Ray-based Exploration of Large Time-varying Volume Data Using Per-ray Proxy Distributions

Ko-Chih Wang, Tzu-Hsuan Wei, Shareef Naeem, and Han-Wei Shen *Member, IEEE*,

Abstract—The analysis and visualization of data created from simulations on modern supercomputers is a daunting challenge because the incredible compute power of modern supercomputers allow scientists to generate datasets with very high spatial and temporal resolutions. The limited bandwidth and capacity of networking and storage devices connecting supercomputers to analysis machines become the major bottleneck for data analysis such that simply moving the whole dataset from the supercomputer to a data analysis machine is infeasible. A common approach to visualize high temporal resolution simulation datasets under constrained I/O is to reduce the sampling rate in the temporal domain while preserving the original spatial resolution at the time steps. Data interpolation between the sampled time steps alone may not be a viable option since it may suffer from large errors, especially when using a lower sampling rate. We present a novel ray-based representation storing ray based histograms and depth information that recovers the evolution of volume data between sampled time steps. Our view-dependent proxy allows for a good trade off between compactly representing the time-varying data and leveraging temporal coherence within the data by utilizing interpolation between time steps, ray histograms, depth information, and codebooks. Our approach is able to provide fast rendering in the context of transfer function exploration to support visualization of feature evolution in time-varying data.

Index Terms—Data visualization, time-varying data, large data modeling, scientific simulation.

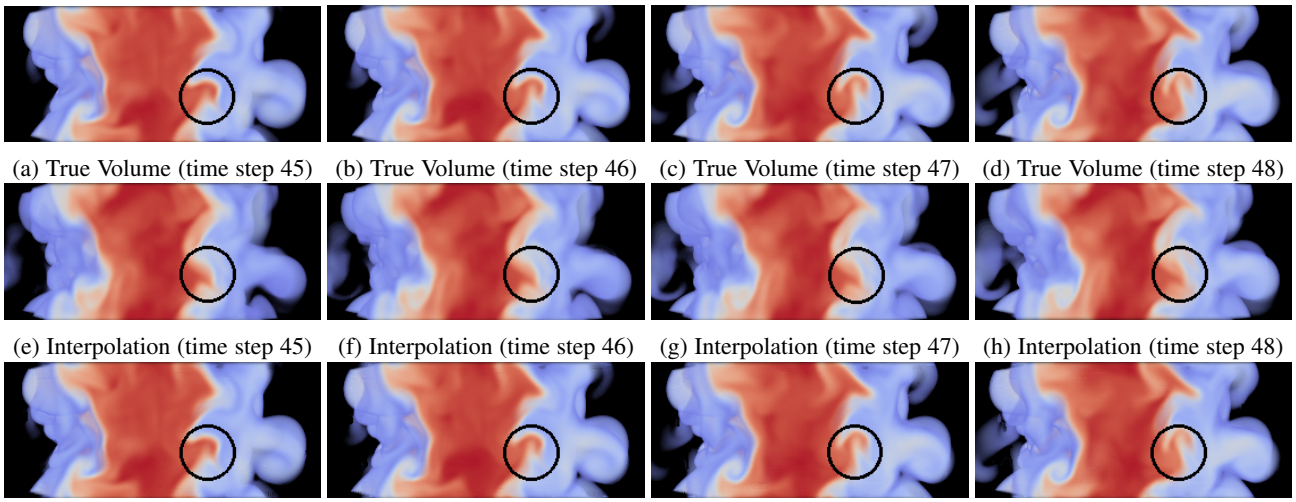
1 Introduction

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 [1], [2], [3]. 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 [4], [5], [6]. 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 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 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 [7], [8], [9] and spatial subsampling-based techniques [2], [10], [11]. 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 [1], [12], [13] 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

• *K.-C. Wang, T.-H. Wei, N. Shareef and H.-W. Shen are with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, 43210.*
E-mail: {wang.3182, wei.225, shareef.1, shen.94}@osu.edu



(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)

Fig. 1: Our approach mixes 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 the 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 a similar feature evolution in the second row. This may mislead the scientists' analysis.

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.

2 Related Work

Large data representations: Many approaches have reduced data size in object space in order to overcome I/O overhead. Xie et al. [14] used a hybrid technique combining GPU-based SLIC clustering and a multi-resolution approach to handle large data on desktop PCs. Sauer et al. [15] used a novel data structure that combines Eulerian and Lagrangian reference frames to provide efficient data sampling and querying. Chen et al. [4] improved pathline computation by interpolating down-sampled data and then modeling the error with a Gaussian distribution. Ma [5] and Yu et al. [6] proposed systems for in-situ simulations. Guthe and Strasser [9], Binotto et al. [16] and Lum et al. [17] introduced various compression strategies. Lakshminarasimhan et al. [7] proposed a data compression technique utilizing B-splines to compactly model a set of sorted scalar values and preserve a mapping between sorted and unsorted data for decompression. Lindstrom and Isenburg [8] used floating-point lossy compression. These object-based approaches do not leverage image space parameters nor temporal coherence in time-varying data. Meyer et al. [18] surveyed data reduction techniques for scientific datasets and categorized approaches in term of different use cases.

