Image and Distribution Based Volume Rendering for Large Data Sets

Ko-Chih Wang* Naeem Shareef† Han-Wei Shen‡

The Ohio State University

ABSTRACT

Analyzing scientific datasets created from simulations on modern supercomputers is a daunting challenge due to the fast pace at which these datasets continue to grow. Low cost post analysis machines used by scientists to view and analyze these massive datasets are severely limited by their deficiencies in storage bandwidth, capacity, and computational power. Trying to simply move these datasets to these platforms is infeasible. Any approach to view and analyze these datasets on post analysis machines will have to effectively address the inevitable problem of data loss. Image based approaches are well suited for handling very large datasets on low cost platforms. Three challenges with these approaches are how to effectively represent the original data with minimal data loss, analyze the data in regards to transfer function exploration, which is a key analysis tool, and quantify the error from data loss during analysis. We present a novel image based approach using distributions to preserve data integrity. At each view sample, view dependent data is summarized at each pixel with distributions to define a compact proxy for the original dataset. We present this representation along with how to manipulate and render large scale datasets on post analysis machines. We show that our approach is a good trade off between rendering quality and interactive speed and provides uncertainty quantification for the information that is lost.

1 INTRODUCTION

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, 14, 34]. 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 [7, 9, 14, 31]. Among the different techniques, the imaged-based approach has emerged as a promising method for in-situ visualization and analysis [2, 26, 27]. 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, 27], 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 [7, 12, 25, 30]. 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 [29] 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 paper is organized as follows. Section 2 discusses related work. Section 3 is an overview of our approach. Section 4 describes our proxy generation algorithm. The rendering algorithm on the post analysis is discussed in Section 5. Section 6 presents our results. Section 7 provides a discussion of our approach. Finally, Section 8 concludes the paper and provides future work.

2 RELATED WORK

Distribution-based data proxy: A traditional and commonly used approach is to down-sample the volume dataset to a resolution that is compatible with the target platform. Although this approach is simple, straightforward, and results in huge data reductions, it suffers from extensive information loss that is difficult to quantify and convey to the end user. In addition, visualization and analysis tasks suffer from aliasing and inconsistent artifacts [12, 24, 33]. Recently it has been shown that a distribution based approach is able to summarize a large amount of data into a small and compact proxy and provide a quantification of uncertainty that would be very helpful

*e-mail: wang.3182@osu.edu
†e-mail:shareef.1@osu.edu
‡e-mail:shen.94@osu.edu

2165-8773/18/$31.00 ©2018 IEEE
DOI 10.1109/PacificVis.2018.00013

in the post data analysis. Thompson et al. [25] presented Hixels where each sample point stores a histogram of local values within a voxel and used these for various data analysis tasks. Liu et al. [12] improve on this idea by using a Gaussian Mixture Model to represent each distribution in a local area. Dutta et al. [7] also partitioned the dataset into local blocks and modeled the data in each local block with the Gaussian Mixture Model in an in-situ simulation to save the storage and I/O time. Dutta et al. [8] and Wang et al. [30] proposed approaches to minimize spatial information loss in an object-based distribution representation to provide better quality in the analysis.

Image-based approach: Image-based approaches are particularly suited for visualization of large scale data where a proxy is constructed from the original dataset. Tikhnova et al. [26–28] store image slices that approximate pixel color when the scientist changes the new transfer function. Ye el al. [32] propose an in-situ simulation technique which stores the image-based depth map for isosurfaces. The posterior feature extraction and tracking is enabled from their depth map proxies. [2, 3, 19] create a "Cinema" database from the large scale dataset where snapshots are taken from different views along with stored material information. Once the database is constructed, scientists view and analyze the dataset from the stored images. Meyer et al. [15] define a light field to store images of different materials from the dataset for volume rendering. View-dependent approaches have focused on both data reduction and fast volume rendering by re-organizing the data along the pixel rays [17, 20, 23]. These approaches can bound storage size well. Though, they have limited capabilities for transfer function exploration, which is a key analysis tool. From a single view, transfer function manipulation is able to show occluded features. Those approaches that allow transfer function exploration do not quantify the errors that exist in their approximations, which could be misleading to scientists.

