_1^{weight}(w_e, w_n) \leq T_d$, is used to represent DI_n. Otherwise, DI_n is inserted into the three bitmaps. Figure 5.6 illustrates the incremental algorithm to construct the codebook with the three bitmaps.



Compute Bin-wise Depth Informations from the Pixel Ray
A Pixel Ray
Collect Index Vector for This Pixel Ray from All Indices of All Bin-wise Depth Informations
A Bin-wise Depth Info. (T)
Find Bin-wise Depth Info. (T') from subset of codebook & match the three conditions
$L_1^{weight}(T, T') < T_d$
No
Yes
Use T' to represent T Get index of T' in codebook
Three Bitmaps for Bin-wise Depth info. Search
Subset of Depth Info. Codebook
Insert T into codebook Insert bit vector of T into bitmaps Get index of T in the codebook
Run This Procedure for all Bin-wise Depth Informations from This Pixel Ray

Figure 5.6: Steps to construct the bin-wise depth information codebook for a pixel ray.

5.6 Multi-view Rendering

While rendering when the user's viewpoint is fixed at a view sample provides scientists with a way to get a better understanding of the dataset through temporal evolution and transfer function changes, motion parallax is another important tool for data analysis. Rendering requires reconstruction of samples along each ray in the view-dependent proxy, applying the transfer function, and then blending these to compute a color. Rendering nearby views of a view sample is possible by using the parameters of the camera model. After reconstructing samples on all pixel rays at an index image, these samples are stored in a buffer. If a new view point is given and a sample on a ray from this new view point is requested, the closest reconstructed samples in the spatial domain is returned for color blending.

5.7 Evaluation

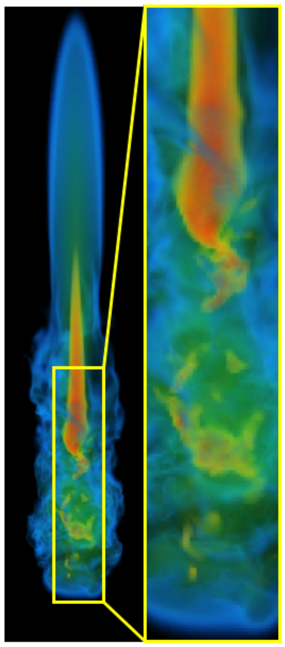
We present qualitative and quantitative results from experiments using three datasets. The Plume dataset is a simulation of Solar Plume for thermal down flow on the surface layer of the Sun and was provided by the National Center for Atmospheric Research. Our experiments used the vector magnitude field. Data resolution is 126x126x512 with 29 time steps (the raw size is 877 MB). The Isabel dataset is a pressure field of Hurricane Isabel from the IEEE Visualization 2004 Contest with a resolution of 500x500x100 and 48 time steps (the raw size is 4577 MB). The Combustion dataset was provided by Sandia National Laboratories where we used the mixture fraction field. The dataset has a resolution of 480x720x120 with 50 time steps (the raw size is 7910 MB).

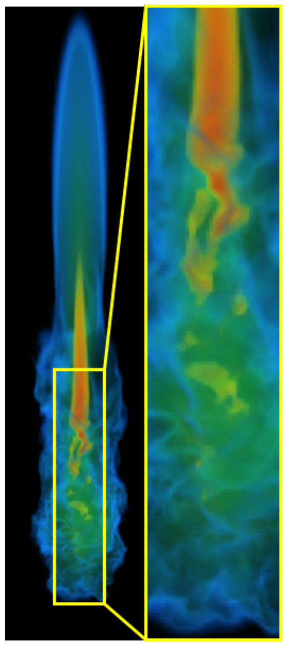
We compare our approach with five other approaches: a traditional interpolation approach, called *interpolation approximation*, a ray-based technique for volume rendering [88], called *IAF*, a state-of-the-art lossy compression technique called *ISABELA* [49], and two popular lossless compression techniques, called *zlib* (version 1.2.1) and *XZ* (version 5.2.3). Storage size comparisons are made with zlib and XZ since it is a lossless compression technique. Comparisons regarding rendering quality under varying transfer functions are made with the interpolation approximation, IAF, and ISABELA approaches. Comparisons regarding reconstruction error of scalar values along pixel rays are made with the interpolation approximation and ISABELA approaches since IAF techniques directly infer the pixel colors from their data structures. Results from the interpolation approximation approach used temporal down-sampling with 2, 5 and 10 subsampling time intervals. Results from the IAF approach use 128 bins in the histograms to produce its attenuation functions. The size of the compressed data from ISABELA's lossy compression is determined by the given error tolerance. We adjust the error tolerance to get a as small as possible compressed data from ISABELA. Because of the limitation of ISABELA, the smallest size of compressed data we can have is around 30% of the raw data.

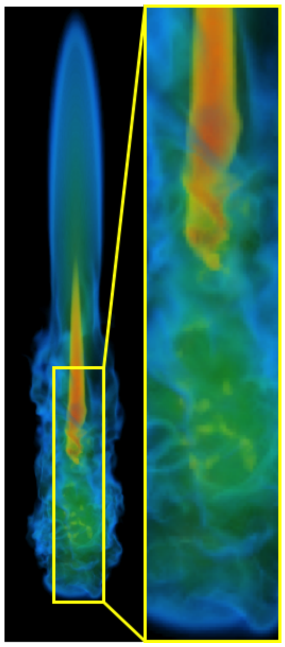
Proxy generation was computed on a machine with two 14-core Intel (Broadwell) Xeon E5-2680 v4 processors, 128 GB DDR3 Memory, one NVIDIA Tesla K80 GPU card. Rendering from the proxy was run on a desktop computer with an Intel 8-Core i-7-4770 3.40GHz CPU, 16 GB main memory, and an NVIDIA GeForce GTX 660 video card with 2 GB of memory. Results from our approach use a sampling interval of 10 time steps, 128 bins for each ray histogram, a ray quality threshold (T_r) of 0.99, a similarity threshold (T_h) of 0.1, a threshold (T_d) of 0.1 for the weight vector on depth information similarity, at most 5% using the decoupled ray representation, and a 512x512 image resolution where the image region covers entire dataset. We also present storage costs, preprocessing, and rendering times in this section.

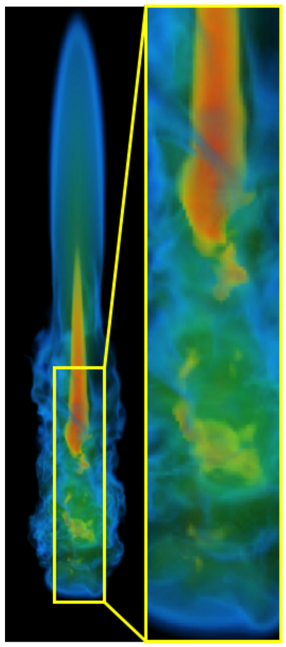
5.7.1 Qualitative Evaluation

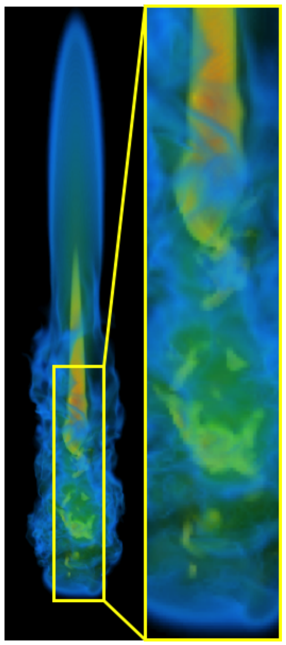
Figures 5.7, 5.8, and 5.9 compare our approach with the others. Figure 5.7 shows a comparison on the Plume dataset where the transfer function is defined to highlight high magnitude regions with orange and high opacity. The low and middle magnitude regions are displayed with blue and green and lower opacity. It is clear that our approach more accurately preserves features and depth cues in high magnitude regions. Figure 5.8 compares results on the Isabel dataset focusing on the hurricane eye, which is a low pressure region. When compared with the ground truth and other ray based approaches, our approach and IAF preserve the shape of the hurricane eye better than other approaches. Figure 5.9 compares results on the Combustion dataset focusing on the region with pure fuel mass, which is

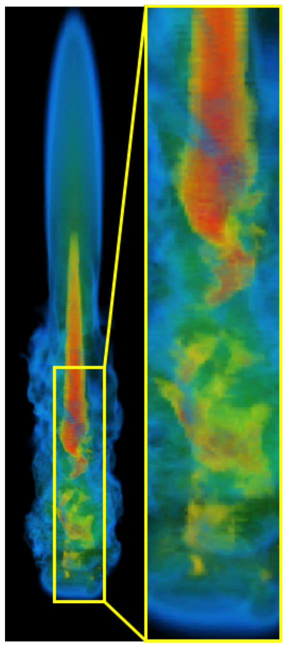

(a) Ground truth


(b) STI: 10


(c) STI: 5


(d) STI: 2


(e) IAF


(f) ISABELA

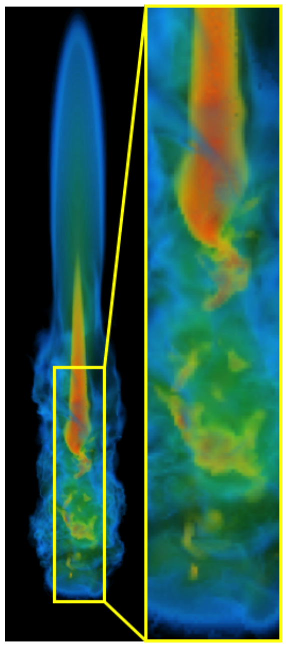
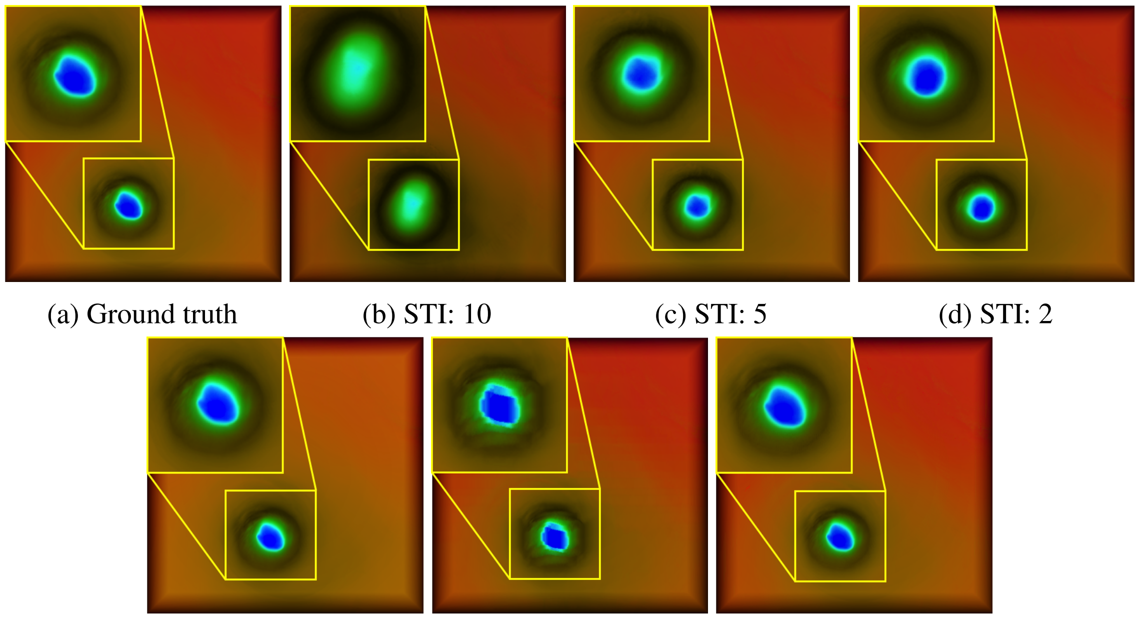

(g) Our approach

Figure 5.7: A comparison of rendering quality of the Solar Plume dataset at time step 6. Images at (b), (c) and (d) are approximated by interpolation from time steps 1 and 11, 3 and 8, and 5 and 7, respectively. STI stands for sampling time interval.



(a) Ground truth (b) STI: 10 (c) STI: 5 (d) STI: 2

(e) IAF (f) ISABELA (g) Our approach

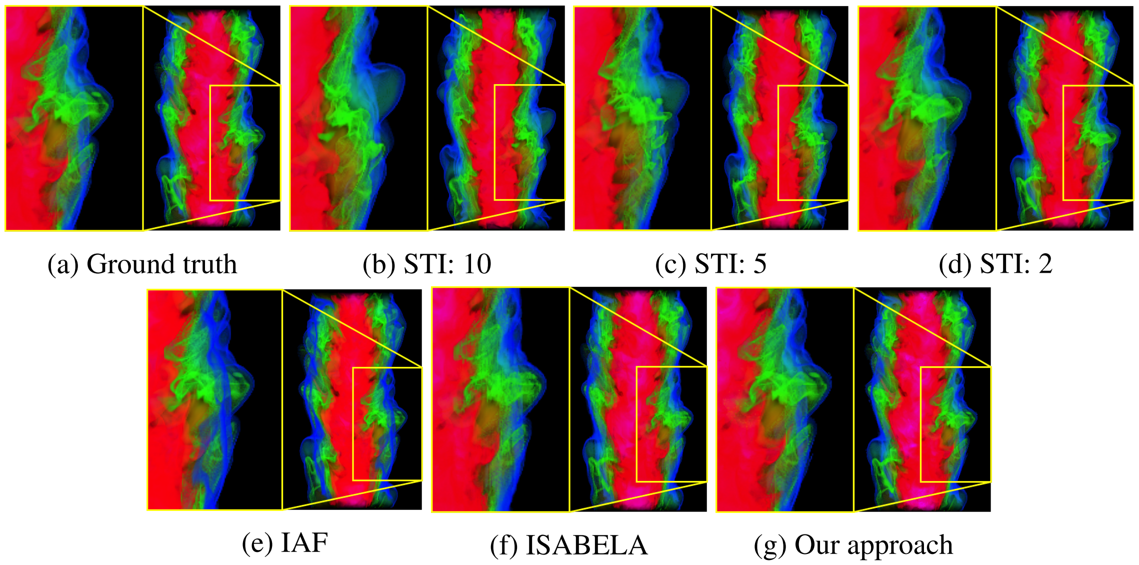
Figure 5.8: A comparison of rendering quality of the pressure variable of the Isabel dataset at time step 6. Images at (b), (c) and (d) are approximated by interpolation from time steps 1 and 11, 3 and 8, and 5 and 7, respectively.

colored with magenta and green. Our approach preserves complex details found in the pure fuel region very well.

Multi-view rendering results are shown in Figure 5.10 using the same view-dependent proxies and transfer functions that were used for the results in Figures 5.7, 5.8 and 5.9. When the view was rotated considerably to 45 degrees from the view sample, it is clear that overall structures are preserved when compared with renderings from the raw data.

