Eurographics Symposium on Parallel Graphics and Visualization (2011)
T. Kuhlen, R. Pajarola, and K. Zhou (Editors)

A Preview and Exploratory Technique for Large-Scale Scientific Simulations

Anna Tikhonova[*1] and Hongfeng Yu[†2] and Carlos D. Correa[‡3] and Jacqueline H. Chen[§2] and Kwan-Liu Ma[¶1]

[1]University of California, Davis
[2]Sandia National Laboratories
[3]Lawrence Livermore National Laboratory

Abstract

Successful in-situ and remote visualization solutions must have minimal storage requirements and account for only a small percentage of supercomputing time. One solution that meets these requirements is to store a compact intermediate representation of the data, instead of a 3D volume itself. Recent work explores the use of attenuation functions as a data representation that summarizes the distribution of attenuation along the rays. This representation goes beyond conventional static images and allows users to dynamically explore their data, for example, to change color and opacity parameters, without accessing the original 3D data. The computation and storage costs of this method may still be prohibitively expensive for large and time-varying data sets, thus limiting its applicability in the real-world scenarios. In this paper, we present an efficient algorithm for computing attenuation functions in parallel. We exploit the fact that the distribution of attenuation can be constructed recursively from a hierarchy of blocks or intervals of the data, which is a highly parallelizeable process. We have developed a library of routines that can be used in a distance visualization scenario or can be called directly from a simulation code to generate explorable images in-situ. Through a number of examples, we demonstrate the application of this work to large-scale scientific simulations in a real-world parallel environment with thousands of processors. We also explore various compression methods for reducing the size of the RAF. Finally, we present a method for computing an alternative RAF representation, which more closely encodes the actual distribution of samples along a ray, using kernel density estimation.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

The growing power of parallel supercomputers has enabled scientists to model increasingly complex physical phenomena. Scientific simulations generate output that is usually volumetric, large-scale, multi-dimensional, and time-varying. Visualizing such data sets may be a time-consuming and resource-intensive task, especially if the data is unstructured or the rendering is performed at high-resolution or employs an expensive lighting model, such as global illumination. This problem is exacerbated when the data is visualized remotely or when the rendering or interaction with the data is performed on low-end devices.

A large body of work is dedicated to reducing the amount of data that needs to be generated, stored, transferred over the network, and accessed to visualize a data set. Some of the recent and practical methods include in-situ and distance visualization. When the data is visualized in-situ, it is rendered while the simulation is running. In this case, only an image of the data is stored and not the 3D data itself, which results in significant space savings. When the visualization is performed remotely, a powerful cluster performs the expensive rendering computations. Then, only the result of these com-

* tikhonov@cs.ucdavis.edu
† hyu@sandia.gov
‡ correac@llnl.gov
§ jhchen@sandia.gov
¶ ma@cs.ucdavis.edu

DOI: 10.2312/EGPGV/EGPGV11/111-120

putations - an image - is served to the user. In both of these approaches, the user is presented with a static image. The lack of explicit 3D information in a static image prevents the user from changing the properties of the data depicted in it. For example, a user cannot modify the transfer function after an image has been rendered.

A number of existing solutions enable exploration in transfer function space by generating a collection of images, where each image is rendered with different transfer function settings. Transfer function changes are then simulated by synthesizing new images from the training set. Other image-based rendering solutions focus on providing fast transfer function changes by caching view-dependent samples. Both of these approaches are not practical for in-situ and distance visualization. Rendering each time step multiple times with different parameters can be time-consuming, especially if the users expect the ability to make transfer function changes of fine granularity or within a large range. Caching view-dependent samples for each time step may also be prohibitively expensive in terms of storage space.

Another approach to preview or exploratory visualization is to use intermediate representations of volume data. As an example, Tikhonova et al. [TCM10b] develop a novel approach for interactive visualization and exploration based on Ray Attenuation Functions (RAF), which encode the distribution of samples along each ray. This representation can be thought of as a compact cache of attenuation values. The RAF not only reduces the computational complexity of data visualization and exploration, but also enables users to defer these operations to a later time. Even though this representation is very compact, it is powerful enough to enable the users to dynamically filter information presented on the screen. For example, users can modify opacity and color of the features depicted in volume rendered images. However, the application of this approach to in-situ and remote visualization demands efficient solutions to the following two challenges: (1) a compact intermediate data representation that can be efficiently stored on disk or distributed over a network; (2) an efficient mechanism for obtaining such a representation, even for large-scale data.