Uncertain data visualization: Both ensemble datasets and the small size proxies generated from the large datasets contain the uncertainty. Reconstructing the missing data and visualizing the uncertain data become challenging and interesting topics. Schlegel et al. [22] point 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 propose a method which uses Gaussian process regression to interpolate uncertain data and gain a continuous uncertainty field for visualization. Liu et al. [12] generate animation by keeping re-sampling the samples from the uncertain data over time to convey the degree of uncertainty at different spots in the dataset. Sakhaee and Entezari [21] propose 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 the sampling location for volume rendering. Athawale and Entezari [4] propose a uncertain level-crossing algorithm to compute the field of the occurrence probability of a isovalue to visualize a uncertainty dataset.

3 System Overview

Figure 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 scientist 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 [13]. Our algorithms are implemented on the VTK-m library [16]. 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.

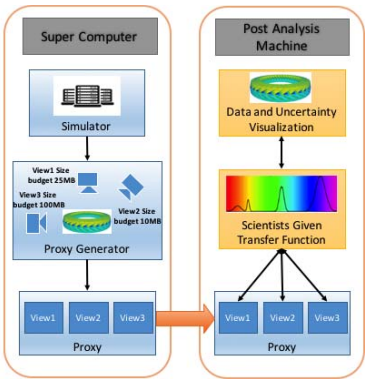


Figure 1: An overview of our proposed technique.

4 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 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 4.1.

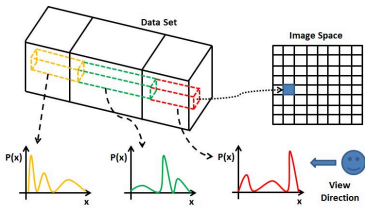


Figure 2: An illustration of our proxy where the image plane is shown on the upper right. Each pixel, such as the pixel shown in blue, stores a depth ordered list of sub-frustums (shown on the left). A distribution is associated with each sub-frustum containing a summary of the information within the sub-frustum.

4.1 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 scientist. 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 [29] which is shown in Equation 1.

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

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 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 ≥ 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 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 1 is expensive, we can use Equation 2 to quickly update the entropy when only a single bin changes and given the entropy of the histogram before the update.

$$\begin{aligned} E_{n+1} = \frac{S_n}{S_n + c}(E_n - (-\frac{f_n^b}{S_n} log \frac{f_n^b}{S_n})) \\ -(S_n - f_n^b)(\frac{1}{S_n + c} log \frac{S_n}{S_n + c}) \\ +(-\frac{f_n^b + c}{S_n + c} log \frac{f_n^b + c}{S_n + c}) \end{aligned} \quad (2)$$

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 2 is derived from the fundemental Shannon entropy equation [29]. We use it to update the entropy at Line 10 in Algorithm 1. This entropy updating is a constant time computation. In addition, the table for logarithm computation is pre-computed for Equation 2. Thus, the process to subdivide the pixel frustum does not introduce too much overhead when generating the proxy.

4.2 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 4.1.

To compute the desired entropy threshold for the given size budget, a straightforward approach is to simply use the algorithm in Section 4.1 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.

4.3 Proxy Data Structure

Figure 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 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.

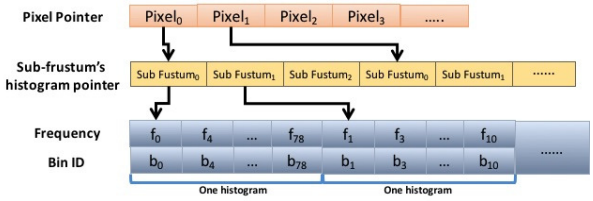


Figure 3: The proxy data structure for one view and one time step.

5 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

$$s_{f_k^p} = s_p * \left(\frac{FS(f_k^p)}{\sum_{i=1}^{K^p} FS(f_i^p)}\right) \tag{3}$$

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.