5.7.2 Quantitative Evaluation

We use Root Mean Square Error (RMSE) to quantitatively evaluate pixel ray reconstruction quality over the entire time sequence between our approach and the interpolation approximation and ISABELA lossy compression approaches applied to the raw data. RMSE



(a) Ground truth (b) STI: 10 (c) STI: 5 (d) STI: 2

(e) IAF (f) ISABELA (g) Our approach

Figure 5.9: A comparison of rendering quality of the mixture fraction field of the Combustion dataset at time step 36. Images (b), (c) and (d) are approximated by interpolation from time steps 31 and 41, 33 and 38, and 35 and 37, respectively.

curves are shown in Figure 5.11 where the red curve is plot from our approach. The orange, green, and blue curves plot results from interpolation approximation and the black curve plots results from the ISABELA lossy compression approach. Note that we do not plot the RMSE at the sampled time steps. The plots illustrate that the reconstruction quality from our approach is better than almost all of the other approaches. The interpolation approximation approach, which is applied to the Isabel dataset with a high temporal sampling rate (blue curves), has a slightly lower error. The reason is that a high sampling rate stores raw data at many more time steps to provide a closer approximation in the temporal domain at non-sampled time steps. In addition, there is higher temporal coherence in the Isabel dataset where interpolation approximation can reconstruct the data well using a higher temporal sampling rate. The local minimums in the curves illustrate how our approach and naive interpolation approximation both exploit information from the sampled time steps at the

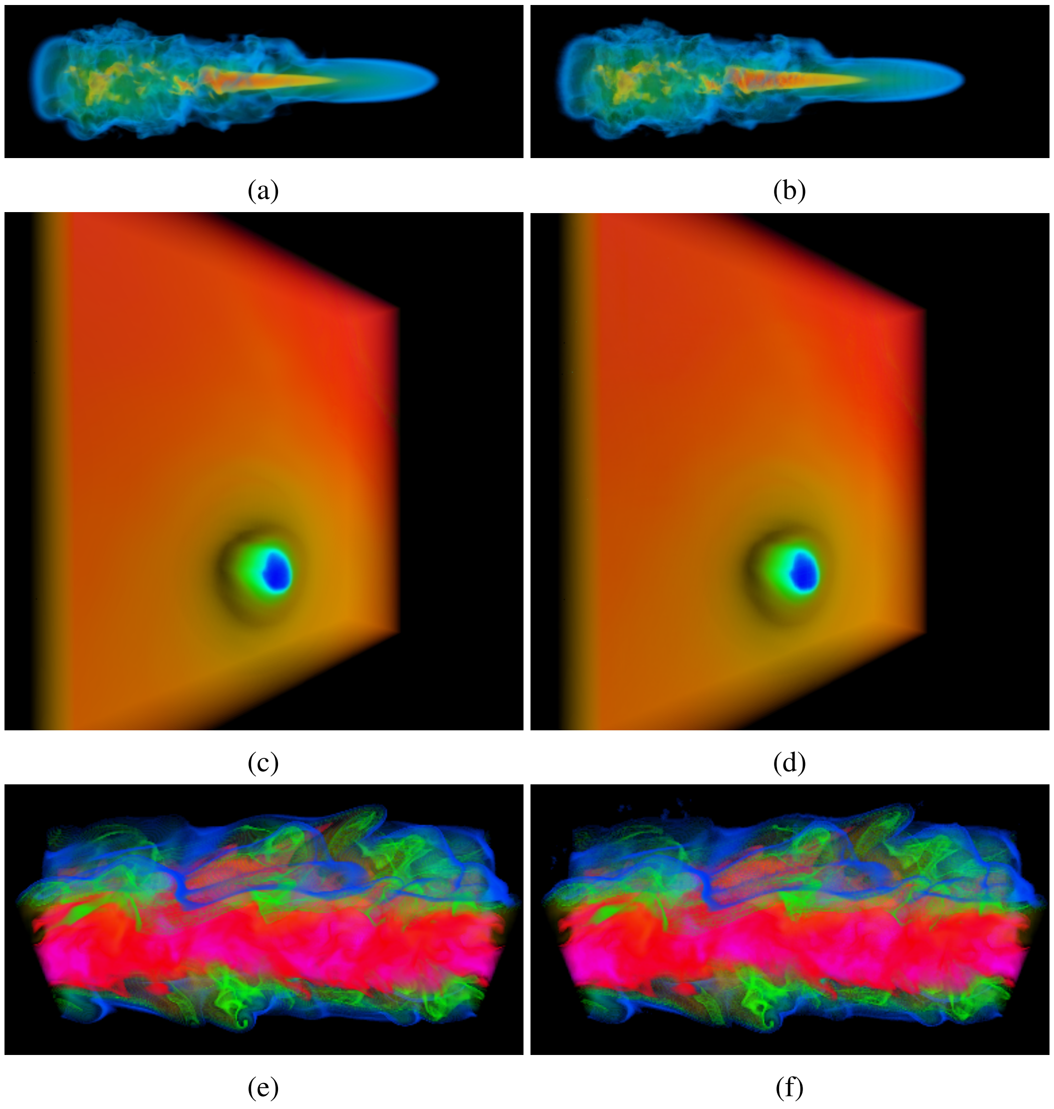
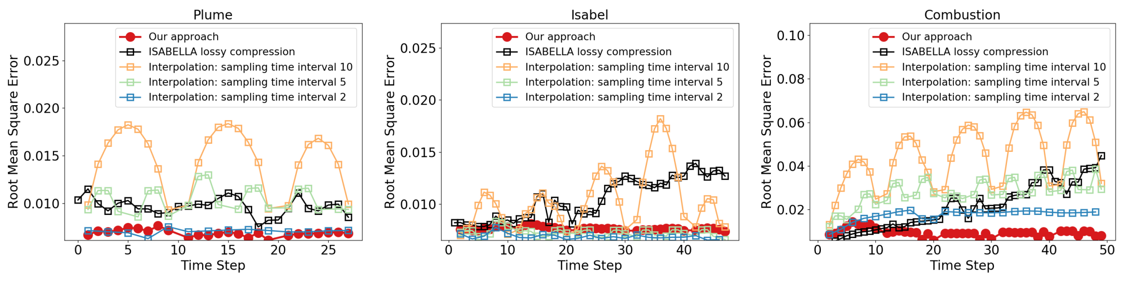


Figure 5.10: A comparison of multi-view rendering and rendering directly from the raw data. The results on the left are rendered from the raw data and results on the right are rendered by rotating 45 degrees from the view sample used in Figures 5.7, 5.8 and 5.9.

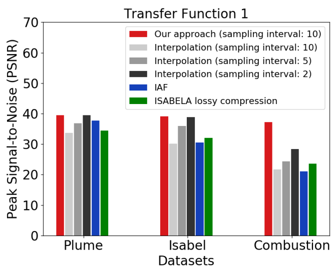
non-sampled time steps. The later time steps in the plots in graph (c) of the figure have higher error because the structure of the data becomes more complex over time making it more difficult for quality reconstruction.



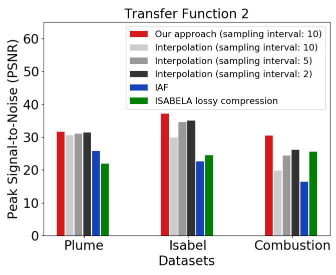
(a) (b) (c)

Figure 5.11: Quantitative evaluation of our approach and interpolation approximation and ISABELA lossy compression using RMSE.

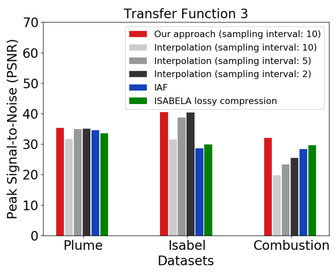
We use the average Peak Signal-to-Noise Ratio (PSNR) to quantitatively evaluate rendering quality over the entire time sequence of the datasets. A lower PSNR indicates rendering quality is worse when compared with the ground truth renderings. We use three transfer functions in our analysis. The first transfer function (TF1) uses a *rainbow* color map with at least two narrow ranges of data values mapping to high opacities. Other data values map to semi-transparent color as the context. This transfer function is used in the renderings shown in Figures 5.7, 5.8, and 5.9. The transfer function (TF2) uses a *sharp* color map where the opacity function maps all scalar values to high opacity. The samples close to the surface of the volume cube contribute color to the rendered image. The transfer function (TF3) uses the same color mapping as TF2, but the opacity function maps all scalar values to low opacity. Here, most of the samples in the data contribute to the rendered image. The graphs in Figure 5.12 show the difference in average PSNR of between the approaches. Our approach has higher image quality over the other approaches for all datasets and the transfer functions. We can also observe that our approach and IAF provided better image quality when using the transfer function TF3 compared with TF2. This is because both our approach and IAF could have incomplete sample order along rays and the incomplete sample order results in more pixel color shift if samples are more opaque.



(a)



(b)



(c)

Figure 5.12: Image quality comparison using average PSNR between our approach and interpolation approximation, IAF, and ISABELA lossy compression.

5.7.3 Proxy Storage Cost and Construction Time

We report in Table 5.2 storage costs of our approach applied to the three datasets that include only the proxies computed at the non-sampled time steps. We use a temporal sampling interval of 10 with a storage cost of 4 - 10% of the size to store the raw volumes found at the non-sampled time steps. Storage size is adaptable to data complexity where more coherence uses less storage, as shown in the storage costs for the Isabel dataset. When material shape is simple and smooth, our approach requires much less storage for bin-wise depth information but still represents this information well.

We also measured the storage cost over the same non-sampled time steps for the other approaches. The interpolation approximation approach with a sampling time interval of 2 stores 3 additional sampled time steps in each set of the nine non-sampled time steps. The additional cost to reconstruct data is 44%. A larger sampling time interval of 5 incurs an additional cost of 11%. Increasing the sampling time interval to 10 incurs no additional cost because it directly approximates the time steps by interpolation. The IAF technique needs additional 16%, 23% and 11% to store the data proxies for the Plume, Isabel, and Combustion datasets, respectively. The ISABELA lossy compression approach needs additional 31%, 32% and 29% to store the data at non-sampled time steps for the Plume, Isabel and Combustion datasets, respectively. Although zlib and XZ lossless compression techniques can preserve data quality perfectly, they have lower data reduction rates. Zlib and XZ incurs 90%, 88% and 47%, and 80%, 69% and 29% to store data at non-sampled time steps for the Plume, Isabel and Combustion datasets, respectively.

Table 5.2: Storage consumption of our approach. N.S. means Non-Sampled and Combus. stands for the Combustion dataset.

	Plume	Isabel	Combus.
Ray histogram (MB)	39.2	65.0	240.0
Bin-wise Depth information (MB)	8.2	0.44	226.0
Index image (MB)	33.5	77.0	80.3
Total size of our approach (MB)	80.9	142.44	580.3
N.S. volume size (MB)	775.2	4005.4	6960.9
Ratio of ours to N.S. volumes	10.4%	3.6%	8.3%

In the construction of the proxy, the average preprocessing time per view sample at a non-sampled time step to compute a data proxy for the Plume, Isabel, and Combustion datasets are 3.77, 11.53 and 18.04 seconds, respectively. This preprocessing time is accelerated by both GPU and multi-thread CPUs. Using the VTK-m library [65], the GPU is used to process pixel rays in parallel where the ray casting algorithm is executed to collect samples, select the appropriate ray representation, and construct the ray histogram and bin-wise depth information. Creating codebooks was executed using OpenMP with multi-threaded CPUs.

The time complexity to create the data proxies at a time step is $O(L*R+R^2*B)$, where R is image resolution, L is the number of samples used for all rays, and B is the number of bins defined for all ray histograms. The term $L*R$ represents the cost to reconstruct samples from the pixel rays and selecting an appropriate pixel ray which is linearly proportional to R and L. The term R^2*B represents the work to construct the ray histogram and bin-wise depth information codebooks. The computational bottleneck here is searching for a similar ray histogram or bin-wise depth information in the current codebook. The number of items in a codebook is linearly proportional to the number of rays which have been processed. The time to insert bin-wise depth information is proportional to the number of bins because a pixel ray can have at most B bin-wise depth information elements. The average fps during rendering from our experiments in Section 5.7.2 are 8.3, 5.4 and 7.8 for the Plume, Isabel, and Combustion datasets, respectively. Rendering is accelerated using a GPU and the VTK-m library.

5.8 Discussion

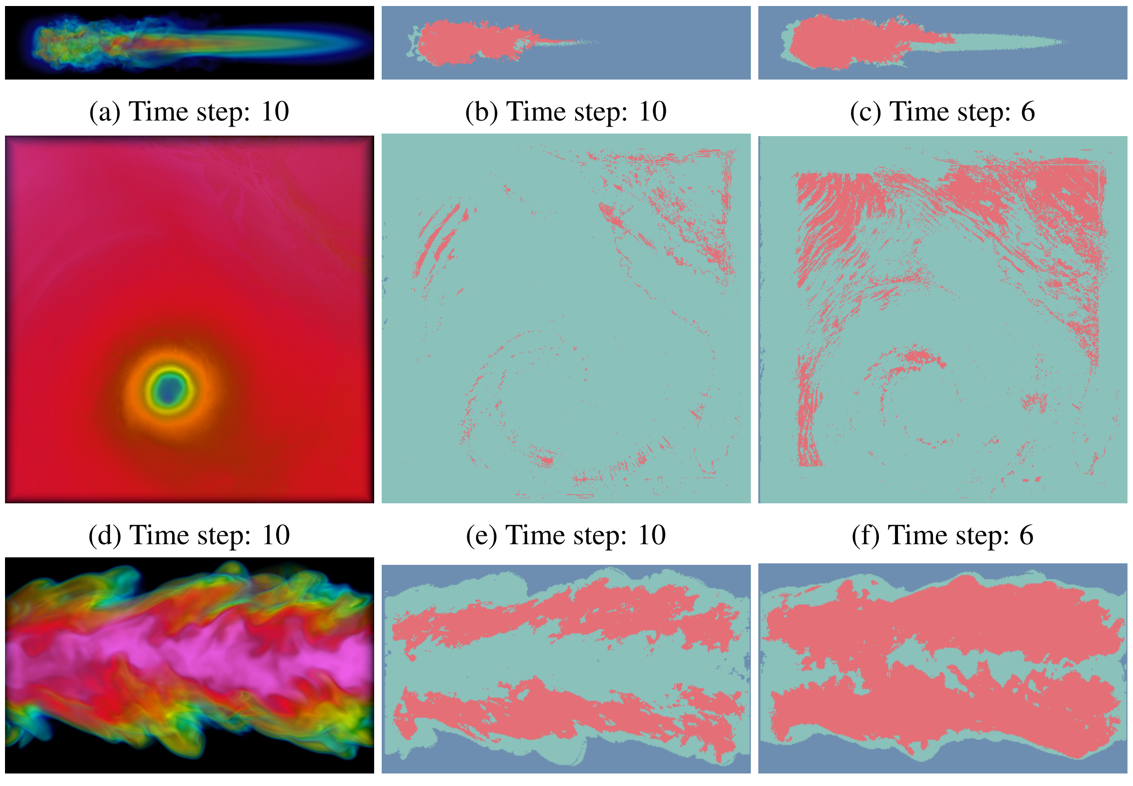
5.8.1 Pixel Ray Representations

We discuss the use of the *interpolation ray*, *depth profile ray* and *decoupled ray* representations and what role each plays in the proxy. Figure 5.13 illustrates the use, distribution, and placement of the three representation types across an index image in various scenarios. The image in the left most column, see (a), (d), (g), are renderings from the raw data and are shown for comparison purposes. The images in the middle column, see (b), (e), and (h), are color coded index images computed at the same time steps and view used in the left most column results. This is a non-sampled time step close to a sampled time step when generating our proxy. The blue, green and red colors indicate the use of the *interpolation ray*, *depth profile ray*, and *decoupled ray* representations at a pixel, respectively. The last column, see (c), (f), and (i), is color coded index images computed at a non-sampled time step in the middle between two sampled time steps.

The interpolation ray representation is defined at pixel rays intersecting regions with scalar values changing smoothly over time. This is illustrated in the Plume and Combustion datasets where scalar values changes are small and smooth over time at the peripheral areas. This is also illustrated at the peripheral areas in Figures 5.13 (b), (c), (h) and (i). The depth profile ray representation was defined at pixel rays intersecting regions with similar structure. For example, the Plume dataset has a higher magnitude region surrounded by a region with lower magnitude. A pixel ray at the sampled time step passing through the region can capture trends that will be similar along pixel rays at the nearby non-sampled time steps in the same region. The depth profile ray representation is heavily used in the Isabel dataset because lower altitudes usually have a higher pressure and higher altitudes usually have lower pressure. This is illustrated in the index images shown in Figures 5.13 (e) and (f), where the depth profile rays are colored green. Pixel rays that pass through regions with drastic changes in the scalar field over time are usually represented well with the decoupled ray representation due to the lack of temporal coherence from the neighboring sampled time steps. This is illustrated in the index images in Figures 5.13 (b), (c), (h) and (i), where the decoupled ray representation, shown in red, is chosen to represent regions with larger value changes over time. The pixel rays shown in the index images from the middle column can be represented well with temporal coherence than the representations shown in the last column in the figure.