In this paper we address both of these challenges. We provide an efficient and practical algorithm for computing attenuation functions in parallel for very large datasets. We extend the RAF representation to interval attenuation functions (IAF). The IAF are computed for every part of a large data set in a distributed environment and then the local results are combined into a global data representation. Thus, the running time of the RAF computation algorithm is reduced from $O(n^3)$ to $O(n^3/p+C)$, where n is the number of voxels, p is the number of processors, and C is compositing ($O(log(p))$ steps) time. Each processor performs computations on $O(n^3/p)$ voxels. We demonstrate that this process can be implemented in a real-world parallel environment with relative ease. We develop a library of routines that can be used in the distance visualization scenario or can be called directly from a simulation code to generate explorable images in-situ. In addition, since the attenuation functions encode image-like data, they can be stored compactly using conventional data reduction and image compression strategies, such as quantization and lossless PNG compression. We also use statistical modeling of the ray attenuation functions to obtain compression that better preserves the quality of the information stored in the RAF. We observe that attenuation functions are weighted histograms that represent the distribution of attenuation along a ray traversing a volumetric object. Therefore, we can leverage parametric and non-parametric models for probability distribution functions to encode the RAF more efficiently.

The ability to obtain the RAF faster and to encode them more efficiently is crucial for in-situ and remote visualization. While simulation time dominates the computational time of large-scale scientific studies, our methodology allows scientists to allocate computation time to pre-visualization and storage of attenuation functions. We envision our contribution will enable scientists to store more of the data they are currently forced to discard. Since the intermediate representation is explorable, we also expect that the combination of the intermediate representations and the saved volume data will enrich their exploratory experience.

Using our technique, users can easily steer a simulation by computing compact representations of their large-scale data while the simulation is running. They can quickly experiment with different transfer function parameters on their laptop or mobile device. Once they are satisfied with the settings, they can instruct the system to use them for the rest of the simulation. Our technique can also be used for remote or distance visualization. Instead of waiting for a remote cluster to volume render an image and to transfer it over the network, the users can work with an intermediate representation and instruct the remote system to volume render a high-resolution image only when absolutely necessary.

2. Related Work

Data Reduction Techniques. Previous solutions focus on reducing the amount of data that needs to be generated, stored, transferred over the network, and accessed to visualize and to explore large-scale data sets. Data reduction techniques are most commonly used for reducing the data to a more compact representation for efficient subsequent analysis and visualization. Some of the most common methods include subsampling in the spatial or temporal domains (i.e., skipping time steps), scalar and vector quantization, transform-based compression, and feature extraction. The choice of an appropriate method depends on the storage space constraints, network bandwidth, the acceptable amount of error and precision in visualization, and sometimes on the parallel domain decomposition used in a particular simulation code. It is also possible to employ several data reduction methods together (e.g., quantization and subsampling) to reduce the data to an even greater extent.

Typically, data reduction is a post-processing step. The output size of scientific simulations can also be reduced in-situ. The in-situ approach is particularly effective because it decreases resource requirements for all subsequent data transmission, storage, and processing tasks. However, the reduced volume data still has to be visualized using a 3D algorithm, such as direct volume rendering, which requires many (often redundant) accesses to volume data. Another solution is to extract physically-based features of interest, which are generally much smaller than the original data. This requires extensive domain knowledge, especially for time-varying data sets. An overview of data reduction methods in the context of in-situ processing is provided in [MWYT07].

Image-based Rendering. Image-based rendering methods provide an alternative to storing 3D data. Most of these methods generate a collection of images or intermediate layers that can be used to explore one or several volume rendering parameters. For example, He et al. [HHKP96] and Marks et al. [MAB*97] propose the pre-computation of a collection of images, each of which contains a different combination of opacity and color settings. Users can browse such a collection as an alternative to direct manipulation in 3D space. To alleviate the need for generating a large number of images, these approaches can be improved with automated analysis [WQ07] or effective user interfaces [RPSH08, RBG07]. An alternative to exploring the image-space of volume rendering is to use intermediate results. These can be cached results from volume data [MCP91, LP03], compressed runs of structured [SCM03] or unstructured [SLSM06] volume data, or alpha layers extracted from a set of images [TCM10a]. Although in the same spirit of these approaches, other layer-based techniques, such as semantic layers [RBG07], opacity [RSK06] and feature peeling [MMG07] are designed for improving the visibility of features in 3D volume rendering. As image-based representations, these can also be useful for caching intermediate rendering results.