View-dependent approaches: These approaches have had success for rendering on commodity hardware using proxy representations. Image-based rendering approaches can perform fast rendering from a fixed viewpoint. View samples were pre-computed and the volume was rendered from these different viewpoints [19], [20], [21]. Mueller et al. pre-computed slabs at each view sample [22]. Wang et al. [3] used distributions to summarize data within each pixel frustum to enable transfer function exploration. However, these approaches using ray based data proxies have not utilized temporal coherence. Tikhonova et al. [12], [13], [23] proposed a view-dependent technique that stores

image slices that allows for fast transfer function changes. These approaches have good storage utilization, but their performance depends on various assumptions such as the spatial relation of the data layers or the initial transfer function. Ahrens et al. [1] proposed a system which renders images in-situ from raw data and only outputs images to the post-analysis machine, thus reducing I/O overhead. The volume depth image proposed by Fernandes et al. [24] modeled data along a pixel ray in the spatiotemporal domain and compactly stored time-varying scientific datasets. Although both of these approaches provide compact data representations, scientists cannot freely explore occluded features by changing the transfer function. Correa and Ma [25] assisted transfer function design to reveal occluded features with a ray distribution for visibility. They automatically generated transfer functions to reveal occluded features.

Distribution representations: Recent research has focused on distribution-based techniques for compact data representation. Thompson et al. [26] and Liu et al. [27] used distributions to summarize local region information in the object and simulation parameter spaces [28]. Wang et al. [10] used spatial distributions to store sample information in data value intervals. Dutta et al. [2] used an object space approach to improve on storage savings and I/O bottlenecks by partitioning the dataset into local blocks where each block is modeled with the Gaussian Mixture Model in an in-situ simulation. Sicat et al. [11] constructed distributions at different levels-of-detail for multi-resolution volume rendering. These distribution-based representations do not leverage the image space parameters used to visualize the data to create the data proxies as we do in our work. Chaudhuri et al. [29], [30] and Lee et al. [31] used a histogram based Summed Area Table in order to access the histogram of any sub-volume of the data set. Wei et al. [32], [33], [34] used bitmap indexing to efficiently query the data distribution on arbitrary sub-domains. However, their focus was efficient data access instead of the data reduction.

3 Overview

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 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 6) 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) 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 4) 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 1.

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

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 7.

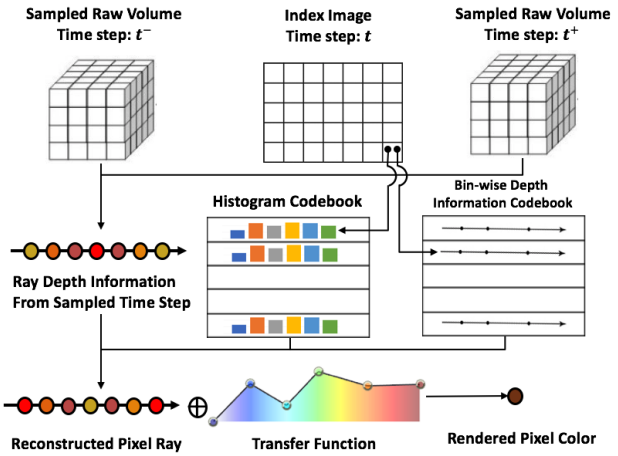


Fig. 2: The proxy data structure stores sampled volume time steps and a collection of index images at each non-sampled time step, which are located between two neighboring sampled time steps (t^- and t^+). The samples defined on a pixel ray are represented by a ray histogram and bin-wise depth informations 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.

4 Ray-based Proxy Representation

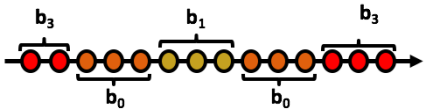
Each non-sampled volume time step is replaced with a proxy consisting of index images defined at pre-chosen view samples. At each pixel ray of an index image, a ray is cast into the non-sampled volume and sampled in depth order. The samples are encoded in a statistical summary, called a *ray histogram*, 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 7 describes how the ray histograms and bin-wise depth information are stored into codebooks for further savings. Sections 5 and 6 present two alternative pixel ray representations to the *decoupled ray representation* that are more compact by leveraging temporal coherence.

4.1 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.

4.2 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 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 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 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 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 3, bin b_3 has two runs along the ray with weights 0.4 and 0.6, respectively.



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)

Fig. 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.

$$w_b(c) = \frac{SmpCount_b(c)}{\sum_{i=0}^{N-1} SmpCount_b(i)} \tag{2}$$

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 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 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.

$$o_b(c) = NextBinID(c) - BinID(c) \tag{3}$$

4.3 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 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.

$$SmpCount_{b_i}(c) = S * H(b_i) * w(c) \tag{4}$$

The steps to compute the pixel color are shown in Algorithm 1. 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.