5.8.2 Parameter Study

There are a few parameters that need to be determined for proxy generation that affect the trade off between storage size and rendering quality, which is shown in Figure 5.14. Since the threshold T_r determines reconstruction quality along pixel rays, decreasing it will increase the RMSE of the reconstruction and decrease the proxy size, as shown in Figures 5.14 (a) and 5.14 (b). The interpolation ray and depth profile rays will more likely be selected in this case, which require less storage than the decoupled ray representation. The ray histogram similarity threshold T_h affects the size of the ray histogram codebook. Larger values of the threshold allows for larger groups of ray histograms that can be represented by a single representative histogram in the codebook, which results in a smaller codebook size and reduced accuracy. This is shown in Figures 5.14 (c) and 5.14 (d), where the ray histogram codebook size decreases and the RMSE increases for higher threshold values. Two parameters affect the proxy size and reconstruction quality when using the decoupled ray representation. Larger values for the upper bound ratio e increase the accuracy for reconstruction decreasing the RMSE. Figure 5.14 (e) shows that the RMSE of


(a) Time step: 10 (b) Time step: 10 (c) Time step: 6

(d) Time step: 10 (e) Time step: 10 (f) Time step: 6

(g) Time step: 40 (h) Time step: 40 (i) Time step: 36

Figure 5.13: The placement of the three pixel ray representations in index images. The left most column shows renderings from the raw data. The middle column shows index images created at non-sampled time steps close to sampled time steps. The interpolation and depth profile ray representations are mostly used here. The right most column shows index images created at non-sampled time steps in the center between two sampled time steps. The decoupled ray representation is used more here than in the middle column case because there is less temporal coherence that can be leveraged.

the Combustion dataset decreases when the upper bound ratio increases. Though, the RMSE does not significantly change in the results on the Plume and Isabel datasets because most rays in these datasets can be represented well with the interpolation ray and the depth profile ray representations. Increasing the upper bound ratio does not increase the number of rays represented by the decoupled ray representation. Similarly, the proxy sizes using the Plume and Isabel datasets change slightly, as shown in Figure 5.14 (f). For a dataset containing more complexity, such as the Combustion dataset, increasing the upper bound ratio can be a good trade off between proxy size and quality. The bin-wise depth information similarity threshold T_d has similar trade offs with the ray histogram similarity threshold T_h. Larger values of the threshold T_d define a higher error tolerance for the codebook and result in a smaller codebook size with lower quality. This is illustrated in Figures 5.14 (g) and 5.14 (h). Figure 5.14 (j) shows that the data proxy size increases if more samples are taken from rays. The reason is that more samples result in more non-zero bins and increased size in the bin-wise depth information. The RMSE decreases when more samples are taken on rays, which is shown in Figure 5.14 (i).

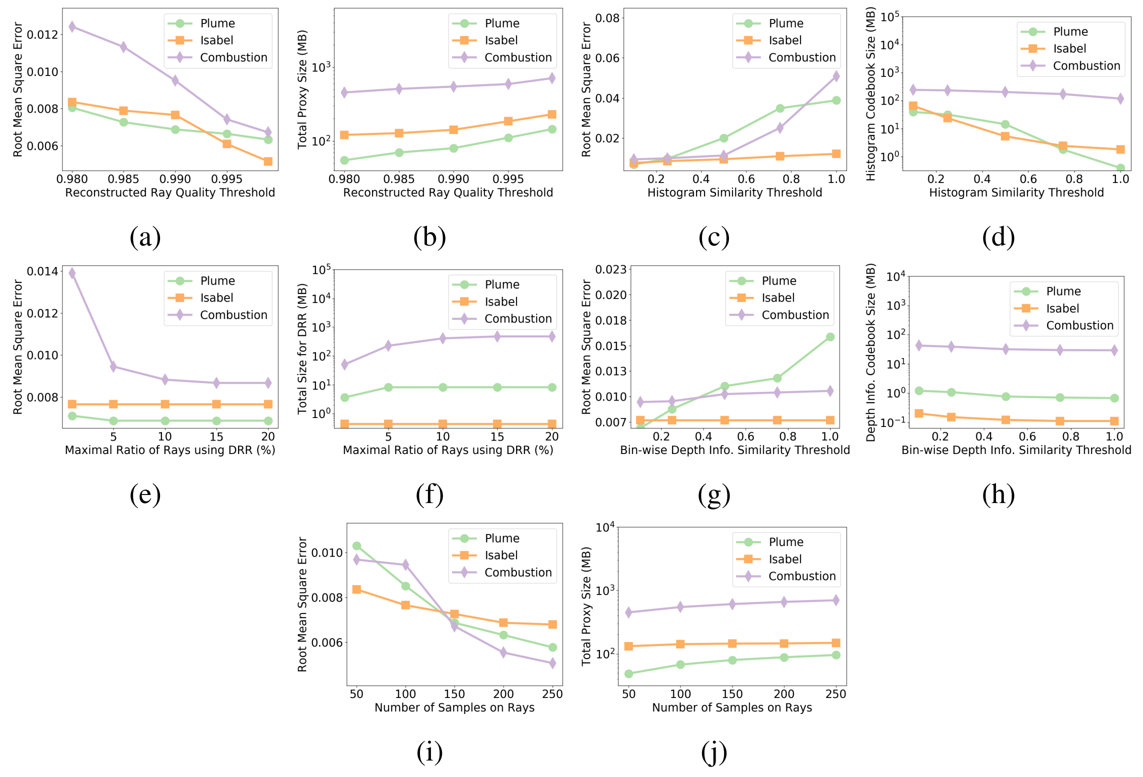


Figure 5.14: Parameter study on thresholds of the reconstructed ray quality, histogram similarity, bin-wise depth information similarity, maximal ratio of rays using the decoupled ray representation and number of samples on rays. DRR stands for decoupled ray representation.

5.8.3 View Selection

Selecting view samples to locate index images can be determined manually or automatically. The scientists may have prior-knowledge about the data generated by their simulations. Thus using their experts domain knowledge, scientists can manually select views which can benefit their analysis goals. However, if prior-knowledge is not available, the view angle which provides a larger variation of important features over time is usually preferred. The view selection techniques [12, 44] can automatically choose several top salient views to generate our data proxies by analyzing the statistical characteristic of features in the dataset. Our multi-view rendering technique is able to show scientists the shape evolution of features for improved data analysis.

5.9 Conclusion and Future Work

We present a new approach to high quality visualization and analysis of large time-varying data by replacing most of the volumes in the time series with proxies using a novel ray-based representation at the non-sampled time steps. To capture the evolution of features at non-sampled time steps, our representation decouples a ray into a distribution of the sample values and information encoding the locations of these samples along a pixel ray. This information is stored into compact cookbooks. Temporal coherence is utilized to further reduce the size of the proxy in two ways. Depth information at pixel rays defined at a sampled time step provides a depth profile that may be used to locate samples at pixels defined at non-sampled time steps. Regions of a volume that have constant or smooth changing scalar values can be reconstructed on pixel rays from nearby sampled time steps using linear interpolation. In future work, We will apply our approach to multivariate and vector datasets. We need to determine a new distribution representation to replace the ray histogram for scalar values because a multivariate histogram will be expensive in terms of storage cost. Our approach can also be applied to applications such as pathline tracing for vector datasets and isosurface rendering for scalar datasets.

Chapter 6: Statistical Super Resolution for Data Analysis and Visualization of Large Scale Cosmological Simulations

Exploring and modeling parameters of the initial conditions of our universe is one of the most important tasks in cosmology, which can help cosmologists to build more precise universe models [5, 20]. Recently, a cosmological simulation code called Nyx, developed by Lawrence Berkeley National Laboratory, is used for simulating large-scale cosmological phenomena [5]. It requires performing many cosmological simulations to conduct a large-scale, multi-dimensional parameter study to find the "best matching" set of parameters that matches the current state of the observable universe. In addition, every cosmological simulation output is 'scientifically rich' [77]. Cosmologists may discover new insights into the physical model by interactively exploring and analyzing the datasets [33]. When the size of parameter study is small, the data analysis pipeline consists of running the simulation with different parameter inputs, writing the output to disk, and repeatedly loading the data from disk to visualize and analyze the datasets. To compare ensemble simulation runs with observations and understand the evolution of the universe, it requires high-resolution simulations to match the desired accuracy. This easily produces datasets at the order of petabytes, and the traditional data analysis pipeline becomes infeasible because of disk bandwidth constraints and limited capacity of storage devices. Disk bandwidth constraints result in longer I/O time. Also, only a portion of the output can be kept in disk due to the limited capacity of disks. Therefore, it is essential to develop an approach to improve the whole cosmological data analysis pipeline.

To tackle such large-scale data problem, the concept of *in situ* data processing has been proposed. *In situ* techniques use supercomputer resources to generate compact data representations for analysis without moving the raw datasets. A naive approach is to down-sample the datasets, but the quality of this data representation would be insufficient for many data analysis tasks [80]. Statistically modeling the datasets can quantify the error of the down-sampled datasets, but the quality of the data representation is not improved significantly [23, 56]. Wang et al. [93] and Soumya et al. [26] propose accurate and compact data representations, but they need a long data processing time. Other *in situ* techniques produce the data representations for specific data analysis or visualization requirements [2, 33, 94].

The goal of this work is to develop an efficient technique which can compactly represent multi-variate (quantity) cosmological datasets and used for various data analysis and visualization tasks. We propose an *in situ* technique called *statistical-based super-resolution* (SbSR) for cosmological simulations. In computer vision and image processing community, the super-resolution technique [52, 99] enhances the resolution of an image using a prior knowledge database to predict high-resolution images from low-resolution images. Prior knowledge in our statistical-based super-resolution is created by the raw data from a small portion of simulation runs and used to reconstruct down-sampled data with other simulation parameters. The down-sampling is accomplished in situ by a Gaussian Mixture Model (GMM). We use GMMs to compactly represent low-resolution cosmology data, because data values of some quantities (e.g., *density*), from different cosmological simulation runs, can spread in an extremely wide value range, and a 3D down-sampled region can contain a large number of data values. Distributions provide richer information to facilitate the process of high-resolution data reconstruction. To minimize computation time for *in situ* statistical down-sampling, a parallel Expectation-Maximization (EM) algorithm based on the VTK-m framework [65] is implemented to train multiple GMMs concurrently by utilizing either GPU or multi-threads CPU. We show that our *in situ* statistical down-sampling can significantly reduce data storage and does not increase the total simulation time.

The remainder of this chapter is organized as follows. Section 6.1 is an overview of our approach. Section 6.2 describes the in situ statistical down-sampling. The prior knowledge creation is introduced in Section 6.3. Section 6.4 presents the process to reconstruct statistical down-sampled data to high-resolution data. Section 6.5 shows the quantitative and qualitative evaluations. Section 6.6 is the discussion of our approach. Finally, Section 6.7 concludes the paper and provides future work.

6.1 Overview

The Nyx cosmological simulation [5] solves compressible hydrodynamics with an N-body treatment of the dark matter. The simulation can have up to 10 different simulation input parameters. Fixing the values for these simulation input parameters will produce one *simulation run*. In each simulation run, Nyx produces both derived quantities volumes and particle data. In this work, we focus on data analysis of the derived quantities. Nyx can output up to seven derived quantities, *density*, *xmom*, *ymom*, *zmom*, *Temp*, *rho_e* and *phi_grav*. Each quantity is a regular grid and can have up to 4096^3 grid points according to the settings of the simulation. Nyx produces hundreds of time steps in each simulation run. The number of time steps varies according to the converging speed of the simulation run. Therefore, if the domain experts have P simulation input parameters of interest to study, have Q quantities of interest to study, take I value samples on each input parameter, set the output resolution of each quantity of interest to R^3 and each simulation run has T time steps, the scale of raw data size will be $O(I^P * T * Q * R^3)$. This raw data size usually is hundreds of TBs to several PBs. To output such huge datasets to disks and access these datasets in a post data analysis pipeline is challenging. We propose an *in situ* statistical super-resolution to address this challenge.

Figure 6.1 shows the workflow of our method for a single time step and a single quantity of Nyx. The leftmost circles in Figure 6.1 are simulation parameter inputs and each one is fed to Nyx to perform one simulation run. Our technique uses latin hypercube sampling [22] to draw a small number of simulation parameter inputs to run the simulation (blue arrows). These simulation runs will write the full resolution data to storages. These full resolution datasets are used to compute the prior knowledge. For other simulation parameter inputs (e.g. brown circles), we perform statistical down-sampling *in situ* to reduce the datasets

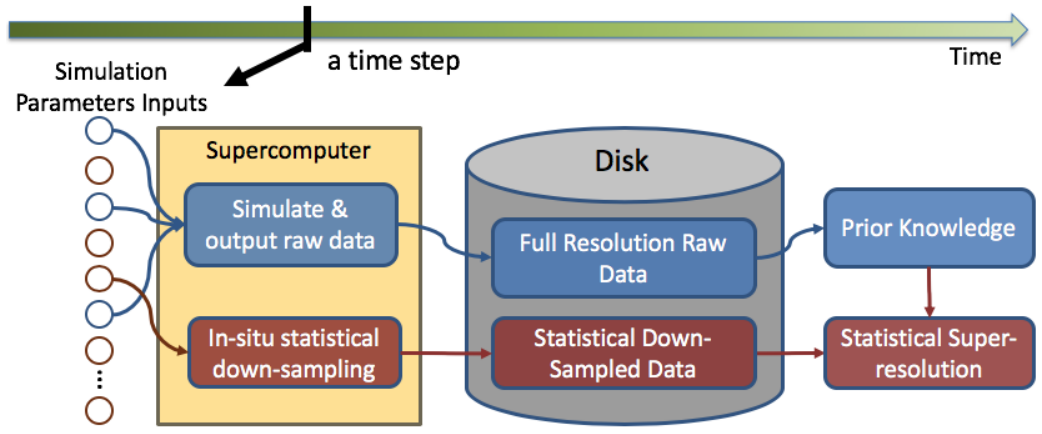

a time step
Time
Simulation Parameters Inputs
Supercomputer
Simulate & output raw data
In-situ statistical down-sampling
Disk
Full Resolution Raw Data
Statistical Down-Sampled Data
Prior Knowledge
Statistical Super-resolution

Figure 6.1: We illustrate the overview of the workflow of our proposed technique.

generated from the simulation. A local data block size, B, is pre-set by users and a point (ℓ_p) in the low-resolution space is created from a B^3 data block (h_p) in the raw (high) resolution. When performing post data analysis, our approach uses the information from the prior knowledge to reconstruct the statistical down-sampled data to high-resolution data. We create independent prior knowledge for each quantity of interest. Because Nyx produces multiple time step data and the data has coherence in the temporal domain, we create the prior knowledge only at every τ time steps. Figure 6.2 illustrates this scheme. When reconstructing the statistically down-sampled data at time step t (red dot in Figure 6.2) to high resolution, the closest prior knowledge in the temporal domain is used.

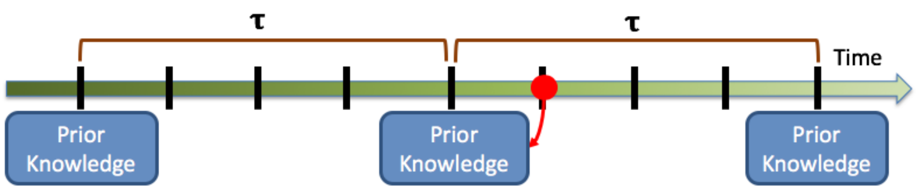


Figure 6.2: We display the overview of prior knowledge creation and usage in the temporal domain.