These approaches are not always practical for in-situ and distance visualization. The pre-computation of a training set for multiple opacity and color combinations is prohibitively expensive for these scenarios. Peeling approaches require access to the full 3D data when opacity values change, and image caches get invalidated when opacity changes result in a cache miss. In this paper, we focus on attenuation histograms as intermediate representations and show that they can be efficiently computed for in-situ visualization, are compact, and can be used to explore opacity and color mappings of volume data.

Parallel and Distributed Rendering. In this paper, we focus on large-scale data sets, generated by parallel scientific simulations. Our goal is to develop an intermediate representation as well as an efficient algorithm to compute it in the parallel or distributed scenario. Typically, parallel rendering solutions consist of the following stages: data partition and distribution, rendering, image compositing, and image delivery. The three well-known parallel rendering approaches include sort-first, sort-middle, and sort-last, defined by Molnar et al. [MCEF94]. Due to its scalability and the simplicity of its load balancing strategy, sort-last rendering is widely used by the visualization community [AP98, EP07], to cite a few. Sort-last rendering requires a final image compositing stage which could be very expensive due to the amount of inter-processor communication involved. The most commonly used image compositing methods, developed for sort-last parallel volume rendering, are direct-send [Neu94] and binary swap [MPHK93]. These two methods require different numbers of messages exchanged among the N compositing processors. The direct-send method requires all-to-all communication and the number of exchanged messages is bounded by $O(N^2)$. The binary swap method balances the compositing workload using a binary tree style compositing process, thus reducing the number of required messages to $O(Nlog(N))$. However, the binary swap method needs N to be a power of two. In this paper, we use the 2-3 swap parallel image compositing algorithm by Yu et al. [YWM08], which is a generalization of the binary swap algorithm and a subset of the Radix-k algorithm [PGR*09]. The 2-3 swap algorithm requires only $O(Nlog(N))$ messages and can use an arbitrary number of processors. The algorithm involves a multistage process and partitions the processors into groups using the 2-3 compositing tree. At any image-compositing stage, a processor communicates only with the other processors in its group, thus reducing the number of message exchanges. The 2-3 swap method scales well to thousands of processors and has been applied in the *in-situ* visualization system for real-world large-scale combustion simulations [YWG*10].

3. Parallel Preview and Exploration Framework

A previewing and exploration system for visualization should enable users to quickly visualize their data and to immediately see the result of a change in visualization parameters. These capabilities are essential for steering a simulation in the in-situ scenario. Remote visualization users can experiment with different visualization parameters on their local machine and only request the system to render a high-resolution image, when absolutely necessary.

As described in Sec. 2, many existing approaches rely on a collection of images to achieve an efficient image-based solution, but offer limited exploration capabilities. In our approach, we use an intermediate representation of volume data, which can be thought of as a view-dependent summary of attenuation information. This allows us to simulate new images with different rendering parameters, without accessing the original data.

The algorithm in [TCM10b] is implemented on the GPU; however, the computation of the RAF is still sequential for each ray. Therefore, the algorithm has limited application to large-scale visualization, where a volume might not fit entirely in CPU or GPU memory. In this paper, we develop an efficient and practical algorithm for computing the RAF in parallel, which assumes that the data is split among p processors in volume segments of arbitrary shape and size. We also show that the resulting functions can be stored efficiently using even more compact representations. In particular, we exploit existing image compression, data reduction, and statistical modeling techniques to reduce the storage requirement of the RAF. We also introduce a method for constructing an alternative RAF representation through the use of KDE.

To develop a parallel algorithm, we introduce *Interval Attenuation Functions (IAF)*, which summarize the attenuation information for a given ray segment. We demonstrate that the full RAF can be built in parallel from a collection of IAF.

3.1. Interval Attenuation Functions (IAF)

The RAF approximate the volume rendering integral by summarizing the attenuation due to each intensity value for a number of data ranges. According to the volume rendering integral [Max95], the color resulting from compositing volume data is:

$$C = \int_0^N C(t)\tau(t)e^{-\int_0^t \tau(s)ds}dt, \tag{1}$$

where $C(t)$ is the radiance or color and $\tau(t)$ is the attenuation of a sample t along a ray, with t ranging from 0 to N. When discretized, this equation becomes:

$$C = \sum_{i}^{N} C(i)\alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j)), \tag{2}$$

where $\alpha(i)$ is the opacity of a sample along the view direction and $\prod_{j=0}^{i-1}(1-\alpha(j))$ is the attenuation due to all sample points in front of a sample i, and N is the number of samples along a ray. Grouping the intensity values into discrete bins, the above equation can be approximated as follows:

$$C \approx \sum_{k=1}^{N_{bins}} C(k) \sum_{\{i|i=0,1,\ldots,N_{bins}\}\,\mathrm{AND}\,bin(i)=k} \alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j)),$$

where $bin(i)$ is a function, assigning an intensity value of a sample i to a bin k, and N_{bins} is the number of bins. The inner sum, the *Ray Attenuation Function* (RAF), describes the distribution of attenuation along a ray with respect to an intensity value. Fig. 1 shows the process of accumulating the attenuation of intensity values into a finite set of bins.