Algorithm 1 Ray reconstruction and rendering with the ray histogram and bin-wise depth information

```
 1: ▷ rh: ray histogram; dptInfos: bin-wise depth informations
 2: ▷ stBinID: starting Bin ID of the ray
 3: ▷ S: expected number of samples on the reconstructed ray
 4: procedure RayReconstruction(rh, depInfors, stBinID, S)
 5:     color = (0,0,0)
 6:     currBin = stBinID
 7:     currComps = [0,0,...,0]
 8:
 9:     while currBin ≠ −1 do
10:         di = dptInfos(currBin)
11:         compSmps = S*rh[currBin]*di.w[currComps[currBin]]
12:         rgba = tf(currBin)
13:         alpha = 1.0 - pow((1.0 - rgba.a), compSmps)
14:         color += clr.rgb * alpha
15:         currBin += di.o[currComps[currBin]]
16:         currComps[currBin] += 1
17:     end while
18:
19:     return color
20: end procedure
```

5 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 4.2. 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 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.

5.1 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 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.

5.2 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 2 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 2 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.

6 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

$$R_t(\ell) = R_{t^-}(\ell) * (1-a) + R_{t^+}(\ell) * a \tag{5}$$

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 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) should be used to reconstruct the pixel ray samples during rendering.

7 Ray Histogram and Bin-wise Depth Information Codebooks

The ray histogram and bin-wise depth data structures presented in Section 4 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.

7.1 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.

7.1.1 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 6

$$L_1(H_i,H_j) = \sum_{k=0}^{B-1} \left| H_i(b_k) - H_j(b_k) \right| \tag{6}$$

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 7

$$HistD(H_i,H_j) = \begin{cases} L_1(H_i,H_j) & if\ nz(H_i) = nz(H_j) \\ \infty & Otherwise \end{cases} \tag{7}$$

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

7.1.2 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 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 [32], [35] for fast value range queries.

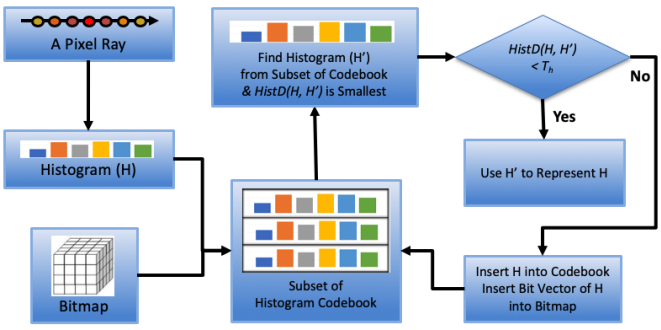


Fig. 4: The ray histogram codebook is constructed by individually inserting each ray histogram. If a ray histogram is similar using a pre-defined threshold to a histogram already contained in the codebook, then it will index the representative histogram. Otherwise the ray histogram is inserted into the codebook.

Bitmap Indexing Table: Equation 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 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 1 shows a codebook containing six

TABLE 1: 2D bitmap indexing example

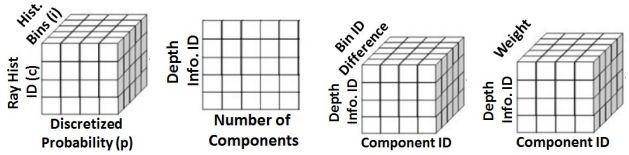
Ray Histogram Database		Bitmap Indexing					
ID	Bin Prob	p_0	p_1	p_2	p_3	p_4	p_5
		0	(0,0.2]	(0.2,0.4]	(0.4,0.6]	(0.6,0.8]	(0.8,1.0]
0	0.7	0	0	0	0	1	0
1	0.6	0	0	0	1	0	0
2	0	1	0	0	0	0	0
3	0.2	0	1	0	0	0	0
4	0.5	0	0	0	1	0	0
5	0.9	0	0	0	0	0	1

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], 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(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 8.

$$bmp_{(c,i,p)} = \begin{cases} 1 & if\ p = 0,\ H_c(b_i) = 0 \\ 1 & if\ p \neq 0,\ f(H_c(b_i)) = p \\ 0 & Otherwise \end{cases} \tag{8}$$

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 8 and inserted it at the bottom of the table.



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

Fig. 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 9

$$C_1 = \Big(\bigwedge_{b \in H_n(b)=0} bmp_{(:,b,0)}\Big) \wedge \Big(\bigwedge_{b \in H_n(b)\neq 0} \neg bmp_{(:,b,0)}\Big) \tag{9}$$

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 10

$$C_2 = \bigwedge_{\forall b'} \Big(\bigvee_{p=f(H_n(b')-T_h)}^{f(H_n(b')+T_h)} bmp_{(:,b',p)}\Big) \tag{10}$$

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 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 $HistD(H_n, H_e) \leq T_h$. 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 6 and 7 and the check $HistD(H_n, H_e) \leq T_h$. 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.

7.2 Bin-wise Depth Information Codebook

Bin-wise depth information described in Section 4.2 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.

7.2.1 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 11

$$L_1^{weight}(w_i, w_j) = \sum_{c=0}^{C-1} \left| w_i(c) - w_j(c) \right| \tag{11}$$

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

7.2.2 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 $L_1^{weight}(w_i, w_j) \leq T_d$.