6.2 *In situ* Statistical Down-sampling

A way to reduce the size of any dataset is to down-sample the data and store it at a lower resolution. The down-sampling process usually applies a filter to each spatial local region to calculate one value, such as an average value, to represent the region. Although this approach can effectively reduce the data size without complicated computation, only limited information is kept in the low-resolution proxy and it introduces error in the post analysis. For example, the minimum and maximum data values in some local data blocks of the density quantity can have a difference on the 10^{12} scale. The average value will introduce severe bias in the data analysis. Therefore, instead of using one value to represent the data in a spatial local block, we suggest a statistical approach to down-sample the dataset. The data in a local block is summarized by a distribution. Distributions can keep abundant information, such as the occurrence probabilities of values and statistical characteristics in the data block, to have the potential to better reconstruct data back to high resolution.

6.2.1 Gaussian Mixture Model

To compactly store a local data block and model the wide data value range in cosmological datasets, we adopt the Gaussian Mixture Model (GMM) to represent the distribution. GMM is a popular parametric distribution model which consists of multiple weighted Gaussian distributions. The unique statistical properties of Gaussian distribution allow GMM to model arbitrary distribution shape in a compact and flexible fashion. The proposed statistical down-sampling method uses GMM described in Equation 6.1 to model and store the samples in a high-resolution data block (h_p) corresponding to a low-resolution point (ℓ_p). Figure 6.3 illustrates an example of statistical down-sampling a volume of a quantity.

$$g_p(x) = \sum_{i=0}^{K} w_i * (x|\mu_i, \sigma_i) \tag{6.1}$$

where $g_p(x)$ is a probability density function which is represented by a GMM. K is the number of Gaussian components. w_i, μ_i and σ_i are the weight, mean and standard deviation of the i^{th} Gaussian component, respectively. The sum of the weights, $\sum_{i=0}^{K} w_i$, in a GMM must be 1. After statistical down-sampling, all local data blocks, from every quantity of interest and time step, are represented by independent GMMs.

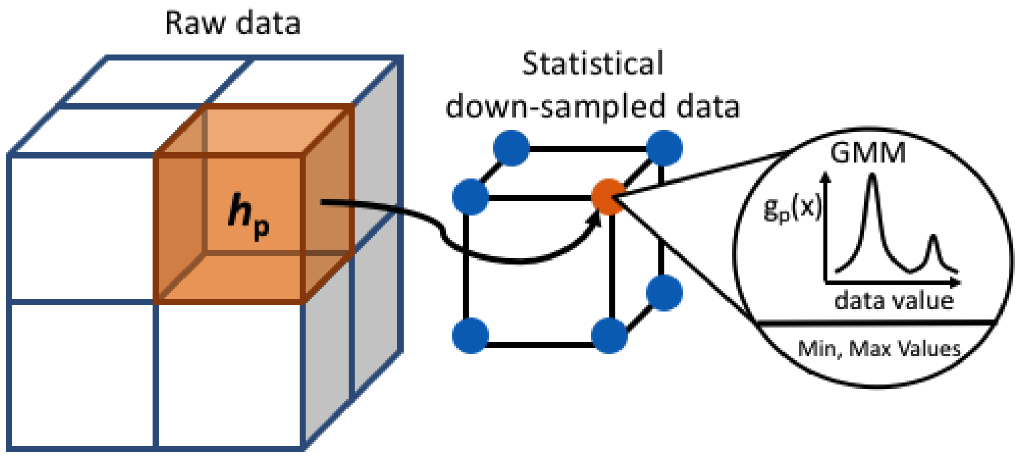


Figure 6.3: We illustrate the statistically down-sampling of a raw volume to 2^3 low-resolution grids. The orange block is a local data block. The orange point is a down-sampled point represented by a GMM which is computed from the orange block, and a minimum and a maximum value of the orange block.

6.2.2 *In situ* Workflow

To implement the statistical down-sampling for Nyx, we append an *in situ* call at the end of each time step. The *in situ* processing collects all data of all quantities of interest which reside in the same computing node to perform the statistical down-sampling. Our statistical down-sampling has to compute the GMM, minimum value, and maximum value for each local data block. Minimum and maximum values are used to filter out the values that are out of the value range when reconstructing the down-sampled data to high resolution. Expectation-Maximization (EM) algorithm is used to estimate the weights, means, and standard deviations of a GMM. The EM algorithm is an iterative method which adjusts the parameters of GMM to maximize the likelihood of input data values. To minimize the time for GMM modeling, we implement a data parallel EM algorithm based on the VTK-m framework [65]. The algorithm takes data from all data blocks of all quantities from one time step on one computing node to estimate the parameters of all GMMs in parallel, which can fully utilize the computational resource on the computing node. In addition, our system allows users to flexibly control the time of *in situ* call by sub-sampling partial data to run the EM algorithm.

6.3 Prior Knowledge

To reconstruct a down-sampled point back to a high-resolution data block, the data values can be obtained by re-sampling from the GMM. However, the data values lack the spatial location information to allocate them in space. Without predicting the spatial location for the re-sampled data values, the reconstruction quality will be poor. To address this, we propose prior knowledge to predict the location of the re-sampled data values from a GMM. The prior knowledge is computed from the raw data of a small number of simulation runs. We call these simulation runs *prior simulation runs*. The data generated by different parameter inputs have a correlation, and by creating the prior knowledge from the prior simulation runs, they are used to compensate the lack of the location information. The prior knowledge consists of two parts. One is the spatial information dictionary and the other one is a feature matching metric. The spatial information dictionary is a mapping between feature vectors to location information, and is used to assign each data value from a GMM to a position of high-resolution data. A feature vector is a descriptor of a given data block and computed from the statistical moments of the local neighboring data blocks. Figure 6.4 gives an overview of the spatial information dictionary. A feature matching metric defines a distance space to match a feature vector for a down-sampled point to a feature vector in the spatial information dictionary to access proper location information for high-resolution data reconstruction. This section introduces the details of the spatial information dictionary and the feature matching metric.

6.3.1 Spatial Information Dictionary

The spatial information dictionary maps feature vectors to spatial location information. Each entry in the dictionary is computed from a B^3 data block of the raw data from the prior simulation runs, where B is pre-defined down-sampled block size. A feature vector consists of the statistical moments from itself and those of the neighboring data blocks. We use the means and standard deviations from the data blocks to build the feature vector. This is illustrated in Figure 6.4. The feature vector is normalized by the minimum and maximum mean values in the vector to capture the relative trend of data values in neighboring regions. Equation 6.2 shows a feature vector of a data block, h_p.

$$f_{h_p}^{\Omega} = \frac{[\mu_{\Delta_0}\ \mu_{\Delta_1}\ ...\ \mu_{\Delta_{n-1}}\ \sigma_{\Delta_0}\ \sigma_{\Delta_1}\ ...\ \sigma_{\Delta_{n-1}}]}{Max([\mu_{\Delta_0}\ ...\ \mu_{\Delta_{n-1}}]) - Min([\mu_{\Delta_0}\ ...\ \mu_{\Delta_{n-1}}])} \tag{6.2}$$

where $f_{h_p}^{\Omega}$ is the feature vector of data block h_p. $\Omega = \{\Delta_0, \Delta_1, ..., \Delta_{n-1}\}$ defines n functions to locate the neighboring data blocks, which also includes h_p itself. The numerator is a vector, and μ_{Δ_i} and σ_{Δ_i} are the mean and standard deviation calculated from data block $h_{\Delta_i(p)}$. If a high resolution data block $h_{\Delta_i(p)}$ is out of the global spatial domain of the dataset, we pad the outside region to zero to calculate the mean and standard deviation. $Max([\mu_0\ ...\ \mu_{n-1}])$ and $Min([\mu_0\ ...\ \mu_{n-1}])$ are the maximum and minimum mean values, respectively. Since the neighbor locality function Ω is predefined, we use f_{h_p}, ignoring Ω, to indicate the feature vector of h_p in the rest of this paper.

Due to the spatial, temporal and parameter correlation between simulation runs, if the distributions of the data values in neighboring data blocks are similar, the locations for data values in the center data block are likely to be similar, too. For example, the locations of the largest values are similar in the local block. When reconstructing a down-sampled point back to high resolution, the re-sampled data will be assigned to space by using the location distribution of data values in the data block, whose feature vector is most similar to the feature vector of the down-sampled point. For example, the largest/smallest value among the re-sampled data will be assigned to the location which has the largest/smallest value in the data block. The detail of computing the feature vector from the down-sampled data is introduced in Section 6.4.

The distribution of data sample locations for each data block in the prior simulation runs is retrieved from the data block then stored as the location information in the dictionary. The location information (s_{h_p}) in the dictionary is represented by a sequence of location indexes. The sequence is sorted according to the corresponding data values and only the location indexes are stored in the dictionary without the data values. We convert the 3D location index to 1D, and this usually requires less than 2 bytes to save a location index in a local data block. Essentially, the stored location information is a function which maps the samples' order defined by samples' value among a group of samples to location indexes in the high-resolution domain. By giving a group of re-sampled data values and calculating the order of data values, s_{h_p}, we can map data values to locations in high-resolution space. Figure 6.4 illustrates the spatial information dictionary creation.

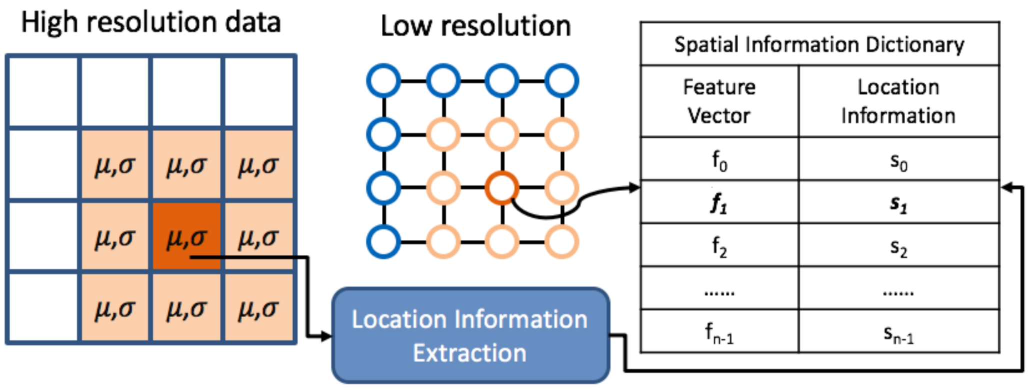


Figure 6.4: We illustrate a 2D example of creating an item, f_1 and s_1 in the spatial information dictionary from the dark orange point in low-resolution space. The orange points are the neighboring points, and the orange blocks in high-resolution data are the corresponding neighboring data blocks. The mean and standard deviations are directly computed from the data in these data blocks to generate f_1. The location information s_1 is extracted from the dark orange data block.

6.3.2 Feature Matching Metric

To recover low-resolution GMM point to high resolution, we need to find a matching feature vector in the dictionary to get the location information. This requires a feature matching metric to define a distance space between feature vectors of the point in the dictionary such that corresponding location information of a feature in the dictionary is more likely to better reconstruct a down-sampled grid point to a high resolution. The distance metric, $D(f_{h_i}, f_{h_j})$, is constructed from prior simulation runs to create a distance space which separates feature vectors if the location information retrieved from the corresponding data blocks of a feature vector does not well reconstruct the disorder data samples collected from the other one. For example, U_{h_i} and U_{h_j} are data samples' collected from data blocks h_i and h_j. Locations of data samples are unknown in U_{h_i} and U_{h_j}. $r_{h_i}^{h_j}$ is a reconstructed block by combining samples from U_{h_i} and the location information (s_{h_j}) which is retrieved from h_j and $r_{h_j}^{h_i}$ is a reconstructed block by combining samples from U_{h_j} and the location information (s_{h_i}) which is retrieved from h_i. We call the reconstruction error of $r_{h_i}^{h_j}$ and $r_{h_j}^{h_i}$ the cross reconstruction error of h_i and h_j. $D(f_{h_i}, f_{h_j})$ should approximate the cross reconstruction error of h_i and h_j. The cross error between h_i and h_j is shown in Equation 6.3.

$$CE(h_i, h_j) = \frac{NR(h_i, r_{h_i}^{h_j}) + NR(h_j, r_{h_j}^{h_i})}{2} \tag{6.3}$$

where $NR(h, r)$ is the function to compute normalized RMSE between the raw and reconstructed data. The detail is shown in Equation 6.4.

$$NR(h, r) = \frac{\sqrt{E^2(h, r)}}{Max(h) - Min(h)} \tag{6.4}$$

where h is a true data block and r is a reconstructed data block in high resolution. $Max(h)$ and $Min(h)$ calculate the maximum and minimum values in h. $E^2(h,r)$ calculates the average of the square differences of all data values at the same location. The fundamental form of distance between two feature vectors in our system is Mahalanobis distance [62] and Equation 6.5 shows the Mahalanobis distance.

$$D(f_{h_i}, f_{h_j}) = \sqrt{(f_{h_i} - f_{h_j}) * S^{-1} * (f_{h_i} - f_{h_j})} \tag{6.5}$$

where f_{h_i} and f_{h_j} are two feature vectors, and S is a covariance matrix. The distance between two feature vectors defined by Equation 6.5 should approximate the cross reconstruction error of the corresponding data blocks. This approximation is achieved by finding a proper covariance matrix S which transforms the feature distance space to approximate the cross reconstruction error space. This transformation is built from data blocks collected from prior simulation runs. Equation 6.6 shows the cost function to be optimized for the covariance matrix calculation.

$$s = \sum_{i=0}^{N-1} \sum_{j=i}^{N-1} (D(f_{h_i}, f_{h_j}) - CE(h_i, h_j))^2 \tag{6.6}$$

where N is the total number of data blocks collected from data of prior simulation runs. The covariance matrix is stored and used to reconstruct high-resolution data.

6.3.3 Data Collection for Prior Knowledge Creation

The prior knowledge is created from the prior simulation runs. To collect data which has good coverage and variation in the parameter space, we use Latin hypercube sampling [22] to sample the parameters and run the simulation to create the prior knowledge. Latin hypercube sampling is often used by domain experts to sample simulation parameters to run a few simulation runs for data analysis, when they cannot output and store results from many simulation runs. The Latin hypercube sampling is similar to the problem of having n rooks (towers) on a high-dimensional chess board without threatening each other. Figure 6.5 gives a 2D example.

When generating data for the prior knowledge creation, the simulation outputs the data at every τ timesteps instead of every time step. Due to data coherence in the temporal domain, the prior knowledge created at a time step can be used to reconstruct data at neighboring time steps. This also reduces the I/O of the prior simulation runs, prior knowledge size, and creation time. To create the prior knowledge, every raw volume of any quantity of interest and any output time step is subdivided into several spatial sub-blocks, called prior knowledge blocks, and each one creates independent prior knowledge. Due to the data coherence in the local spatial domain, mixing data from the whole spatial domain to create a single prior knowledge is not desirable and could lower the quality of the prior knowledge. Note that a prior knowledge block is different from a local data block (h_p) which is represented by a GMM. The whole data spatial domain contains multiple prior knowledge blocks and each prior knowledge block contains multiple local data blocks. For example, if the data spatial domain is divided into L prior knowledge blocks, a domain expert has Q quantities of interest, and T' is the output time steps, our system will create $(L * Q * T')$ independent prior knowledges. Figure 6.5 shows the prior knowledge creation workflow.