To understand our IAF computation algorithm, let's take another look at the light transport equation for a volumetric model. For a given interval (l,h) in the intensity domain, the accumulated attenuation is the combined contribution of all samples with an intensity value that falls within that interval:

$$a(l,h) \approx \sum_{\substack{i=0\\ l<V(i)\le h}}^{N} \alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j)), \tag{3}$$

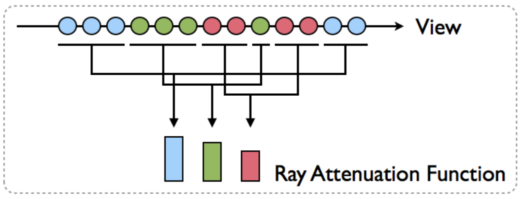

Figure 1: *Ray Attenuation Functions (RAF) are obtained by accumulating the attenuation of intensity values grouped into a finite set of bins.*

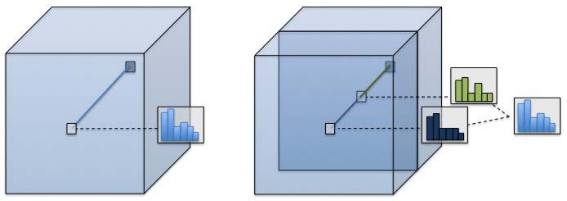

Figure 2: *Computation of Interval Attenuation Functions. Left: In a sequential framework, the attenuation function must be computed for an entire ray. Right: In a parallel framework, the attenuation function can be computed in parallel, for several intervals. The IAF are then combined to create the full attenuation function.*

where $\alpha(i)$ is the opacity of a sample along a ray, $V(i)$ is the scalar intensity value at that point, and N is the number of samples.

This suggests, that the IAF can be computed hierarchically, where the function for a large interval can be expressed as a linear combination of any number of functions of smaller intervals. Therefore, the ray in Eq. 1 can be split in two intervals ($t \in [0,M)$ and $t \in [M,N)$) and the equation can be rewritten as:

$$\begin{aligned} C \;=\; & \int_0^{M-1} C(t)\tau(t)e^{-\int_0^t \tau(s)ds}dt \;+ \\ & e^{-\int_0^{M-1}\tau(s)ds}\int_M^N C(t)\tau(t)e^{-\int_M^N \tau(s)ds}dt \end{aligned} \tag{4}$$

The above relationship can be discretized as a sum of products of attenuations of N samples along a ray. In this case, the ray is split into two sets of samples, where $i \in \{0,\ldots,M-1\}$ and $i \in \{M,\ldots N\}$:

$$\begin{aligned} C \;=\; & \sum_{i=0}^{M-1} C(i)\alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j)) \;+ \\ & \prod_{j=0}^{M-1}(1-\alpha(j))\sum_{i=M}^{N} C(i)\alpha(i)\prod_{j=M}^{i-1}(1-\alpha(j)) \end{aligned} \tag{5}$$

These attenuations can be grouped into bins, in order to approximate color, as shown in [TCM10b]. Moreover, we can define an interval attenuation function for an interval $[L,M]$, which adds up the contributions of all samples in the interval $[L,M]$ that fall within a bin k:

$$\mathsf{F}_{[L,M]}(k) = \sum_{\substack{i=L \\ Bin(i)=k}}^{M} \alpha(i) \prod_{j=L}^{i-1} (1-\alpha(j)) \tag{6}$$

so that the composited color along a ray interval can be approximated as:

$$\sum_{i=L}^{M} C(i)\alpha(i) \prod_{j=0}^{i-1} (1-\alpha(j)) \approx \sum_{k=1}^{N_{bins}} C(k)\mathsf{F}_{[L,M]}(k), \tag{7}$$