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 (b) - (d). A 2-dimensional bitmap indexing table (see Figure 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 12

$$compBmp_{(d,c)} = \begin{cases} 1 & if\ Comp(DI_d) = c \\ 0 & Otherwise \end{cases} \tag{12}$$

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(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(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 6 illustrates the incremental algorithm to construct the codebook with the three bitmaps.

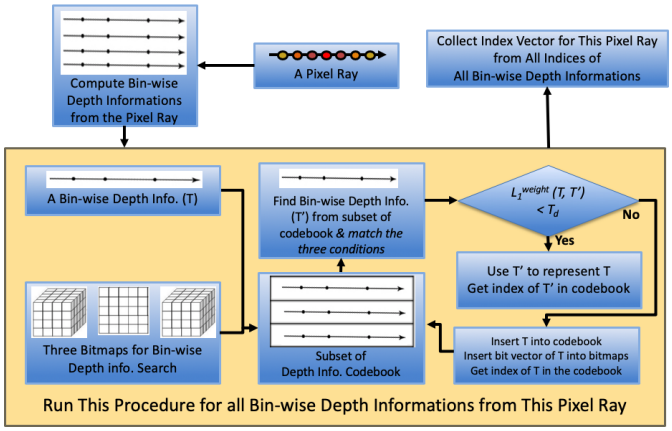


Fig. 6: Steps to construct the bin-wise depth information codebook for a pixel ray.

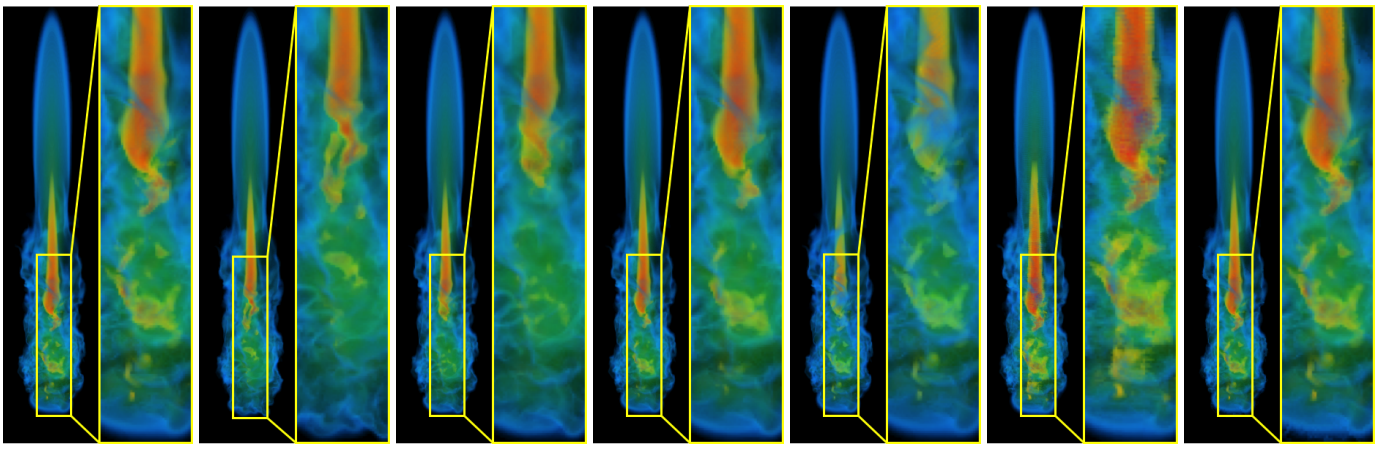
8 Multi-view Rendering

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.

9 Evaluation

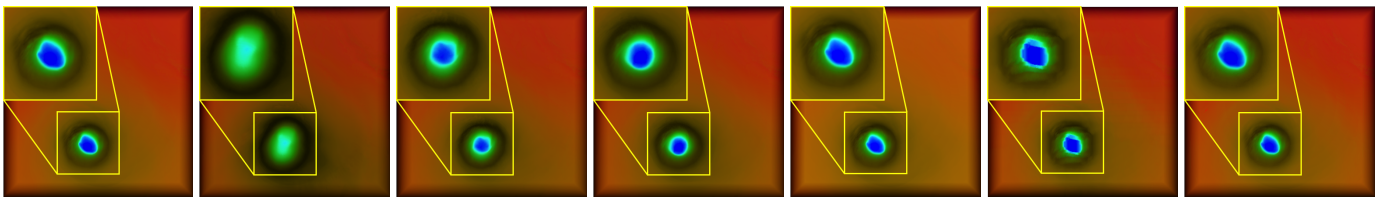
We present qualitative and quantitative results from experiments using three datasets. The Plume dataset is a simulation of Solar Plume for thermal down flow on the surface layer of the Sun and was provided by the National Center for Atmospheric Research. Our experiments used the vector magnitude field. 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 [36], called *IAF*, a state-of-the-art lossy compression technique called *ISABELA* [37], 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,



(a) Ground truth (b) STI: 10 (c) STI: 5 (d) STI: 2 (e) IAF (f) ISABELA (g) Our approach

Fig. 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

Fig. 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.

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.