To collect the data for prior knowledge creation, we use a "sliding block" with size B^3 to collect sample instances from the prior simulation runs, where B^3 is the predefined down-sampled block size. The sample instances are examined to only keep one instance to represent multiple similar sample instances across multiple prior simulation runs. The similarity is defined by the cross-error of Equation 6.3. This preprocess simplifies the prior

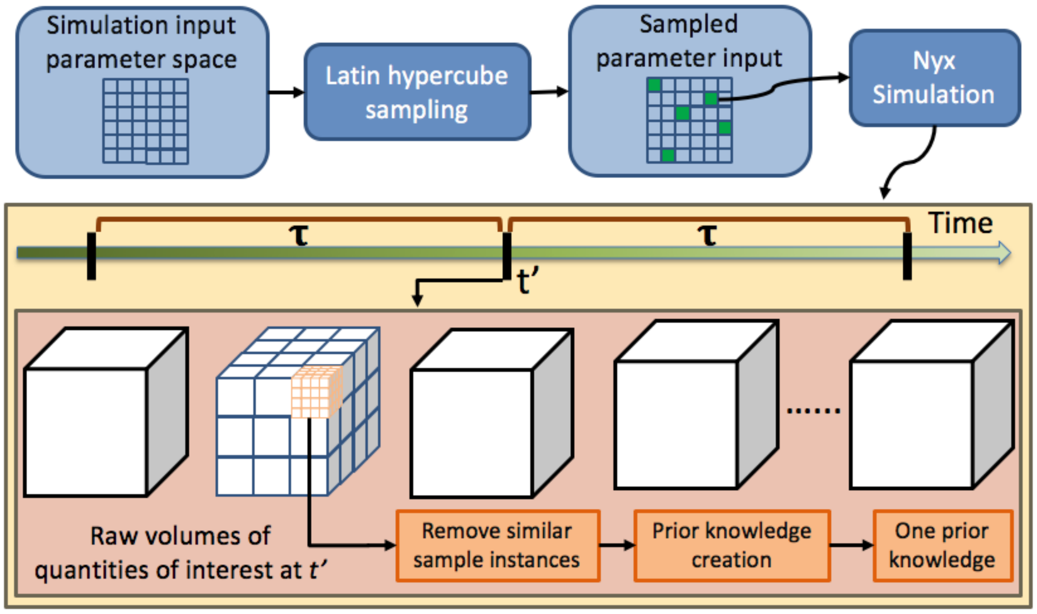


Figure 6.5: We illustrate prior knowledge creation workflow. The chess board at the left upper corner is a 2D example of the simulation parameter input space. The green squares are the result of the Latin hypercube sampling. We illustrate one sampled parameter, which is the input to Nyx, and one time step (t') output, for prior knowledge creation here. The white cubes at the bottom are the output volumes of quantities of interest at time step t'. The blue divisions illustrate the defined prior knowledge blocks. The orange block illustrates using a sliding block to extract sample instances creating one prior knowledge block.

knowledge creation and makes the data reconstruction process in Section 6.4 more efficient, because it shrinks the size of spatial information dictionary.

6.4 Statistical Super-resolution

After creating the prior knowledge from a few simulation runs, remaining simulation runs in the parameter space can be performed with *in situ* statistical down-sampling and later reconstructed with prior knowledge. Only the compact statistical down-sampled data are saved to disk. This does not only reduce the I/O time when writing data from supercomputers and loading data to the post-analysis machine, but also can keep output of more simulation runs on disk for later cosmological data analysis. This section will introduce reconstructing the statistical down-sampled data to high resolution. The step of up-sampling the low-resolution data to high resolution consists of (1) re-sampling data values from GMMs (2) creating the feature vectors from statistical down-sampled data and finding the best matching location information in prior knowledge, and (3) using the re-sampled data and the location information to reconstruct the high resolution data.

6.4.1 GMM Data Re-sampling

Each point in low resolution is represented by a GMM, and the GMM is computed from the data values within a B^3 data block in high resolution. The data values in a high-resolution data block can be reconstructed by drawing B^3 data values from the GMM. The procedure to draw a sample from a GMM has two steps. The first step is to select a Gaussian component from the GMM according to the weight associated with each Gaussian component. The second step is to draw a value from the selected Gaussian distribution. By repeating the sampling step B^3 times, the data values in the high-resolution block are reconstructed. In addition, the domain of a Gaussian distribution is defined in an infinite value space, so unnecessary values, though associated with extremely small probability in the Gaussian distribution, still have chances to be drawn. To avoid drawing these unnecessary values, we only accept values that are within 3 standard deviations of the mean, and between the stored minimum and maximum values of the data block. Otherwise, the sample is rejected and re-drawn from the Gaussian distribution. These re-sampling values do not have the location information to allocate them in the high-resolution spatial space yet.

6.4.2 Location Information Query

To query location information for the reconstruction of a low-resolution point, the first step is to compute the feature vector of the point. In Section 6.3.1, we introduced that the feature vector of a high-resolution data block consists of means and standard deviations of itself and those of the neighboring data blocks. In contrast to the spatial information dictionary creation in Section 6.3.1, the high-resolution data blocks are not available here. Therefore, we have to derive means and standard deviations from GMMs. Equation 6.7 and 6.8 show the mean and standard deviation calculation of a GMM.

$$\mu_{g_p} = \sum_{k=0}^{K-1} (w_i * \mu_i) \tag{6.7}$$

$$\sigma_{g_p} = \sqrt{\sum_{k=0}^{K-1} (w_i * \mu_i^2) + \sum_{k=0}^{K-1} (w_i * \sigma_i^2) - \mu_{g_p}^2} \tag{6.8}$$

where μ_{g_p} and σ_{g_p} are the mean and standard deviation of the GMM, g_p, at a low resolution point p. K is the number of Gaussian components of g_p. w_i, μ_i and σ_i are the i^{th} weight, mean and covariance matrix of g_p, respectively. After computing the mean and standard deviation of a GMM, the feature vector of a point can be computed by Equation 6.2. If the feature vector computation requests the mean and standard deviation from a point that is out of the data spatial domain, the mean and standard deviation are set to 0.

To query the location information for reconstruction of a low-resolution point with quantity q at time step t, we first look up the prior knowledge whose block spatial domain contains the point, and time step is the closest to t and quantity is q. The feature vector f_p computed from the point p is used to query the location information s' which is associated with the feature vector f' in the dictionary. f' is the feature vector which is closest to f_p in the spatial location dictionary. The distance between feature vectors was introduced in Section 6.3.2 and the covariance matrix S learned from data are involved to find the closest feature vector in the spatial location dictionary. Equation 6.9 shows how to find the feature vector f'.

$$f' =_f \sqrt{(f - f_p) * S^{-1} * (f - f_p)} \tag{6.9}$$

After sampled values and location information (s') are known, they are used to reconstruct the high-resolution data block.

6.4.3 High Resolution Data Reconstruction

Reconstruction uses the location information function (s'), which maps samples' order defined by samples' value among a group of samples to location indexes, to assign the samples to the spatial domain. We first sort the samples drawn from GMM according to the data values to get an order of samples. Then using the location information (s'), we map each data sample to the spatial domain. Algorithm 6 summarizes the steps to reconstruct high-resolution data from a low-resolution grid point. Line 6 in Algorithm 6 allocates a buffer with size B^3 for high-resolution data reconstruction. Line 7 selects the prior knowledge according to the time step, quantity and spatial coordinate of the reconstructing grid point. The details of Line 7 are in Section 6.4.2. Line 8 and Line 9 generate the feature vector for p and find location information from the prior knowledge. Line 10 retrieves the GMM at location p. In Line 11, u_p is the collection of samples which are re-sampled from g_p. Line 12 sorts data in u_p and returns the sorted array to u'_p. The loop between Line 13 and 17 goes through the re-sampled values in order, and places data into the high-resolution data block buffer following the location information. Since the reconstruction of each low-resolution grid point is independent, we have a portable parallel statistical super-resolution implementation based on the VTK-m framework [65]. This implementation can increase the speed of super-resolution on either GPU (CUDA) or multicore CPU (Intel TBB) backend according to the available resource on the post-analysis machine by changing the compiler configuration, and without changing the code.

Algorithm 6 Statistical Super-resolution

```
1: ▷ $L_{t,q}$: a statistical down-sampled data at time $t$ with quantity $q$
2: ▷ $p$: the low resolution coordinate of a point for reconstruction
3: ▷ $PK$: collection of all prior knowledge
4: ▷ $B$: pre-defined down-sampled data block size
5: procedure StatisticalSR($L_{t,q}$, $p$, $PK$, $B$)
6:     $h_p$ = AllocateBuffer($B^3$)
7:     $pk$ = PriorKnowledgeSelector($PK$, $t$, $q$, $p$)
8:     $f_p$ = FeatureVectorGenerator( $L_{t,q}$, $p$ )
9:     $s'$ = LocationInformationQuery($pk$, $f_p$)
10:    $g_p = L_{t,q}(p)$
11:    $u_p$ = GmmResampler($g_p$, $B^3$)
12:    $u'_p$ = Sorting($u_p$)
13:    for $i = 1 \rightarrow B^3$ do
14:        $\ell = s'(i)$
15:        $v = u'_p[i]$
16:        SetValueToLocation($h_p$, $v$, $\ell$)
17:    end for
18:    return $h_p$
19: end procedure
```

6.5 Evaluation

To carry out the evaluation, the simulation parameter space consists of three parameters of interest: hubble constant (*comoving_h*), the total density of baryons (*comoving_OmB*), and the total matter density (*comoving_OmM*). Due to the cosmologists' efforts in the past decades, they have shrunk the possible value ranges of these parameters to smaller intervals. The domain ranges of interest are $0.55 \leq comoving\_h \leq 0.85$, $0.0215 \leq comoving\_OmB \leq 0.0235$ and $0.12 \leq comoving\_OmM \leq 0.155$. 10 samples are regularly taken in each dimension of interest across the simulation input parameter space which results in 1000 possible simulation inputs. Each simulation run can have a different number of time steps according to the convergence speed of each simulation input. In this experiment, we use the first 200 time steps from each simulation run to carry out this evaluation, because all simulation runs have at least 200 time steps.

We evaluate our technique on 7 different output quantities: density of dark matter particles (*density*), X-momentum (*xmom*), Y-momentum (*ymom*), Z-momentum (*zmom*), internal energy of the gas (*rho_e*), temperature (*Temp*) and gravitational potential (*phi_grav*). Each output data is a regular grid with a resolution of 256^3 grid points. The prior knowledge is created from only 5 simulation runs, which is sampled from the simulation parameter input space by latin hypercube sampling. In addition, we create the prior knowledge at every 10 time steps and set the prior knowledge block size to 64^3. For the feature vector creation, the surrounding grid points (Ω in Equation 6.2) of a given grid point at (i,j,k) are 7 points which includes (i,j,k), $(i+1,j,k)$, $(i-1,j,k)$, $(i,j+1,k)$, $(i,j-1,k)$, $(i,j,k+1)$ and $(i,j,k-1)$. The *in situ* statistical down-sampling for all other simulations only uses 25% of grid points, in each 16^3 data block, to compute a GMM with 5 Gaussian components. Each simulation run is carried out on a node of supercomputer with two 14-core Intel (Broadwell) Xeon E5-2680 v4 processors and 128 GB DDR3 Memory.

6.5.1 Quantitative Evaluation

Storage

Our technique outputs the prior knowledge and the statistical down-sampled data for all output quantities, time step and simulation runs. The ratio of the statistical down-sampled data size to the raw data size can be computed by Equation 6.10.

$$Z_{SDS} = \frac{G * 3 + 2}{B^3} \tag{6.10}$$

where $G * 3$ means each GMM has G Gaussian components. Each Gaussian component is represented by a weight, a mean and a standard deviation. 2 represents the stored minimum and maximum values of a down-sampled data block. B is the down-sampled data block resolution. In our experiment, we use a GMM with 5 Gaussian components to represent a 16^3 data block. The storage consumption of the statistical down-sampled data is only 0.42% of the raw data. Our prior knowledge for all 7 quantities is generated at every tenth times step. The total size of the prior knowledge is 25GB. If we store raw data for 1000 simulation runs with 200 time steps and 7 quantities, it requires 85 TB storage. In contrast, our approach only consumes 388 GB (25 GB for prior knowledge and 363 GB for all statistical down-sampled data), which is a total of 0.44% of the raw data.

Error of Reconstruction

We compare the reconstruction error with 4 other different approaches. The first is a naive down-sampling, which uses the average value to represent a local data block and reconstructs data by interpolation. The second is statistical down-sampling, without the prior knowledge. It reconstructs data by re-sampling data from a GMM, and randomly placing samples to a data block. The third and fourth approaches are scientific data lossy compression techniques, “ISABELA” [49] and “SZ” [21].

To achieve a fair comparison on reconstruction error, we adjust the parameters of these four approaches to generate data proxies whose size are as close as possible to our approach. Comparing with raw data size, our approach outputs 0.44%. For naive down-sampling approach, each 6^3 data block is represented by one average value and this consumes 0.46% of the raw size. The statistical down-sampling without prior knowledge also down-samples a 16^3 data block to a GMM and consumes 0.42% of the raw size. We adjust the error tolerance and the number of stored coefficients to minimize the size from ISABELA. The smallest size of compressed data we can achieve is approximately 24% of the raw data, around 55 times more than our approach. For SZ compression, we adjust the PSNR error bound to 12 dB to generate a compressed data with an average around 2.44% of the raw size.

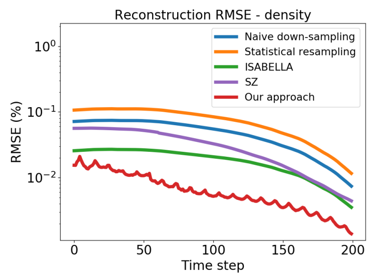
We randomly draw 250 simulation parameter inputs to run the simulation and compare the error of the ground truth to the reconstructed data in each case. The RMSE of one reconstructed volume is calculated by Equation 6.4, where h and r are the whole raw volume and reconstructed volume, respectively. $Max(h)$ and $Min(h)$ are the maximum and minimum values of the raw volume. Figure 6.6 shows the average Root Mean Square Error (RMSE) of the reconstructed data among 250 simulation runs. Since we create the prior knowledge at every 10 times steps and use the closest prior knowledge in the temporal domain to reconstruct data, the error curves for our approach have periodic bumps. Our approach has the best reconstruction RSME, except for several time steps of the *Temp* quantity. By examining the data, we found the simulated universe initially has a uniform temperature over the whole space. Therefore, the denominator of Equation 6.4 is extremely small, greatly amplifying errors due to the re-sampling process from GMM. In general, our approach shows lower reconstruction error than all other approaches for all 7 quantities.

We used a domain tool, Gimlet [33], to compute the power spectrum from the raw data and reconstructed data. The power spectrum of the density fluctuations is a frequently used cosmological statistical measure. The error of the power spectrum computed from raw data and reconstructed data is plotted in Figure 6.6(h). The error from our technique is lower than other alternatives.

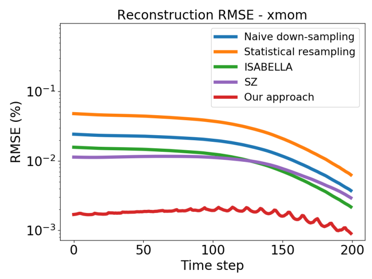
Performance

The computation time of the *in situ* statistical down-sampling consists of two parts, time for the EM algorithm to compute GMMs and time for writing down-sampled data to disk. The average time to write the down-sampled data for one time step to disk is 0.8 seconds among all time steps and simulation runs. In contrast, the time to write full resolution raw data to disk is 11.2 seconds. The statistical down-sampling runs the EM algorithm to compute GMMs and the average time to compute GMMs for one step is 9.4 seconds. The time to compute GMMs at different time steps is not constant. Because the data values usually have higher variation in later time steps in the Nyx simulation, the EM algorithm requires more time to converge for the later time steps. Figure 6.6(i) shows the time of *in situ* down-sampling over time steps. The average of the *in situ* statistical down-sampling time per simulated time step is 10.19 seconds. Therefore, our approach significantly reduces the stored data size without spending more supercomputer time and reduces the data movement cost to post-analysis computers.