Therefore, Eq. 4 becomes:

$$\begin{aligned} \sum_{k=1}^{N_{bins}} C(k)\mathsf{F}(k) &= \sum_{k=1}^{N_{bins}} C(k)\mathsf{F}_{[0,M-1]}(k) + \\ & \prod_{j=0}^{M-1}(1-\alpha(j)) \sum_{k=1}^{N_{bins}} C(k)\mathsf{F}_{[M,N]}(k), \end{aligned} \tag{8}$$

and then it becomes clear that the attenuation function can be defined recursively as:

$$\mathsf{F}_{[0,N]}(k) = \mathsf{F}_{[0,M-1]}(k) + \prod_{j=0}^{M-1}(1-\alpha(j))\mathsf{F}_{[M,N]}(k) \tag{9}$$

In other words, the IAF of an interval, composed of two sub-intervals, can be defined in terms of the IAF of each sub-interval, with the attenuation contributions of the second sub-interval (further into the volume from the eye position) modulated by the *total attenuation* of the first sub-interval. By induction, each sub-interval can be computed as a combination of smaller intervals. Without loss of generality, let us assume there are P sub-intervals. The attenuation of the i^{th} sub-interval is denoted as:

$$\mathsf{A}(i) = \prod_{j=0}^{N_i-1}(1-\alpha(j)), \tag{10}$$

where N_i is the number of samples in a sub-interval. Thus, the total IAF for a bin k can be composited as:

$$\mathsf{F}(k) = \sum_{i=0}^{P} \mathsf{F}(i,k) \prod_{j=0}^{i-1} \mathsf{A}(j), \tag{11}$$

where $\mathsf{F}(i,k)$ is the IAF of the i^{th} sub-interval for a bin k, and can be obtained using Eq. 6.

We can see that the above compositing operation is associative. For example, assume that there are four sub-intervals, their IAF are denoted as F_0, F_1, F_2, F_3, and their attenuation values are denoted as A_0, A_1, A_2, A_3. The accumulated IAF of the first two sub-intervals is:

$$\mathsf{F}_{01} = \mathsf{F}_0 + \mathsf{A}_0\mathsf{F}_1 \tag{12}$$

and the accumulated attenuation value of the first two sub-intervals is:

$$\mathsf{A}_{01} = \mathsf{A}_0\mathsf{A}_1 \tag{13}$$

Similarly, the accumulated IAF of the last two sub-intervals is:

$$\mathsf{F}_{23} = \mathsf{F}_2 + \mathsf{A}_2\mathsf{F}_3 \tag{14}$$

Thus, the, total RAF calculated from F_{01} and F_{23} is:

$$\begin{aligned} \mathsf{F} &= \mathsf{F}_{01} + \mathsf{A}_{01}\mathsf{F}_{23} \\ &= \mathsf{F}_0 + \mathsf{A}_0\mathsf{F}_1 + \mathsf{A}_0\mathsf{A}_1\mathsf{F}_2 + \mathsf{A}_0\mathsf{A}_1\mathsf{A}_2\mathsf{F}_3, \end{aligned} \tag{15}$$

which is same as the result obtained by evaluating Eq. 11 in sequential fashion. We can also see that the sequence of the operands of the compositing operation is in the visibility order, which can not be changed, i.e. the compositing operation is not commutative.

3.2. Parallel Algorithm for Attenuation Functions

Based on the above formulation, we can build a parallel algorithm for computing attenuation functions. Our algorithm reduces the running time of the computation of ray attenuation functions from $O(n^3)$ to $O(n^3/p+C)$, where n is the number of voxels, p is the number of processors, and C is compositing time, where compositing is performed in $O(log(p))$ steps. Each processor performs computations on $O(n^3/p)$ voxels. The derivation of our block-based algorithm and the details of our parallel algorithm are as follows.

We construct the RAF on the same machine used to run a given simulation. Thus, we directly use the domain decomposition used in a simulation to compute the IAF, thus avoiding unnecessary data replication. We assume the data is partitioned in a block-based fashion among p processors, which is a typical scenario in our target simulations [CCdS*09]. To achieve seamless IAF results, the data along the partition boundaries is duplicated among the neighboring processors. Each processor needs to propagate boundary information to its 26 neighboring processors in 3D. We employ the diagonal communication elimination method [DH01] to minimize the communication costs associated with this task. Using this method, a processor communicates with only 6 of its neighbors for boundary exchange.

After exchanging boundary data, each processor evaluates Eq. 6 on its corresponding intervals to obtain F_i, along with its local accumulated attenuation A_i. After this phase completes, each processor has an attenuation value and an array of IAF values for each pixel.