5.1 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 scientist 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 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.

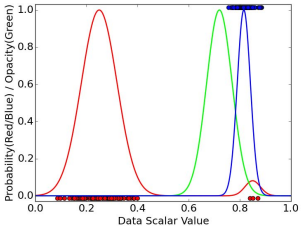


Figure 4: The green curve is the opacity function. The red dots are 100 samples drawn from a data distribution (red curve). Most red dots have no significant opacity, so most of the sampling time is wasted. The blue dots are 100 samples drawn from the importance distribution (blue curve), which incorporates both the data distribution (red curve) and the opacity function (green curve) by Equation 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.

Our technique uses an importance distribution, which is shown by the blue curve in Figure 4. This distribution increases the chance to retrieve samples in the data that have high opacities. Equation 4 shows how to compute the importance distribution from the given opacity function and a pixel sub-frustum's distribution.

$$h_I(b) = h(b) * O(b) \tag{4}$$

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 4.

5.2 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, and using Equation 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.

$$s'_{f_k^p} = s_{f_k^p} * \left(\frac{E(h_I)}{E^{max}}\right)^c \tag{5}$$

where $s_{f_k^p}$ is the re-sample count for the sub-frustum f_k^p from Equation 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 5 can be rewritten in the following way

$$s'_{f_k^p} = s_{f_k^p} * (\frac{nzb}{B})^c \tag{6}$$

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.

5.3 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 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.

$$A_{f_k^p} = 1 - \prod_{b=1}^{B} (1 - O(b))^{s_{f_k^p} * h^N(b)} \tag{7}$$

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 8 shows the opacity of the sub-frustum when we use the adjusted re-sample count and draw samples from the importance distribution.

$$A'_{f_k^p} = 1 - \prod_{b=1}^{B} (1 - O'(b))^{s'_{f_k^p} * h_I^N(b)} \tag{8}$$

where $s'_{f_k^p}$ is the adjusted re-sample count in Section 5.2 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 9.

$$O'(v_b) = 1 - (1 - O(v_b))^{(\ell/\ell')} \tag{9}$$

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.

5.4 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 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 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

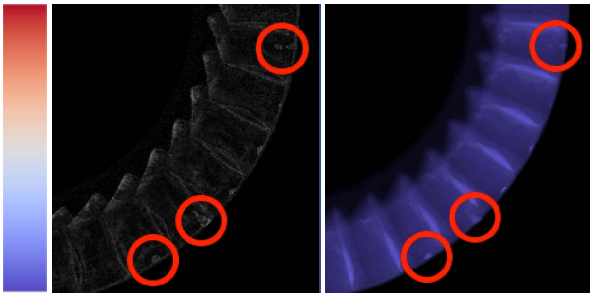
Algorithm 2 Proxy rendering

▷ p: a pixel which contains histograms of sub-frustums
▷ s^p: given re-sample count of p
▷ c: parameter to adjust re-sample count
▷ T^c and T^o: color and opacity function
procedure ProxyRendering(p, s^p, c, T^c, T^o)
    $color$ = (0,0,0,0)
    $pxlFS = PixelFreqSum(p)$ ▷ *compute freqency sum of all histograms on p*
    $K^p = GetNumOfHist(p)$
    for $k = 1 \rightarrow K^p$ **do**
        $h = GetHist(p,k)$ ▷ *get the k-th histogram on p*
        $h_I = ImpDistr(h, T^o)$ ▷ *compute the importance distribution*
        $s = (s^p * \frac{FreqSum(h)}{pxlFS}) * (\frac{Nzb(h_I)}{Bins(h_I)})^c$ ▷ *adjust the re-sample count*
        $T_k^o = AdjustOpacityFunc(T^o, s^p, s)$ ▷ *modulate the opacity function*
        for $i = 1 \rightarrow s$ **do**
            $v = ReSample(h_I)$
            $colorV = T^c(v)$
            $opacityV = T_k^o(v)$
            $color = blend(color, colorV, opacityV)$
        end for
        $CheckEarlyRayTermination(color)$
    end for
    return $color$
end procedure

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 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.