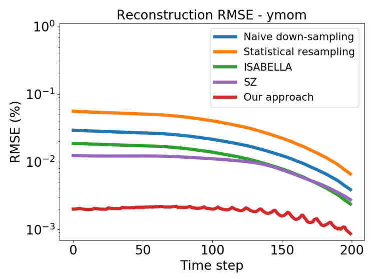
In this experiment, we created 64 * 7 * 21 prior knowledge data sets, because we have 64 prior knowledge blocks per volume, 7 quantities, and 21 time steps for prior knowledge creation. We subdivided the prior knowledge creation tasks into 140 non-overlapping sets,



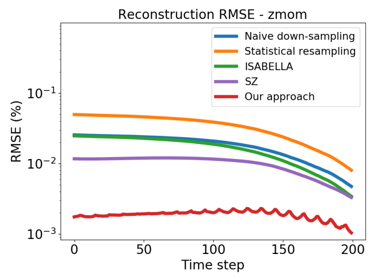
(a)



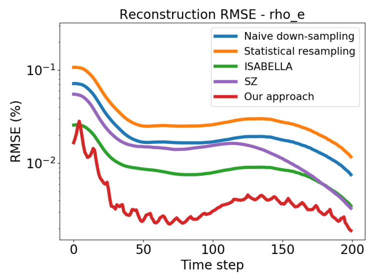
(b)



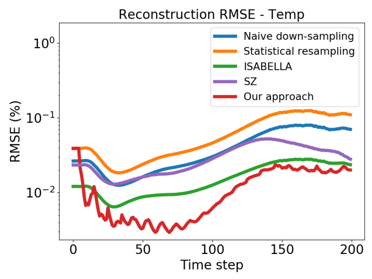
(c)



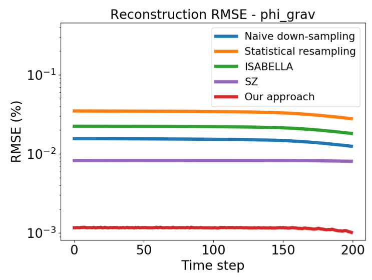
(d)



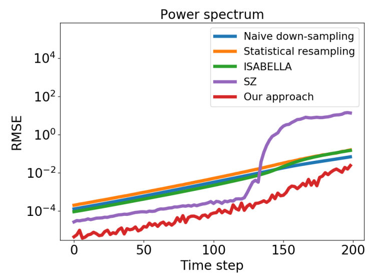
(e)



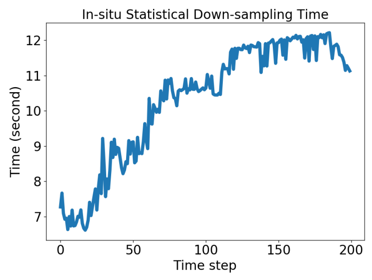
(f)



(g)



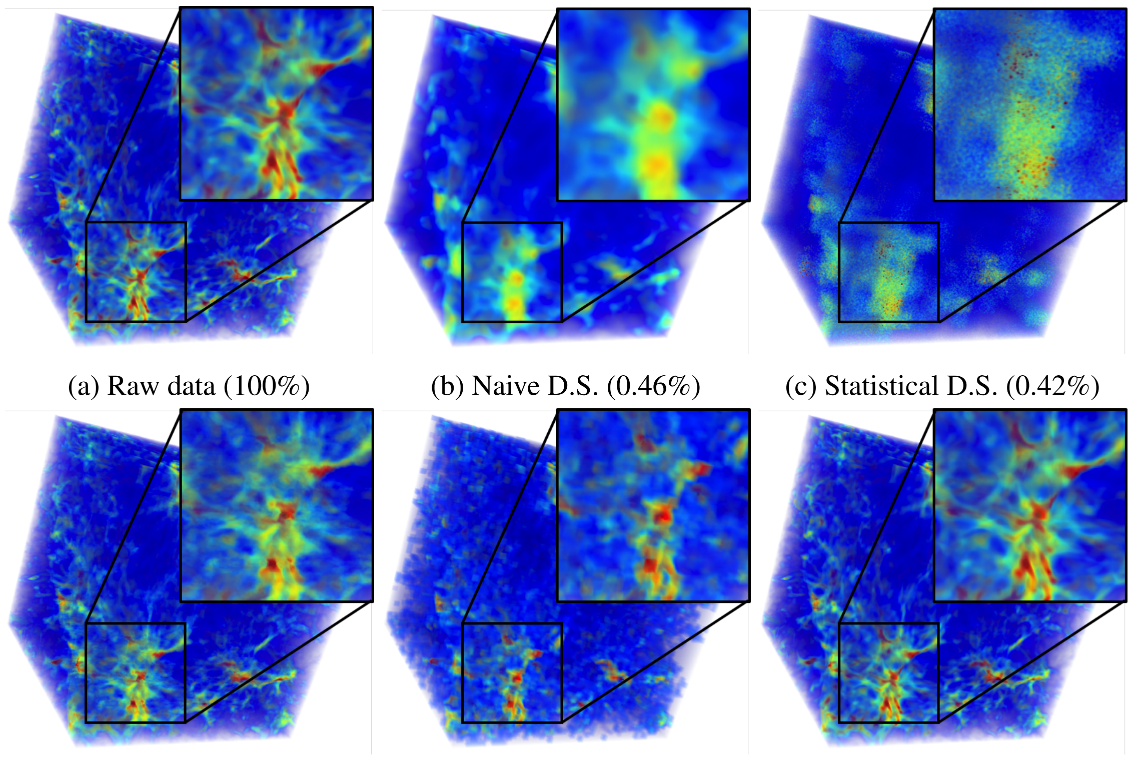
(h)



(i)

Figure 6.6: (a) - (g) are average reconstruction RMSE of 7 quantities over 250 simulation runs. (h) is the average error of the power spectrum. (i) is the average *in situ* down-sampling time. X-axis is the time steps. Y-axis in (a) - (h) is the RMSE in log scale.

where each set takes 1.37 hours on average. Note that the prior knowledge creation is a one-time task from the 5 simulation runs. Once the prior knowledge is generated, it can be used for reconstructing down-sampled data of all subsequent simulation runs. The wall time can be reduced by subdividing the tasks into more sets if more computational resources are available. The time to reconstruct a statistical down-sampled volume to full resolution


(a) Raw data (100%) (b) Naive D.S. (0.46%) (c) Statistical D.S. (0.42%)

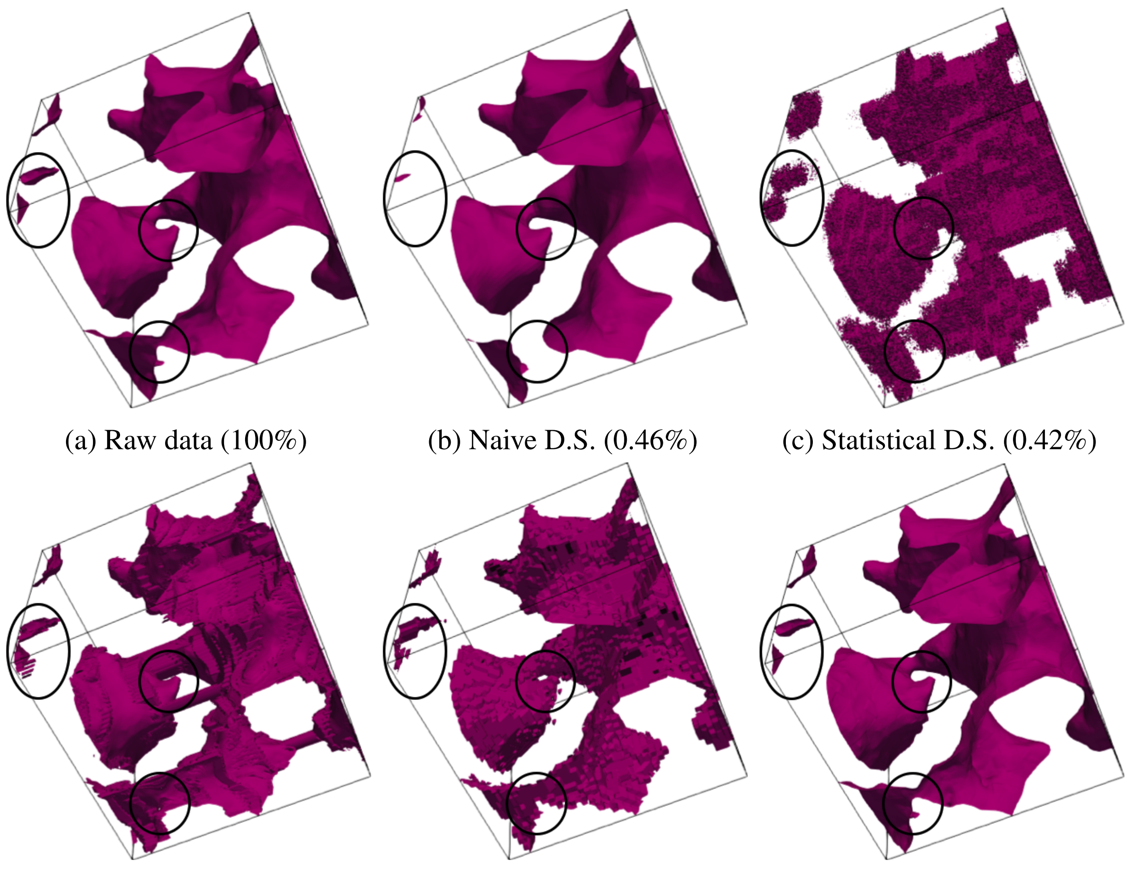
(d) ISABELA (24%) (e) SZ (2.44%) (f) Our approach (0.44%)

Figure 6.7: We display volume rendering of *density* quantity at time step 180 from a simulation run with input parameters, *comoving_h = 0.61666*, *comoving_OmB = 0.02216* and *comoving_OmM = 0.14328*. D.S. stands for down-sampling. The numbers in sub-figure captions are the ratio of the storage consumption to the raw data size.

is 1.28 seconds, using an Intel 8-Core i-7-4770 3.40GHz CPU with 16GB memory. Each local data block reconstruction is an independent task, and the time can be further reduced if more computational resources are available.

6.5.2 Qualitative Evaluation

We use direct volume rendering, isosurfaces, streamlines, and Gimlet [33] to show the quality of reconstruction. Figure 6.7 shows rendering images of the reconstructed data for



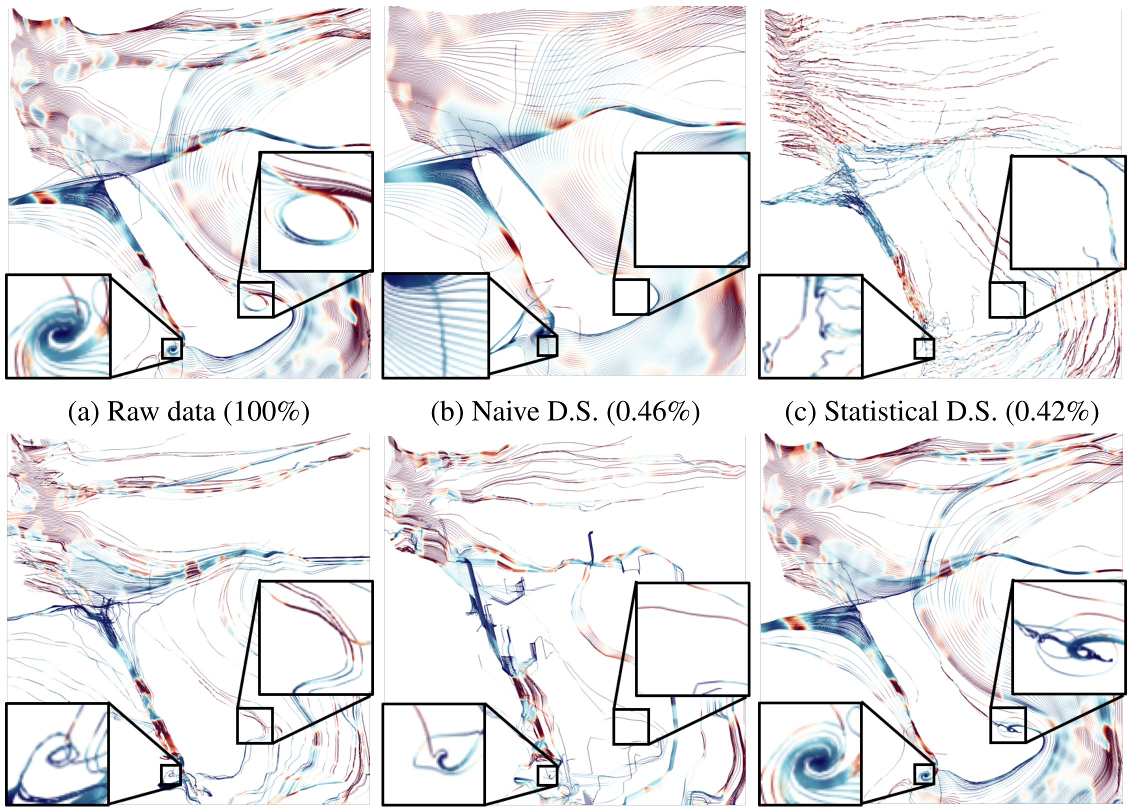
(a) Raw data (100%) (b) Naive D.S. (0.46%) (c) Statistical D.S. (0.42%)

(d) ISABELA (24%) (e) SZ (2.44%) (f) Our approach (0.44%)

Figure 6.8: We display isosurface of *phi_grav*=0 at time step 153 from a simulation run with input parameters, *comoving_h = 0.55*, *comoving_OmB = 0.02326* and *comoving_OmM = 0.1394*. D.S. stands for down-sampling. The numbers in sub-figure captions are the ratio of the storage consumption to the raw data size.

density using one selected time step and input parameters. Our approach shows the most similar image compared to the raw data.

Figure 6.8 shows an isosurface from the reconstructed data of *phi_grav* quantity. Our approach most accurately preserves the shape of the isosurface. We circle several regions where other alternatives show incorrect shapes, such as the pieces at the left upper corner, the connection at the bottom left corner, and the hold at the middle.

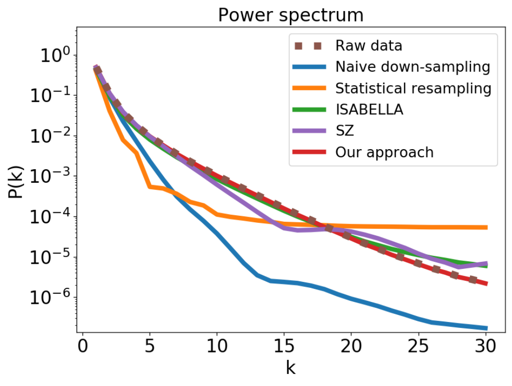


(a) Raw data (100%) (b) Naive D.S. (0.46%) (c) Statistical D.S. (0.42%)

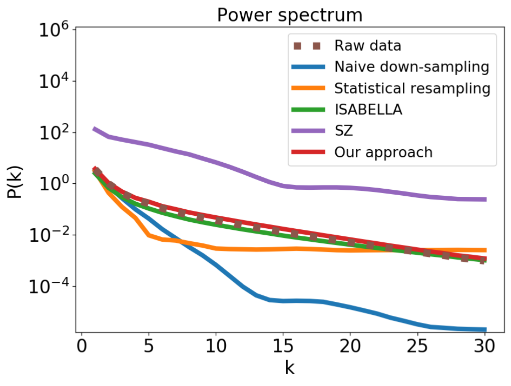
(d) ISABELA (24%) (e) SZ (2.44%) (f) Our approach (0.44%)

Figure 6.9: We display streamlines of the vector field from *xmom*, *ymom* and *zmom* quantities at time step 200 with input parameters, *comoving_h = 0.58333*, *comoving_OmB = 0.02304*, and *comoving_OmM = 0.12776*. 250 seeds are put on the line between [0,0,0] and [255,255,255] to compute streamlines. We zoom-in to sub-domains of the data to show the details. The red and blue color on the streamlines indicate the higher and lower vector magnitudes, respectively. D.S. stands for down-sampling. The percentage in the sub-figure captions are the ratio of the storage consumption to the raw data size.

Figure 6.9 shows streamlines using the vector field from *xmom*, *ymom*, and *zmom* quantities. We highlight and zoom-in two regions, a vortex and a swirl. Our approach reproduces the left vortex. Naive down-sampling also reproduces it, but the vortex shifts in space. All methods cannot reproduce the right swirl, but our approach approximates this feature the best.