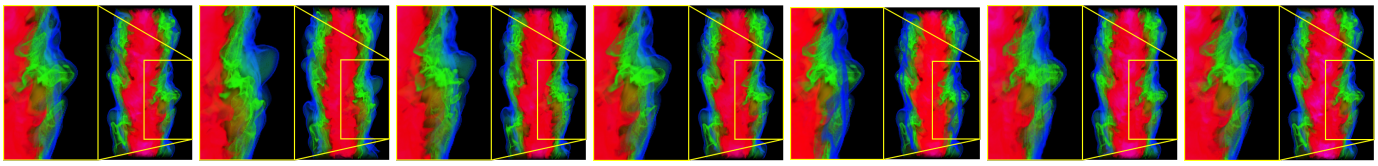
9.1 Qualitative Evaluation

Figures 7, 8, and 9 compare our approach with the others. Figure 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 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 9 compares results on the Combustion dataset focusing on the region with pure fuel mass, which is 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 10 using the same view-dependent proxies and transfer functions that were used for the results in Figures 7, 8 and 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.

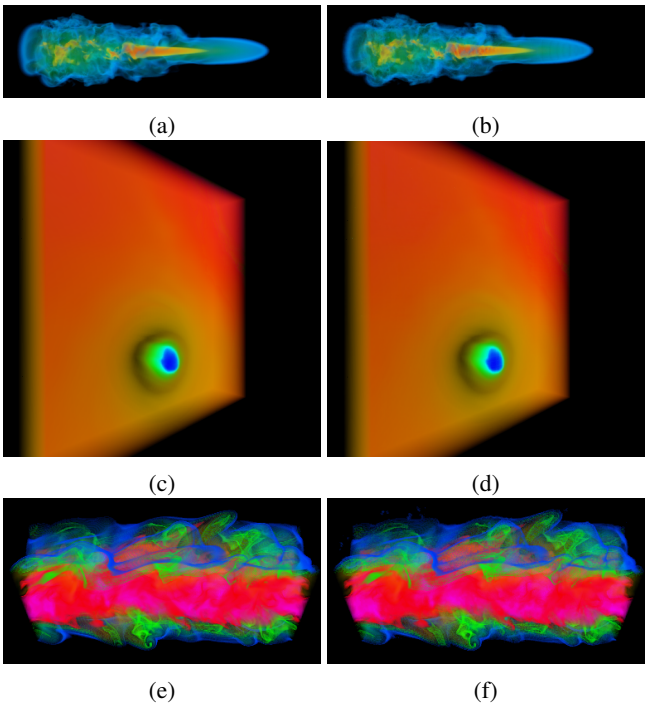
9.2 Quantitative Evaluation

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 curves are shown in Figure 11 where the red curve is



(a) Ground truth (b) STI: 10 (c) STI: 5 (d) STI: 2 (e) IAF (f) ISABELA (g) Our approach

Fig. 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.



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

Fig. 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 7, 8 and 9.

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 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.

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 7, 8, and 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. Each bar in Figure 12 shows the average PSNR of rendered images of all time steps of a single approach and transfer function. 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.

9.3 Proxy Storage Cost and Construction Time

We report in Table 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

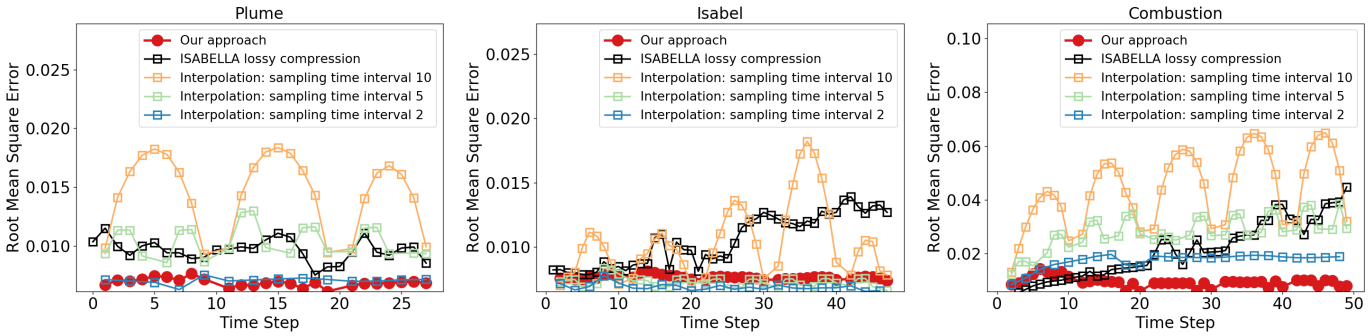


Fig. 11: Quantitative evaluation of our approach and interpolation approximation and ISABELA lossy compression using RMSE.