Algorithm 3 Static uncertainty visualization

```
▷ p: a pixel which contains histograms of sub-frustums
▷ s^p: given re-sample count of p
▷ T^c and T^o: color and opacity function for data rendering
▷ T^c_U: color map for uncertainty visualization
procedure StaticUncertaintyVisualization(p, s^p, T^c, T^o, T^c_U)
    color = (0,0,0,0)
    pxlFS = PixelFreqSum(p)
    K^p = GetNumOfHist(p)
    for k = 1 → K^p do
        h = GetHist(p,k)
        h_I = ImpDistr(h,T^o)
        h_I^YUV = TransformToYUVSpace(h_I, T^c)      ▷ map scalar to YUV space
        e = ComputeEntropy(h_I^YUV)
        colorU = T^c_U(e)                            ▷ look up color by e
        opacityU = ExpectedOpacity(h, T^o, (s^p * FreqSum(h)/pxlFS))
                  ▷ compute the opacity of a sub-frustum in data rendering image
        color = blend(color, colorU, opacityU)
        CheckEarlyRayTermination(color)
    end for
    return color
end procedure
```



(a) (b) (c)

Figure 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).

6 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 [7]. 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).

6.1 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 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 1.

The time reported in Table 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 4.2) 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 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.

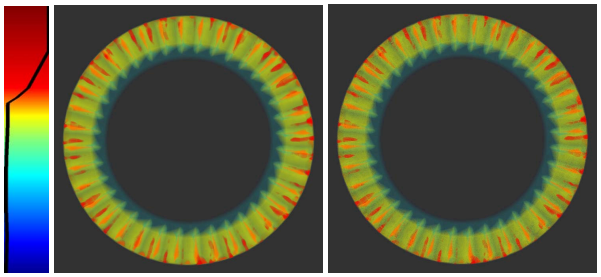
6.2 Image Quality and Rendering Time

We evaluate the quality of rendering images in this section. We choose one view from the proxies generated in Section 6.1 to evaluate. The view of each dataset is shown in Figure 6, 7, 8 and 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. 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 6, 7, 8 and 9.

Table 1: The proxy size and generation time.

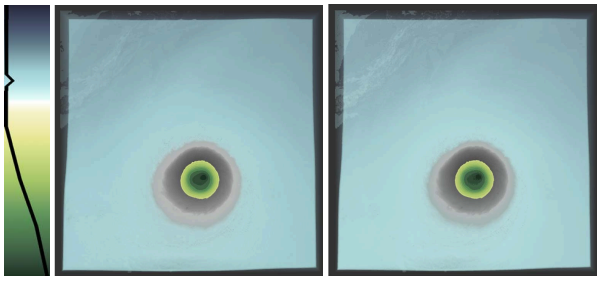
	Size			Time (Average)		
	Size Budget (Per view/time step proxy)	Raw Data (Total)	Proxy (Total)	Proxy Generation	Entropy Selection (1st time step)	Entropy Selection (From 2nd time step)
Isabel	50MB	385GB	14.2GB	19.6s	42.3s	11.4s
Plume	50MB	189GB	8.9GB	23.2s	125.2s	17.7s
Combustion	50MB	407GB	15.2GB	27.6s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

To evaluate the image quality, we calculate the peak signal-to-noise ratio (PSNR) [10] 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 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 6, 7, 8 and 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 6, 7, 8 and 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.



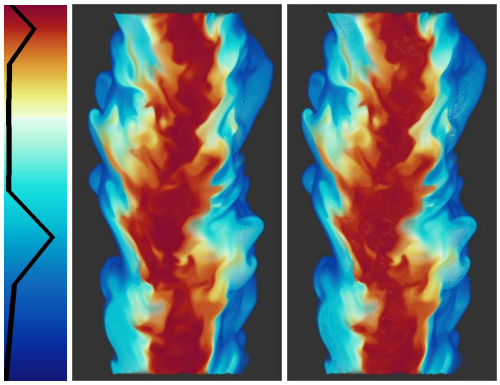
(a) (b) (c)

Figure 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 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.