(a)



(b)

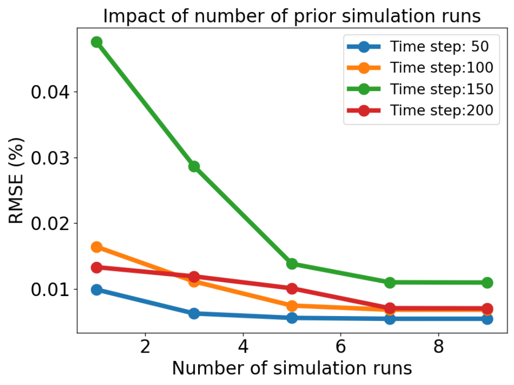
Figure 6.10: (a) and (b) are power spectra computed from data at time step 200 with input parameters, *comoving_h* = *0.84997*, *comoving_OmB* = *0.02304* and *comoving_OmM* = *0.13552*, and *comoving_h* = *0.61666*, *comoving_OmB* = *0.02216* and *comoving_OmM* = *0.14328*, respectively.

Figure 6.10 shows the power spectrum at time step 200 for two simulation runs. We observe that the power spectrum computed from our approach is the most similar to the raw data. A low error power spectrum is also computed from reconstructed data using ISABELA, but ISABELA requires 55 times storage than our approach.

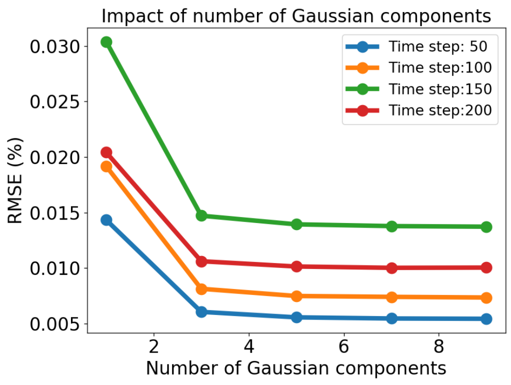
6.6 Discussion

6.6.1 Parameter Turning

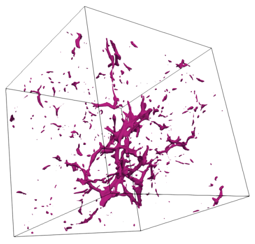
In our system, several parameters have to be selected. They are the number of Gaussian components of each GMM, the number of prior simulation runs, the size of a down-sampled block, and the number of samples in each local data block for the EM algorithm. The size of a down-sampled block is mainly determined by the affordable size of statistical down-sampled data. The selection of the number of samples in each local data block for the EM algorithm determines *in situ* time. A very small number simulation runs with sparse *in situ* calls in the temporal domain can be used to determine the number of sub-sampling data samples to run the EM algorithm. The number of prior simulation runs impacts the reconstruction quality. To study the reconstruction quality, we compared the reconstruction results when parameters are changed. Keeping all high-resolution data for this study is not feasible. Therefore, data generated from fewer qualities, and smaller spatial and temporal resolution are used to study and select the parameters. We have run this study by 64^3 resolution, 3 quantities (*"density"*, *"zmom"* and *"Temp"*), 4 time steps (50, 100, 150, 200), and a down-sampled block size 8^3. The raw data of all 1000 simulation runs, defined in the simulation parameter space in Section 6.5, are saved for this study using 12 GB. We use Latin hypercube sampling to sample 1, 3, 5, 7, 9 simulation runs for prior knowledge creation. Figure 6.11(a) shows the normalized RMSE. When only one simulation run is used to create the prior knowledge, the reconstruction error is high. The error decreases when the number of simulation runs for prior knowledge creation increases. We see diminishing returns when the number of simulation runs is more than 5. In addition, we also varied the number of Gaussian components to study the impact on reconstruction quality. In Figure 6.11(b), we observe similar diminishing returns if the number of Gaussian components is greater than 4. Figure 6.11(c)-(f) show the quality of reconstructed data with fewer Gaussian components and prior simulation runs. With fewer prior simulation runs, the GMM samples are more likely to be assigned incorrect locations. If fewer Gaussian components are used, the GMM may not accurately summarize the data, and a large error may be introduced to re-sampled data.



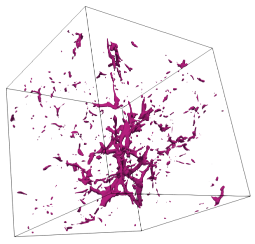
(a)



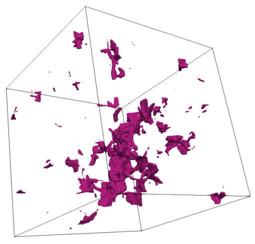
(b)



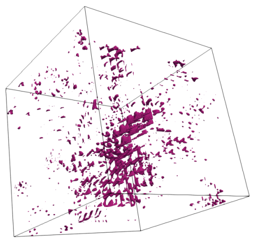
(c)



(d)



(e)



(f)

Figure 6.11: We show the impact of the reconstruction quality using different numbers of simulation runs for prior knowledge creation and a different number of Gaussian components. Every point on each curve is the average RMSE of quantities, *"density"*, *"zmom"* and *"Temp"*. (c) is an isosurface from raw data. (d), (e) and (f) are isosurfaces from the reconstructed data with 5 prior simulation runs and 5 Gaussian components, 5 prior simulation runs and 1 Gaussian component, and 1 prior simulation run and 5 Gaussian components, respectively.

6.6.2 Data Variation in Parameter Space of Nyx

We will discuss the variation produced from Nyx simulation. Density, at time step 200, is used as an example in this discussion. We examine all simulation runs in Section 6.5 and find the maximum density value, $3.8 * 10^{13}$, and the minimum value in the same simulation run is $4.8 * 10^{8}$. For the smallest valued simulation, the maximum density value is $1.2 * 10^{11}$. So, Nyx can produce datasets where the maximum value of a simulation run is 300 times larger than that of another. In addition, the standard deviation of the maximal values among simulation runs is $1.7 * 10^{13}$. We show how data values change among different simulation runs in Figure 6.12(a). We compared values along raycast segments from simulation runs as an example. The ray is cast from the center of the x-y plane with z=0 to the center of the x-y plane with z=255 on the density quantity of 5 prior simulation runs in Section 6.5 and 20 randomly sampled test simulation runs to collect data values on the ray and plot. It shows that data values from different simulation runs have large differences and the trends of data value change among simulation runs are also different. For example, the curves circled by the purple ellipse have a concave in the region, but the curves circled by the red and green ellipses are peaks. On the other hand, we also observe that multiple curves may have similar trends of data value change in a local region, but these curves still have quite different data values. In our approach, GMM accurately captures the data values without keeping location information. The trends of value change from the prior knowledge, the colored solid lines in Figure 6.12(a), is used to place values into space. We show the isosurfaces from a prior simulation run and a test simulation run, which are marked in Figure 6.12(a). Although they have relatively close curves in Figure 6.12(a), the isosurfaces in Figure 6.12 (b) and (c) are significantly different.

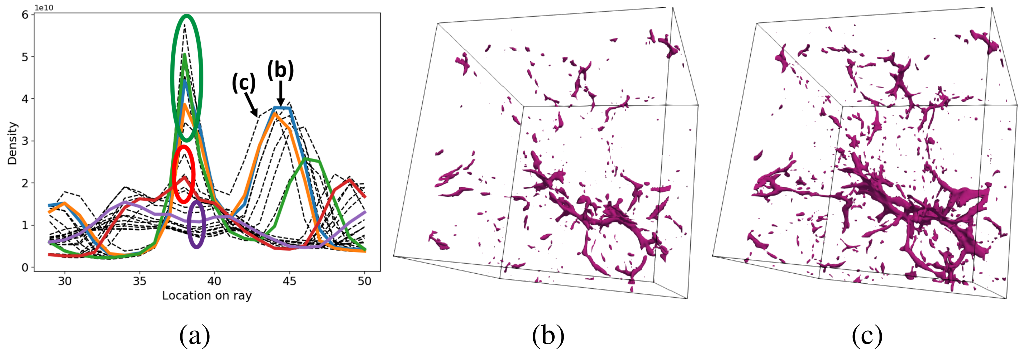


Figure 6.12: (a) Data values on raycast segments from 5 prior simulation runs and 20 randomly sampled test simulation runs. Data values on rays from prior simulation runs and sampled test simulation runs are plotted by colored solid lines and black dashed lines, respectively. Two simulation runs, a prior simulation run marked by (b) and a test simulation run marked by (c), show isosurfaces with isovalue 10^{10} in (b) and (c), respectively.

6.6.3 Domain Expert Feedback

We have interviewed a cosmologist, who is one of the authors of Nyx simulation, and summarized his comments. We discuss the applicability of our technique in their data analysis workflow. Using the setting in Section 6.5, it requires approximately 2 hours to finish one simulation. As we have noted before, it is not feasible to retain data from the high-resolution simulations permanently, due to the size. Re-running the simulation every time cosmologists need a data will require waiting hours to continue their analysis tasks. Our technique does not significantly reduce the overall simulation time, but it provides a high data reduction rate and reconstruction quality to retain data. Thus, the cosmologists are able to keep simulation runs and access data without re-running the simulation. The domain expert also commented, These simulations are often very expensive, so you don't want to run and re-run them every time you want to do some analysis. The cosmologist commented that the visual quality looks good and it's good that our method has the lowest RMSE.

6.7 Conclusion and Future Work

This paper presents a novel *in situ* technique for cosmological data analysis and visualization. The data from a few simulation runs are selected by Latin hypercube sampling and used to create prior knowledge which captures the relation between low- and high-resolution. Data from simulation runs with other simulation parameter inputs are down-sampled *in situ* to reduce the requirements of I/O bandwidth and disk storage. We implement data parallel statistical down-sampling and data sub-sampling to remain the simulation time of new workflow at the same scale of the traditional workflow. Because our approach hugely reduces the data from the cosmological simulation, the data from more simulation runs is affordable to save for data analysis. We qualitatively and quantitatively demonstrate that our technique outperforms other alternative approaches. In future work, we will explore using error quantification and machine learning technique. To quantify the point-wise error and convey that to scientists are valuable because knowing the error range in the visualization could change scientists' decision and avoid that the data analysis task is misled by the region with a higher error. Machine learning techniques, such as neural network based approaches, could improve the prior knowledge to capture more precise feature shapes and allow our technique to be used on more diverse types of simulations.

Chapter 7: Conclusion and Future Works

7.1 Conclusion

In this dissertation, we have demonstrated different types of distribution-based techniques and their effectiveness in large dataset analysis and visualization. To tackle the problem of big data volume rendering, we proposed a ray- and distribution-based approach which collects the samples projected to each image pixel and compactly summarizes them by distributions. When rendering the dataset from the proxy, a proposed importance distribution computed from the ray-based distribution and the user-given transfer function are used to draw small number of samples and render high quality images. In chapter 4, we focus on the problem of the spatial information loss in the traditional distribution representation when it is used to tackle the big data problem caused by high spatial resolution. Spatial Gaussian Mixture Model corresponding to each defined data value interval stores the spatial location distribution of samples in the data value interval. The traditional distribution representation is augmented by the proposed spatial GMMs to reconstruct the data value at arbitrary location with a narrow uncertainty. This data representation can not only be applied to various visualization algorithms but also provide high quality visualization. We also develop a view-dependent technique to tackle the big data problem caused by high temporal resolution. A pixel ray at a non-sampled time step is decoupled into a value distribution and their location information. We either directly store the samples' location information or utilize the data coherence in the temporal domain to represent the location information. We demonstrate that our data proxy spends small storage size on the non-sampled time steps but reconstructs evolution of features with high quality. For ensemble dataset, we propose a compact data proxy which utilizes the data coherence in the simulation parameter space. The data coherence is used to predict the data samples' locations when we reconstruct a statistical down-sampled data to high resolution. We show that our data proxy can reach a high data reduction rate and a small reconstruction error.

7.2 Future Research Directions

In this section, we discuss the potential future directions which can be extended from the research works in this proposal. Current research works use the distribution representation to compactly store the high resolution datasets in spatial domains. We can extend our current in-situ workflow to in-transit workflow [9, 33]. The in-situ workflow directly halts the simulation and runs the analysis to produce the compact data proxies after the simulation run of every time step. The advantage of in-situ worklet is that the implementation is easier, but the disadvantage is that the sumpercomputer resource could be wasted if the analysis part does not need the same scale of computational resource as the simulation. In-transit workflow allows the simulation and analysis to run simultaneously although it usually needs a more invasive code modification in the simulation code. A technique should be developed to predict the need of the computational resource for the analysis part. If the computational resource requirement of the analysis part is low, we can switch the workflow to in-transit and utilize the resource better. Otherwise, we can keep using the in-situ workflow. In addition, machine learning algorithms have the potential to tackle big data problems of ensemble datasets in scientific visualization. For example, GAN (Generative Adversarial Networks) [37] is often used to train a model to capture the distribution of the training dataset. The model can take input parameters to reproduce different results. In scientific visualization and analysis scenarios, GAN has the potential to capture relation among ensemble members and take new simulation parameters to produce other ensemble members later.

Bibliography

[1] Color map webpage of data science at scale of los alamos national laboratory. https://datascience.lanl.gov/colormaps.html. Accessed: 2017-09-25.

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

[3] James Ahrens, John Patchett, Andrew Bauer, Sébastien Jourdain, David H Rogers, Mark Petersen, Benjamin Boeckel, Patrick OLeary, Patricia Fasel, and Francesca Samsel. In situ mpas-ocean image-based visualization. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Visualization & Data Analytics Showcase*, 2014.

[4] Theodore T Allen. *Introduction to engineering statistics and six sigma: statistical quality control and design of experiments and systems*. Springer Science & Business Media, 2006.

[5] Ann S Almgren, John B Bell, Mike J Lijewski, Zarija Lukić, and Ethan Van Andel. Nyx: A massively parallel amr code for computational cosmology. *The Astrophysical Journal*, 765(1):39, 2013.

[6] Tushar Athawale and Alireza Entezari. Uncertainty quantification in linear interpolation for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2723–2732, 2013.

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

[8] Andrew C Bauer, Hasan Abbasi, James Ahrens, Hank Childs, Berk Geveci, Scott Klasky, Kenneth Moreland, Patrick O'Leary, Venkatram Vishwanath, Brad Whitlock, et al. In situ methods, infrastructures, and applications on high performance computing platforms. In *Computer Graphics Forum*, volume 35, pages 577–597. Wiley Online Library, 2016.

[9] Janine C Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, Attila Gyulassy, Tong Jin, Scott Klasky, Hemanth Kolla, Manish Parashar, Valerio Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 49. IEEE Computer Society Press, 2012.

[10] Jeff A Bilmes et al. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[11] Alécio Pedro Delazari Binotto, Joao LD Comba, and Carla MD Freitas. Real-time volume rendering of time-varying data using a fragment-shader compression approach. In *Parallel and Large-Data Visualization and Graphics, 2003. PVG 2003. IEEE Symposium on*, pages 69–75. IEEE, 2003.

[12] Udeepta D Bordoloi and H-W Shen. View selection for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 487–494. IEEE, 2005.

[13] Lars Buitinck et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[14] Steven P Callahan, Jason H Callahan, Carlos E Scheidegger, and TC Silva. Direct volume rendering. *Comput Sci Eng*, 1:88–92, 2008.

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

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

[17] Chun-Ming Chen, Ayan Biswas, and Han-Wei Shen. Uncertainty modeling and error reduction for pathline computation in time-varying flow fields. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 215–222. IEEE, 2015.

[18] Jae-jeong Choi and Yeong Gil Shin. Efficient image-based rendering of volume data. In *Computer Graphics and Applications, 1998. Pacific Graphics' 98. Sixth Pacific Conference on*, pages 70–78. IEEE, 1998.

[19] Carlos D Correa and Kwan-Liu Ma. Visibility-driven transfer functions. In *Visualization Symposium, 2009. PacificVis' 09. IEEE Pacific*, pages 177–184. IEEE, 2009.