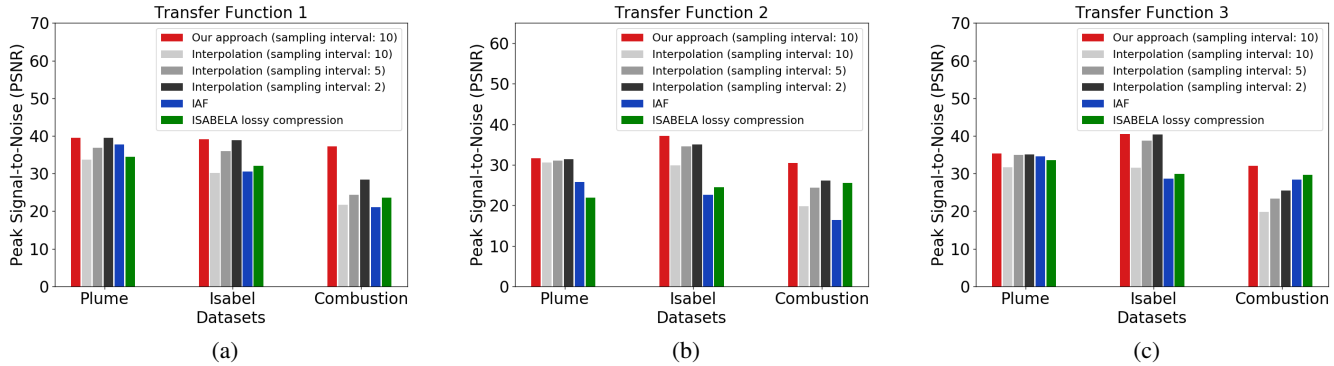


Fig. 12: Image quality comparison using average PSNR between our approach and interpolation approximation, IAF, and ISABELA lossy compression.

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 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 [38], 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 9.2 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.

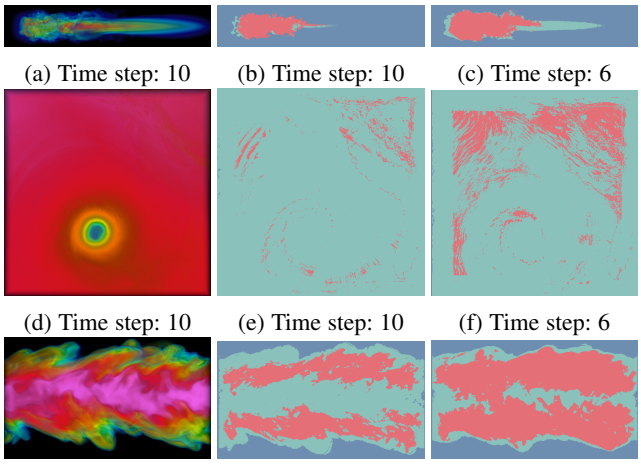
10 Discussion

10.1 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 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 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 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 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.



(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

Fig. 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.

10.2 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 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 14 (a) and 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 14 (c) and 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 14 (e) shows that the RMSE of 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 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 14 (g) and 14 (h). Figure 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 14 (i).

10.3 View Selection

Selecting view samples to locate index images can be determined manually or automatically. The scientist 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 [39], [40] 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.

11 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

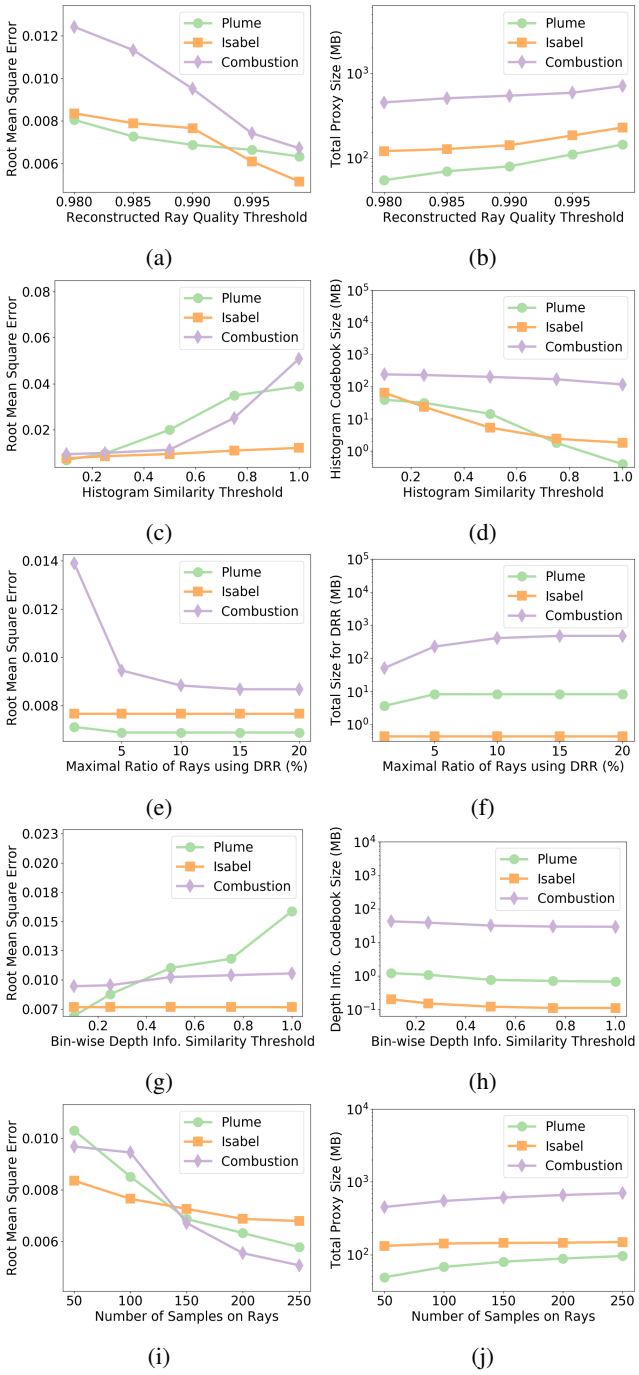


Fig. 14: Parameter study on thresholds of the reconstructed ray quality, histogram similarity, bin-wise depth information similarity, maximal ratio of rays using the decoupled ray representation and number of samples on rays. DRR stands for decoupled ray representation.