(a) (b) (c)

Figure 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.

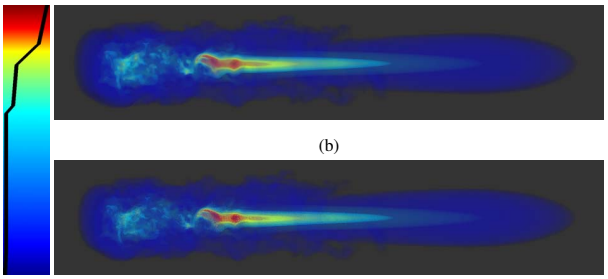
6.3 Importance Sampling

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 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 6.2. 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 decreases 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 10 indicates the re-sample count is the sames as the re-sample count used in Section 6.2.

7 Discussion

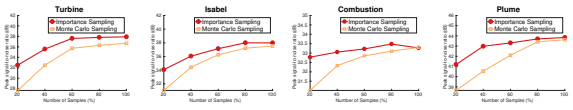
7.1 Transfer Function

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 6.2, but give different opacity functions to run the experiment. Table 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 6.2. The opacity function 2 (OF2) is designed to give all scalar values high opacity (0.85). It means only



(a) (b) (c)

Figure 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.



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

Figure 10: Image quality comparison between Importance sampling and basic Monte Carlo sampling for varying numbers of re-sample count.

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 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. 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 [16]. 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 2.

7.2 Bin Count of Histograms

The bin count of histograms directly affects the image quality, preprocessing time and rendering time. Figure 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 6, 7, 8 and 9 for the four datasets.

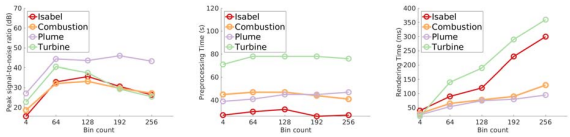
In Figure 11(a), we generate proxies with different bin counts by giving each time step proxy a 50MB budget. Transfer functions in

Table 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)

Figure 6, 7, 8 and 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 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 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 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.



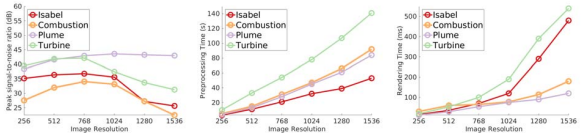
(a) (b) (c)

Figure 11: The impact of different histogram's bin counts on the image quality, proxy generation time and rendering time.

7.3 Image Resolution

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 7.2. Figure 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 his-

togram. This also slightly increases the uncertainty and error in the re-sample process. Figure 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.

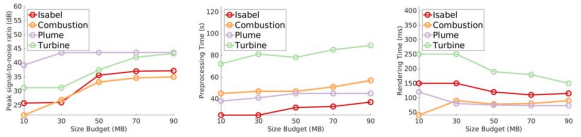


(a) (b) (c)

Figure 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.

7.4 Size Budget

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 7.2. Figure 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 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 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.



(a) (b) (c)

Figure 13: The impact of different size budgets on the image quality, proxy generation time and rendering time.

7.5 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 [6] 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 [5, 11]. 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 7.2, 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 6.

8 Conclusion and Future Work

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 is 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.

In future work, we will explore techniques to add depth order information into our proxy. This can further increase the visualization quality of our approach. Although the scientist can select a small

number of views to produce the proxies that cover the scientist's region of interest, smoothly rotating the data is always a preference. We will investigate how to observe the dataset from neighboring view angles of a proxy. Our current ray casting algorithm is a single node implementation based on VTK-m. When processing even larger datasets, such as at a petabyte scale, the efficient and well-scalable multi-node ray casting implementation will be advantageous. We could develop our algorithm on the top of the ray casting technique in a distributed-system [18]. About the static uncertainty visualization, the granularity of the histogram in YUV space for color randomness computation is an interesting parameter to study in the future. The current test datasets are up-sampled datasets. We will seek the collaborations with scientists who has simulators to produce large scale raw datasets and apply our approach on these datasets. In addition, we can also ask the experts' feedback about whether the image quality from our proxy is sufficient for their real data analysis needs.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants IIS- 1250752, IIS-1065025, and US Department of Energy grants DE- SC0007444, DEDC0012495,program manager Lucy Nowell.