[20] Marc Davis, George Efstathiou, Carlos S Frenk, and Simon DM White. The evolution of large-scale structure in a universe dominated by cold dark matter. *The Astrophysical Journal*, 292:371–394, 1985.

[21] Sheng Di and Franck Cappello. Fast error-bounded lossy hpc data compression with sz. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 730–739. IEEE, 2016.

[22] D Donovan, K Burrage, P Burrage, TA McCourt, B Thompson, and EŞ Yazici. Estimates of the coverage of parameter space by latin hypercube and orthogonal array-based sampling. *Applied Mathematical Modelling*, 57:553–564, 2018.

[23] Soumya Dutta, Chun-Ming Chen, Gregory Heinlein, Han-Wei Shen, and Jen-Ping Chen. In situ distribution guided analysis and visualization of transonic jet engine simulations. *IEEE transactions on visualization and computer graphics*, 23(1):811–820, 2017.

[24] Soumya Dutta, Chen Chun-Ming, Gregory Hernlein, Han-Wei Shen, and Jenping Chen. In situ distribution guided analysis and visualization of transonic jet engine simulations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):837–846, 2016.

[25] Soumya Dutta, Han-Wei Shen, and Jen-Ping Chen. In situ prediction driven feature analysis in jet engine simulations. In *Pacific Visualization Symposium (PacificVis), 2018 IEEE*, pages 66–75. IEEE, 2018.

[26] Soumya Dutta, Jonathan Woodring, Han-Wei Shen, Jen-Ping Chen, and James Ahrens. Homogeneity guided probabilistic data summaries for analysis and visualization of large-scale data sets. In *Pacific Visualization Symposium (PacificVis), 2017 IEEE*, pages 111–120. IEEE, 2017.

[27] Nathan Fabian, Kenneth Moreland, David Thompson, Andrew C Bauer, Pat Marion, Berk Gevecik, Michel Rasquin, and Kenneth E Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96. IEEE, 2011.

[28] Oliver Fernandes, Steffen Frey, Filip Sadlo, and Thomas Ertl. Space-time volumetric depth images for in-situ visualization. In *Large Data Analysis and Visualization (LDAV)*, pages 59–65. IEEE, 2014.

[29] Nathaniel Fout and Kwan-Liu Ma. Transform coding for hardware-accelerated volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1600–1607, 2007.

[30] A Francos, Francisco Javier Elorza, Fayçal Bouraoui, Giovanni Bidoglio, and Lorenzo Galbiati. Sensitivity analysis of distributed environmental simulation models: understanding the model behaviour in hydrological studies at the catchment scale. *Reliability Engineering & System Safety*, 79(2):205–218, 2003.

[31] Steffen Frey and Thomas Ertl. Auto-tuning intermediate representations for in situ visualization. In *Scientific Data Summit (NYSDS), 2016 New York*, pages 1–10. IEEE, 2016.

[32] Steffen Frey, Filip Sadlo, and Thomas Ertl. Explorable volumetric depth images from raycasting. In *Graphics, Patterns and Images (SIBGRAPI), 2013 26th SIBGRAPI Conference on*, pages 123–130. IEEE, 2013.

[33] Brian Friesen, Ann Almgren, Zarija Lukić, Gunther Weber, Dmitriy Morozov, Vincent Beckner, and Marcus Day. In situ and in-transit analysis of cosmological simulations. *Computational Astrophysics and Cosmology*, 3(1):4, 2016.

[34] Jinzhu Gao, Han-Wei Shen, Jian Huang, and James Arthur Kohl. Visibility culling for time-varying volume rendering using temporal occlusion coherence. In *Proceedings of the conference on Visualization'04*, pages 147–154. IEEE Computer Society, 2004.

[35] Sanjay Garg. Aircraft turbine engine control research at nasa glenn research center. *Journal of Aerospace Engineering*, 26(2):422–438, 2013.

[36] Enrico Gobbetti, José Antonio Iglesias Guitián, and Fabio Marton. Covra: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. In *Computer Graphics Forum*, volume 31, pages 1315–1324. Wiley Online Library, 2012.

[37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[39] Stefan Guthe and Wolfgang Straßer. Real-time decompression and visualization of animated volume data. In *Proceedings of the conference on Visualization'01*, pages 349–356. IEEE Computer Society, 2001.

[40] Stefan Guthe and Wolfgang Strasser. Advanced techniques for high-quality multi-resolution volume rendering. *Computers & Graphics*, 28(1):51–58, 2004.

[41] Subhashis Hazarika, Soumya Dutta, Han-Wei Shen, and Jen-Ping Chen. Codda: A flexible copula-based distribution driven analysis framework for large-scale multivariate data. *IEEE transactions on visualization and computer graphics*, 2018.

[42] Wenbin He, Chun-Ming Chen, Xiaotong Liu, and Han-Wei Shen. A bayesian approach for probabilistic streamline computation in uncertain flows. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 214–218. IEEE, 2016.

[43] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics letters*, 44(13):800–801, 2008.

[44] Guangfeng Ji and Han-Wei Shen. Dynamic view selection for time-varying volumes. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

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

[46] Dimitris Karlis and Evdokia Xekalaki. Choosing initial values for the EM algorithm for finite mixtures. *Computational Statistics & Data Analysis*, 41(3):577–590, 2003.

[47] Gordon Kindlmann and James W Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 79–86. ACM, 1998.

[48] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. In *European Conference on Parallel Processing*, pages 366–379. Springer, 2011.

[49] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, Choong-Seock Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013.

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

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

[52] Juan Li, Jin Wu, Shen Yang, and Jin Liu. Dictionary learning for image super-resolution. In *Control Conference (CCC), 2014 33rd Chinese*, pages 7195–7199. IEEE, 2014.

[53] Shaomeng Li, Nicole Marsaglia, Christoph Garth, Jonathan Woodring, John Clyne, and Hank Childs. Data reduction techniques for simulation, visualization and data analysis. In *Computer Graphics Forum*, volume 37, pages 422–447. Wiley Online Library, 2018.

[54] Jimmy J. Lin, Jian Pei, Xiaohua Hu, Wo Chang, Raghunath Nambiar, Charu Aggarwal, Nick Cercone, Vasant Honavar, Jun Huan, Bamshad Mobasher, and Saumyadipta Pyne, editors. *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*. IEEE, 2014.

[55] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

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

[57] Li-ta Lo, Christopher Sewell, and James P Ahrens. Piston: A portable cross-platform framework for data-parallel visualization operators. In *EGPGV*, pages 11–20, 2012.

[58] Eric B Lum, Kwan Liu Ma, and John Clyne. Texture hardware assisted rendering of time-varying volume data. In *Proceedings of the conference on Visualization'01*, pages 263–270. IEEE Computer Society, 2001.

[59] Claes Lundström, Patric Ljung, Anders Persson, and Anders Ynnerman. Uncertainty visualization in medical volume rendering using probabilistic animation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1648–1655, 2007.

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

[61] Desire Luc Massart, A J Smeyers-verbeke, et al. Practical data handling visual presentation of data by means of box plots. 2005.

[62] Geoffrey McLachlan. *Discriminant analysis and statistical pattern recognition*, volume 544. John Wiley & Sons, 2004.

[63] M Meyer, S Takahashi, and A Vilanova. Data reduction techniques for scientific visualization and data analysis. *STAR*, 36(3), 2017.

[64] Miriah Meyer, Hanspeter Pfister, Charles Hansen, Chris Johnson, Miriah Meyer, Hanspeter Pfister, Charles Hansen, and Chris Johnson. Image-based volume rendering with opacity light fields. *no. UUSCI-2005-002. Tech Report*, 2005.

[65] Kenneth Moreland, Christopher Sewell, William Usher, Li-ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, et al. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE computer graphics and applications*, 36(3):48–58, 2016.

[66] Klaus Mueller, Naeem Shareef, Jian Huang, and Roger Crawfis. Ibr-assisted volume rendering. *Late Breaking Hot Topics of Visualization99*, pages 5–9, 1999.

[67] Keith W Oleson, Gordon B Bonan, J Feddema, and Mariana Vertenstein. An urban parameterization for a global climate model. part ii: Sensitivity to input parameters and the simulated urban heat island in offline simulations. *Journal of Applied Meteorology and Climatology*, 47(4):1061–1076, 2008.

[68] Patrick OLeary, James Ahrens, Sébastien Jourdain, Scott Wittenburg, David H Rogers, and Mark Petersen. Cinema image-based in situ analysis and visualization of mpas-ocean simulations. *Parallel Computing*, 55:43–48, 2016.

[69] J Patchett and G Gisler. Deep water impact ensemble data set. *Los Alamos National Laboratory, LA-UR-17-21595, available at http://dssdata. org*, 2017.

[70] David Posada and Thomas R Buckley. Model selection and model averaging in phylogenetics: advantages of akaike information criterion and bayesian approaches over likelihood ratio tests. *Systematic biology*, 53(5):793–808, 2004.

[71] AL Read. Linear interpolation of histograms. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 425(1):357–360, 1999.

[72] Christof Rezk-Salama, Severin Todt, and Andreas Kolb. Raycasting of light field galleries from volumetric data. In *Computer Graphics Forum*, volume 27, pages 839–846. Wiley Online Library, 2008.

[73] Marcos Balsa Rodríguez, Enrico Gobbetti, José Antonio Iglesias Guitián, Maxim Makhinya, Fabio Marton, Renato Pajarola, and Susanne K Suter. A survey of compressed gpu-based direct volume rendering. In *Eurographics (STARs)*, pages 117–136, 2013.

[74] Elham Sakhaee and Alireza Entezari. A statistical direct volume rendering framework for visualization of uncertain data. *IEEE transactions on visualization and computer graphics*, 23(12):2509–2520, 2017.

[75] Franz Sauer, Jinrong Xie, and Kwan-Liu Ma. A combined eulerian-lagrangian data representation for large-scale applications. *IEEE transactions on visualization and computer graphics*, 23(10):2248–2261, 2017.

[76] Steven Schlegel, Nico Korn, and Gerik Scheuermann. On the interpolation of data with normally distributed uncertainty for visualization. *IEEE transactions on visualization and computer graphics*, 18(12):2305–2314, 2012.

[77] Christopher Sewell, Katrin Heitmann, Hal Finkel, George Zagaris, Suzanne T Parete-Koon, Patricia K Fasel, Adrian Pope, Nicholas Frontiere, Li-ta Lo, Bronson Messer, et al. Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 50. ACM, 2015.

[78] Naeem Shareef, Teng-Yok Lee, Han-Wei Shen, and Klaus Mueller. An image-based modelling approach to gpu-based unstructured grid volume rendering. *In Proc. of Volume Graphics*, pages 31–38, 2006.

[79] Han-Wei Shen, Ling-Jen Chiang, and Kwan-Liu Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 371–377. IEEE Computer Society Press, 1999.

[80] Ronell Sicat, Jens Kruger, Torsten Möller, and Markus Hadwiger. Sparse pdf volumes for consistent multi-resolution volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2417–2426, 2014.

[81] Kurt Stockinger, John Shalf, Kesheng Wu, and E Wes Bethel. Query-driven visualization of large data sets. In *IEEE Visualization*, volume 5, page 22, 2005.

[82] Yu Su, Gagan Agrawal, and Jonathan Woodring. Indexing and parallel query processing support for visualizing climate datasets. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 249–258. IEEE, 2012.

[83] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Fixed-psnr lossy compression for scientific data. *arXiv preprint arXiv:1805.07384*, 2018.

[84] David Thompson, Joshua A Levine, Janine C Bennett, P-T Bremer, Attila Gyulassy, Valerio Pascucci, and Philippe P Pébay. Analysis of large-scale scalar data using hixels. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 23–30. IEEE, 2011.

[85] Anna Tikhonova, C Correa, and Kwan-Liu Ma. Visualization by proxy: A novel framework for deferred interaction with volume data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1551–1559, 2010.

[86] Anna Tikhonova, Carlos D Correa, and K-L Ma. An exploratory technique for coherent visualization of time-varying volume data. In *Computer Graphics Forum*, volume 29, pages 783–792. Wiley Online Library, 2010.

[87] Anna Tikhonova, Carlos D Correa, and Kwan-Liu Ma. Explorable images for visualizing volume data. In *PacificVis*, pages 177–184. Citeseer, 2010.

[88] Anna Tikhonova, Hongfeng Yu, Carlos D Correa, Jacqueline H Chen, and Kwan-Liu Ma. A preview and exploratory technique for large-scale scientific simulations. In *EGPGV*, pages 111–120. Citeseer, 2011.

[89] Dennis Wackerly, William Mendenhall, and Richard L Scheaffer. *Mathematical statistics with applications*. Nelson Education, 2007.

[90] Chaoli Wang and Han-Wei Shen. Information theory in scientific visualization. *Entropy*, 13(1):254–273, 2011.

[91] Junpeng Wang, Subhashis Hazarika, Cheng Li, and Han-Wei Shen. Visualization and visual analysis of ensemble data: A survey. *IEEE transactions on visualization and computer graphics*, 2018.

[92] Junpeng Wang, Xiaotong Liu, Han-Wei Shen, and Guang Lin. Multi-resolution climate ensemble parameter analysis with nested parallel coordinates plots. *IEEE transactions on visualization and computer graphics*, 23(1):81–90, 2017.

[93] Ko-Chih Wang, Kewei Lu, Tzu-Hsuan Wei, Naeem Shareef, and Han-Wei Shen. Statistical visualization and analysis of large data using a value-based spatial distribution. In *Pacific Visualization Symposium (PacificVis), 2017 IEEE*, pages 161–170. IEEE, 2017.

[94] Ko-Chih Wang, Naeem Shareef, and Han-Wei Shen. Image and distribution based volume rendering for large data sets. In *Pacific Visualization Symposium (PacificVis), 2018 IEEE*, pages 26–35. IEEE, 2018.

[95] Tzu-Hsuan Wei, Chun-Ming Chen, and Ayan Biswas. Efficient local histogram searching via bitmap indexing. In *Computer Graphics Forum*, volume 34, pages 81–90. Wiley Online Library, 2015.

[96] Tzu-Hsuan Wei, Soumya Dutta, and Han-Wei Shen. Information guided data sampling and recovery using bitmap indexing. In *Pacific Visualization Symposium (PacificVis), 2018 IEEE*, pages 56–65. IEEE, 2018.

[97] Jonathan Woodring, J Ahrens, J Figg, Joanne Wendelberger, Salman Habib, and Katrin Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum*, volume 30, pages 1151–1160. Wiley Online Library, 2011.

[98] Jinrong Xie, Franz Sauer, and Kwan-Liu Ma. Fast uncertainty-driven large-scale volume feature extraction on desktop pcs. In *Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on*, pages 17–24. IEEE, 2015.

[99] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010.

[100] Yucong Chris Ye, Yang Wang, Robert Miller, Kwan-Liu Ma, and Kenji Ono. In situ depth maps based feature extraction and tracking. In *Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on*, pages 1–8. IEEE, 2015.

[101] B Yegnanarayana and S Prahallad Kishore. Aann: an alternative to gmm for pattern recognition. *Neural Networks*, 15(3):459–469, 2002.

[102] H Yela, I Navazo, and P Vazquez. S3dc: A 3dcbased volume compression algorithm. *Proc. CEIG*, 2, 2008.

[103] Jiangtao Yin, Yanfeng Zhang, and Lixin Gao. Accelerating expectation-maximization algorithms with frequent updates. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 275–283. IEEE, 2012.

[104] Hamid Younesy, Torsten Möller, and Hamish Carr. Improving the quality of multi-resolution volume rendering. In *EuroVis*, pages 251–258. Citeseer, 2006.

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

[106] Hongfeng Yu, Jinrong Xie, Kwan-Liu Ma, Hemanth Kolla, and Jacqueline H Chen. Scalable parallel distance field construction for large-scale applications. *IEEE transactions on visualization and computer graphics*, 21(10):1187–1200, 2015.