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.

REFERENCES

[1] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. 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*. IEEE Press, 2014, pp. 424–434.

[2] S. Dutta, C. Chun-Ming, G. Hernlein, H.-W. Shen, and J. Chen, "In situ distribution guided analysis and visualization of transonic jet engine simulations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 837–846, 2016.

[3] K.-C. Wang, N. Shareef, and H.-W. Shen, "Image and distribution based volume rendering for large data sets," in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2018, pp. 26–35.

[4] C.-M. Chen, A. Biswas, and H.-W. Shen, "Uncertainty modeling and error reduction for pathline computation in time-varying flow fields," in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2015, pp. 215–222.

[5] K.-L. Ma, "In situ visualization at extreme scale: Challenges and opportunities," *IEEE Computer Graphics and Applications*, vol. 29, no. 6, pp. 14–19, 2009.

[6] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma, "In situ visualization for large-scale combustion simulations," *IEEE Computer Graphics and Applications*, vol. 30, no. 3, pp. 45–57, 2010.

[7] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *European Conference on Parallel Processing*. Springer, 2011, pp. 366–379.

[8] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[9] S. Guthe and W. Strasser, "Real-time decompression and visualization of animated volume data," in *Proceedings of Visualization*. IEEE Computer Society, 2001, pp. 349–572.

[10] K.-C. Wang, K. Lu, T.-H. Wei, N. Shareef, and H.-W. Shen, "Statistical visualization and analysis of large data using a value-based spatial distribution," in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2017, pp. 161–170.

[11] R. Sicat, J. Kruger, T. Möller, and M. Hadwiger, "Sparse pdf volumes for consistent multi-resolution volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2417–2426, 2014.

[12] A. Tikhonova, C. D. Correa, and K.-L. Ma, "Explorable images for visualizing volume data." in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2010, pp. 177–184.

[13] A. Tikhonova, C. Correa, and K.-L. Ma, "Visualization by proxy: A novel framework for deferred interaction with volume data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1551–1559, 2010.

[14] J. Xie, F. Sauer, and K.-L. Ma, "Fast uncertainty-driven large-scale volume feature extraction on desktop pcs," in *Large Data Analysis and Visualization (LDAV)*. IEEE, 2015, pp. 17–24.

[15] F. Sauer, J. Xie, and K.-L. Ma, "A combined eulerian-lagrangian data representation for large-scale applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 10, pp. 2248–2261, 2017.

[16] A. P. D. Binotto, J. L. Comba, and C. M. Freitas, "Real-time volume rendering of time-varying data using a fragment-shader compression approach," in *Parallel and Large-Data Visualization and Graphics*. IEEE, 2003, pp. 69–75.

[17] E. B. Lum, K. L. Ma, and J. Clyne, "Texture hardware assisted rendering of time-varying volume data," in *Proceedings of Visualization*. IEEE Computer Society, 2001, pp. 263–270.

[18] M. Meyer, S. Takahashi, and A. Vilanova, "Data reduction techniques for scientific visualization and data analysis," *STAR*, vol. 36, no. 3, 2017.

[19] N. Shareef, T.-Y. Lee, H.-W. Shen, and K. Mueller, "An image-based modelling approach to gpu-based unstructured grid volume rendering," *In Proceedings of Volume Graphics*, pp. 31–38, 2006.

[20] S. Frey, F. Sadlo, and T. Ertl, "Explorable volumetric depth images from raycasting," in *Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 2013, pp. 123–130.

[21] S. Frey and T. Ertl, "Auto-tuning intermediate representations for in situ visualization," in *Scientific Data Summit (NYSDS)*. IEEE, 2016, pp. 1–10.

[22] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "Ibr-assisted volume rendering," *Late Breaking Hot Topics of Visualization*, pp. 5–9, 1999.

[23] A. Tikhonova, C. D. Correa, and K.-L. Ma, "An exploratory technique for coherent visualization of time-varying volume data," in *Computer Graphics Forum*, vol. 29, no. 3. Wiley Online Library, 2010, pp. 783–792.

[24] O. Fernandes, S. Frey, F. Sadlo, and T. Ertl, "Space-time volumetric depth images for in-situ visualization," in *Large Data Analysis and Visualization (LDAV)*. IEEE, 2014, pp. 59–65.

[25] C. D. Correa and K.-L. Ma, "Visibility-driven transfer functions," in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2009, pp. 177–184.