REFERENCES

[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] 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*, pp. 424–434. IEEE Press, 2014.

[3] J. Ahrens, J. Patchett, A. Bauer, S. Jourdain, D. H. Rogers, M. Petersen, B. Boeckel, P. OLeary, P. Fasel, and F. 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] T. Athawale, E. Sakhaee, and A. Entezari. Isosurface visualization of data with nonparametric models for uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):777–786, 2016.

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

[6] 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), 2012 IEEE Symposium on*, pp. 113–120. IEEE, 2012.

[7] S. Dutta, C.-M. Chen, G. Heinlein, H.-W. Shen, and J.-P. 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.

[8] S. Dutta, J. Woodring, H.-W. Shen, J.-P. Chen, and J. Ahrens. Homogeneity guided probabilistic data summaries for analysis and visualization of large-scale data sets. In *Pacific Visualization Symposium (PacificVis), 2017 IEEE*, pp. 161–170. IEEE, 2017.

[9] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. 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*, pp. 89–96. IEEE, 2011.

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

[11] G. Ji and H.-W. Shen. Dynamic view selection for time-varying volumes. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

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

[13] L.-t. Lo, C. Sewell, and J. P. Ahrens. Piston: A portable cross-platform framework for data-parallel visualization operators. In *EGPGV*, pp. 11–20, 2012.

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

[15] M. Meyer, H. Pfister, C. Hansen, C. Johnson, M. Meyer, H. Pfister, C. Hansen, and C. Johnson. Image-based volume rendering with opacity light fields. *no. UUSCI-2005-002. Tech Report*, 2005.

[16] 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. *IEEE computer graphics and applications*, 36(3):48–58, 2016.

[17] K. Mueller, N. Shareef, J. Huang, and R. Crawfis. Ibr-assisted volume rendering. *Late Breaking Hot Topics of Visualization99*, pp. 5–9, 1999.

[18] P. A. Navrátil, H. Childs, D. S. Fussell, and C. Lin. Dynamic scheduling for large-scale distributed-memory ray tracing. Technical report, Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA (US), 2012.

[19] P. OLeary, J. Ahrens, S. Jourdain, S. Wittenburg, D. H. Rogers, and M. Petersen. Cinema image-based in situ analysis and visualization of mpas-ocean simulations. *Parallel Computing*, 55:43–48, 2016.

[20] C. Rezk-Salama, S. Todt, and A. Kolb. Raycasting of light field galleries from volumetric data. In *Computer Graphics Forum*, vol. 27, pp. 839–846. Wiley Online Library, 2008.

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

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

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

[24] 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*, 20(12):2417–2426, 2014.

[25] 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), 2011 IEEE Symposium on*, pp. 23–30. IEEE, 2011.

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

[27] A. Tikhonova, C. D. Correa, and K.-L. Ma. Explorable images for visualizing volume data. In *PacificVis*, pp. 177–184, 2010.

[28] 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, pp. 783–792. Wiley Online Library, 2010.

[29] C. Wang and H.-W. Shen. Information theory in scientific visualization. *Entropy*, 13(1):254–273, 2011.

[30] 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), 2017 IEEE*, pp. 161–170. IEEE, 2017.

[31] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum*, vol. 30, pp. 1151–1160. Wiley Online Library, 2011.

[32] Y. C. Ye, Y. Wang, R. Miller, K.-L. Ma, and K. Ono. In situ depth maps based feature extraction and tracking. In *Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on*, pp. 1–8. IEEE, 2015.

[33] H. Younesy, T. Möller, and H. Carr. Improving the quality of multi-resolution volume rendering. In *EuroVis*, pp. 251–258. Citeseer, 2006.

[34] 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*, 30(3):45–57, 2010.