The next step is to composite the local IAF into the final result. The naive strategy is for a single processor to first gather the local IAF and the attenuation values from all processors, and then to loop through all the pixels to compute the final RAF values for each pixel, using Eq. 11. However, $N-1$ processors are idle during this compositing process and the desired parallelism is least exploited. Instead, we treat the IAF compositing task as a typical vector reduction problem, i.e., we gather the attenuation functions hierarchically. The associative property of the IAF compositing operation allows us to re-parenthesize the compositing operator

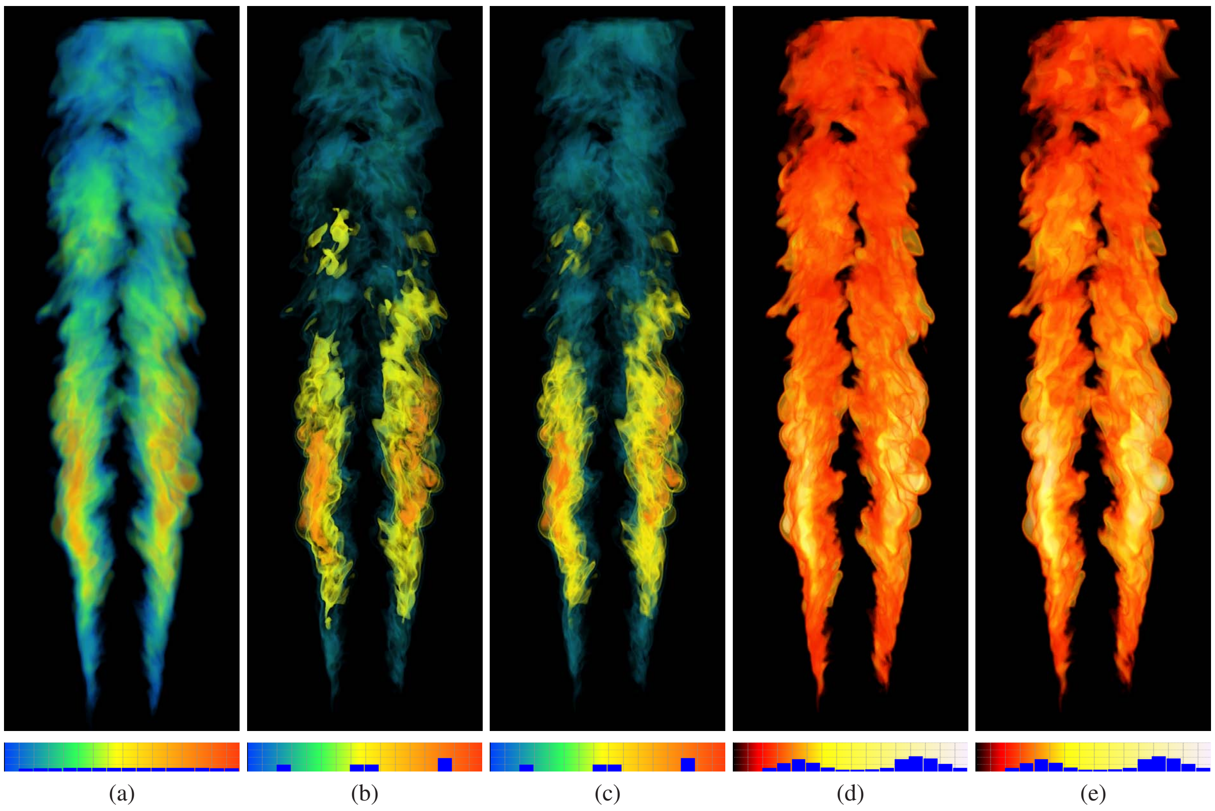

Figure 3: *Transfer function operations for a turbulent combustion simulation (HO_2 variable) using the RAF. (a) Image reconstruction using the attenuation functions with the original transfer function. (b) and (d) Opacity modulation and re-colorization. (c) and (e) Ground truth volume rendered images of (b) and (d), respectively.*

to exploit concurrency. This is similar to the well-studied parallel image compositing problem. From the various solutions for parallel image compositing, we choose the 2-3 swap method by Yu et al. [YWM08]. Rather than propagating RGB tuples, we propagate a tuple of N components of the IAF bins. The hierarchical computation of the attenuation function is depicted below, for four intervals, where $t_1 - t_3$ are levels in the hierarchical reduction:

t_1	t_2	t_3
$(\mathsf{F}_0,\mathsf{A}_0)$		
	$(\mathsf{F}_0+\mathsf{A}_0\mathsf{F}_1,\mathsf{A}_0\mathsf{A}_1)$	
$(\mathsf{F}_1,\mathsf{A}_1)$		
		$(\mathsf{F}_0+\mathsf{A}_0\mathsf{F}_1+\mathsf{A}_0\mathsf{A}_1(\mathsf{F}_2+\mathsf{A}_2\mathsf{F}_3),$ $\mathsf{A}_0\mathsf{A}_1\mathsf{A}_2\mathsf{A}_3)$
$(\mathsf{F}_2,\mathsf{A}_2)$		
	$(\mathsf{F}_2+\mathsf{A}_2\mathsf{F}_3,\mathsf{A}_2\mathsf{A}_3)$	
$(\mathsf{F}_3,\mathsf{A}_3)$,		