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

[27] S. Liu, J. A. Levine, P. Bremer, and V. Pascucci, "Gaussian mixture model based volume visualization," in *Large Data Analysis and Visualization (LDAV)*. IEEE, 2012, pp. 73–77.

[28] J. Wang, S. Hazarika, C. Li, and H.-W. Shen, "Visualization and visual analysis of ensemble data: A survey," *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[29] A. Chaudhuri, T.-Y. Lee, B. Zhou, C. Wang, T. Xu, H.-W. Shen, T. Peterka, and Y.-J. Chiang, "Scalable computation of distributions from large scale data sets," in *Large Data Analysis and Visualization (LDAV)*. IEEE, 2012, pp. 113–120.

[30] A. Chaudhuri, T. H. Wei, T. Y. Lee, H. W. Shen, and T. Peterka, "Efficient range distribution query for visualizing scientific data," in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2014, pp. 201–208.

[31] T.-Y. Lee and H.-W. Shen, "Efficient local statistical analysis via integral histograms with discrete wavelet transform," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2693–2702, 2013.

[32] T.-H. Wei, C.-M. Chen, and A. Biswas, "Efficient local histogram searching via bitmap indexing," in *Computer Graphics Forum*, vol. 34, no. 3. Wiley Online Library, 2015, pp. 81–90.

[33] T.-H. Wei, C.-M. Chen, J. Woodring, H. Zhang, and H.-W. Shen, "Efficient distribution-based feature search in multi-field datasets," in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2017, pp. 121–130.

[34] T.-H. Wei, S. Dutta, and H.-W. Shen, "Information guided data sampling and recovery using bitmap indexing," in *Pacific Visualization Symposium (PacificVis)*. IEEE, 2018, pp. 56–65.

[35] Y. Su, G. Agrawal, and J. Woodring, "Indexing and parallel query processing support for visualizing climate datasets," in *International Conference on Parallel Processing (ICPP)*. IEEE, 2012, pp. 249–258.

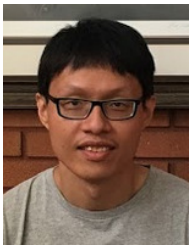
[36] A. Tikhonova, H. Yu, C. D. Correa, J. H. Chen, and K.-L. Ma, "A preview and exploratory technique for large-scale scientific simulations." in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*. Eurographics Association, 2011, pp. 111–120.

[37] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C.-S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Isabela for effective in situ compression of scientific data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 524–540, 2013.

[38] K. Moreland, C. Sewell, W. Usher, L.-t. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K.-L. Ma, H. Childs *et al.*, "Vtk-m: Accelerating the visualization toolkit for massively threaded architectures," *Computer Graphics and Applications*, vol. 36, no. 3, pp. 48–58, 2016.

[39] U. D. Bordoloi and H.-W. Shen, "View selection for volume rendering," in *Proceedings of Visualization*. IEEE, 2005, pp. 487–494.

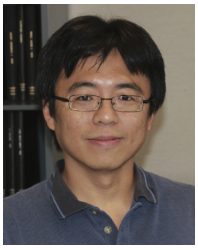
[40] G. Ji and H.-W. Shen, "Dynamic view selection for time-varying volumes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1109–1116, 2006.



Ko-Chih Wang is a Ph.D. student in the Department of Computer Science and Engineering at the Ohio State University. He received his B.S. degree in computer science from Tunghai University in 2004, and M.S. degree from Graduate Institute of Networking and Multimedia of National Taiwan University in 2007. His research interests are mainly in the visualization and analysis of large scientific simulation data, computer graphics and high performance computing.

Tzu-Hsuan Wei is a Machine Learning Engineer at Zillow. He received his B.S. degree from Department of Space Science and Atmospheric at National Central University in 2003, the M.S. degree in computer science from National Central University in 2005, and the Ph.D. degree in computer science at the Ohio State University in 2017. His research interests are mainly in data analysis and visualization, computer graphics, image processing, and high performance computing.

Naeem Shareef is a senior lecturer in the Department of Computer Science and Engineering at the Ohio State University. He received his B.S. degree in applied math and computer science at Carnegie Mellon University in 1990, the MS degree in computer science at the Ohio State University in 1992, and the Ph.D.degree in computer science at the Ohio State University in 2005. His primary research interests are scientific visualization, computer graphics, and artificial intelligence.



Han-Wei Shen is a full professor at the Ohio State University. He received his B.S. degree from Department of Computer Science and Information Engineering at National Taiwan University in 1988, the M.S. degree in computer science from the State University of New York at Stony Brook in 1992, and the Ph.D. degree in computer science from the University of Utah in 1998. From 1996 to 1999, he was a research scientist at NASA Ames Research Center in Mountain View California. His primary research interests are scientific visualization and computer graphics. He is a winner of the National Science Foundation's CAREER award and U.S. Department of Energy's Early Career Principal Investigator Award. He also won the Outstanding Teaching award twice in the Department of Computer Science and Engineering at the Ohio State University.

ACKNOWLEDGMENTS

This work was supported in part by UT-Battelle LLC 4000159557, Los Alamos National Laboratory Contract 471415, and NSF grant SBE-1738502.