4. Results

In our experiments, we used a turbulent combustion simulation performed at Sandia National Laboratories (SNL). Our test environment is the Cray XT5 at the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory (ORNL). The XT5 contains 18,688 compute nodes. Each node contains dual hex-core AMD x86_64 Opteron processors running at 2.6GHz, 16GB memory, and a SeaStar 2+ router. In this combustion simulation, each core is assigned a region of $27 \times 40 \times 40$. We tested three different numbers of cores: 240, 1,920, and 6,480, with the core configuration of $15 \times 8 \times 2$, $30 \times 16 \times 44$, and $45 \times 24 \times 6$, respectively.

Table 1 lists the volume sizes and timing results. All variables for volume data are double floating point (eight bytes). We use 32-bit floating point for the RGBA and depth channels, and for the RAF values. We also use $1,024^2$ image resolution. The timing for simulation, I/O, volume rendering, and RAF computation was measured for one time step. During I/O time, the simulation code stores the simulation data into storage devices. We can see that volume rendering and RAF computation time accounts only for a small fraction of the total simulation and I/O time. For example, if we perform volume rendering and RAF computation at each simulation time step for 6,480 cores and $1,024^2$ image resolution, vol-

Number of processors	240	1920	6480
Simulation time	8.7204	9.3393	9.5573
I/O time	9.4563	26.051	52.565
Total volume rendering time	0.3817	0.6155	0.7359
Boundary voxel exchange	0.0042	0.0059	0.0064
Ray casting	0.0226	0.0148	0.0095
Image compositing	0.3549	0.5948	0.7200
Total IAF computation time	1.2775	1.3938	1.3973
Boundary voxel exchange	0.0026	0.0066	0.0068
IAF construction	0.0806	0.0729	0.0450
IAF compositing	1.1943	1.3143	1.3455

Table 1: *The timing breakdown (in seconds) for different processor counts with $1,024^2$ output image resolution.*

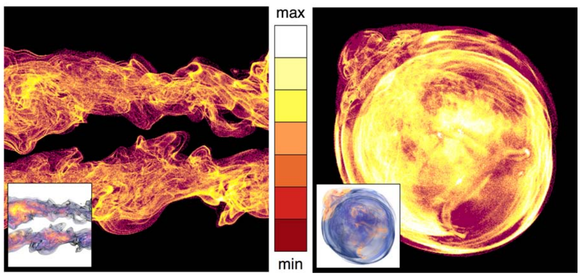

Figure 4: *Reconstruction error (avg. pixel difference) between the images reconstructed from the original RAF vs. images reconstructed from the compressed and then decompressed RAF. We use a heat map to show the variation in error for a combustion (max error: 7/255) and supernova (max error: 8/255) simulations. The error is greater in locations with most overlap of various features.*

ume rendering time is approximately 7.70% of the simulation time, and the RAF computation time is approximately 14.62% of the simulation time; I/O time is more than five times the simulation time. In practice, we usually perform in-situ visualization less frequently (every 10^{th} time step), so the visualization time can be two orders of magnitude less than the overall simulation time.

Fig. 3 shows the results of exploring color and opacity mappings using the RAF for the HO_2 variable. Fig. 3 (a) shows the reconstruction result with the original transfer function. Fig. 3 (b) and (d) show the effect of opacity modulation and re-colorization. Fig. 3 (c) and (e) are the ground truth images obtained using direct volume rendering with the transfer functions in (b) and (d), respectively. We can see that the in-situ RAF results enable us to explore and highlight structures of interest as a post-processing step and without accessing the original simulation data.

4.1. Compression Methods for the RAF

An important consideration for reducing the size of data used in scientific studies is whether the compression is lossless or lossy. Loss of data may not always be acceptable; in fact, scientists usually prefer lossless compression methods that preserve the accuracy of their original results. However, due to storage space constraints and network bandwidth limitations, it is a common practice to store only a subset of the time steps generated by scientific simulations. Sometimes, hundreds of time steps are skipped. Thus, it is important to note that discarding time steps or storing only static images of time steps instead of the original data is, in essence, a lossy compression itself. Therefore, compared to completely discarding all or some intermediate time steps, compressing them in a lossy manner is a substantial improvement. We consider methods in both lossy and lossless compression as candidates for reducing the size of the RAF intermediate representation. In particular, since the data stored in the RAF exhibits image-like qualities, we employ such classic data reduction and image compression methods as quantization and lossless PNG compression. We also use statistical modeling of the attenuation distribution to reduce the size of the RAF representation. Finally, we present a method for constructing an alternative RAF representation, using kernel density estimation.

Conventional Data and Image Compression. Quantization is a commonly used data reduction method. It is the process of approximating a continuous range of values by a relatively small *finite* set of discrete values. In our study, we round single or double precision floating point values (32 and 64 bits, respectively) to 8 bits, which results in $4\times$ to $8\times$ compression.

In our experiments, lossless PNG image compression produced best compression ratios, without a significant loss of visual quality. To take advantage of spatial consistency, instead of storing consecutive RAF values per each pixel (say, 16 values for every pixel), we store the values for each "bin" in the RAF as a single-channel image. Fig. 4 shows the reconstruction error (avg. pixel difference) between the images reconstructed from the original RAF vs. images reconstructed from the compressed and then decompressed RAF. We use a heat map to show the variation in error for a combustion (max error: 7/255) and supernova (max error: 8/255) simulations. The error is greater in locations with most overlap of intricate semi-transparent features. The size of the resulting PNG image for the first example is 754KB, which results in approximately $54\times$ compression in comparison with storing the original RAF (here, 16 floating point values per pixel for an 800×800 image). The size of the resulting PNG image for the second example is 631KB, which results in approximately $64\times$ compression.

Parametric Model Fitting. The goal of parametric model fitting is to estimate the distribution parameters of a given attenuation distribution. For a given pixel, we can think of

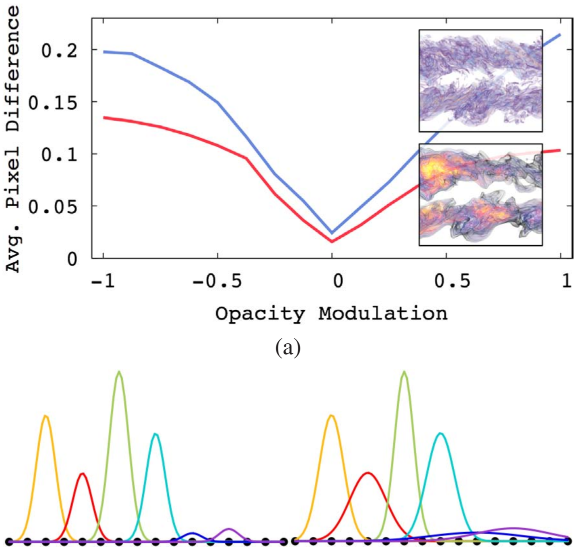

Figure 5: *(a) Reconstruction error (avg. pixel difference) between the images reconstructed from the original RAF vs. the mixture of Gaussians, as we reduce/increase opacity of all intervals for the OH and vorticity variables of a combustion simulation. Opacity modulation of 0 corresponds to no opacity change and is the base error. While the error stays very small, we see a smooth increase in error as opacity is reduced/increased further. (b) and (c) Plots of the mixture of Gaussians for a single pixel (100, 300) in the second image for fixed (b) and estimated (c) c.*

the RAF as a histogram, where the entire intensity value range is divided into N uniform bins. Histograms, however, are usually not smooth and assume the nearest neighbor assignment of data to bin values, which is very sensitive to the number and placement of the discrete bins. On the other hand, parametric models better represent an underlying distribution, using a finite number of parameters, i.e. there is a known upper bound on the required storage space. Since our data is image-based, our mathematical model of choice is the mixture of Gaussian functions of the form: $G(x,A,\mu,\sigma) = Ae^{-(x-\mu)^2/2\sigma^2}$, where A is the amplitude of the curve, and μ and σ are the mean and standard deviation of the curve, respectively. Thus, the RAF per pixel can now be represented as: $\mathsf{F}(x) = \sum_{i=0}^{K} G(x,A_i,\mu_i,\sigma_i)$.

Each sample along a ray contributes to the computation of the RAF for that ray. Since a ray may intersect only a few of the features in a data set, we usually do not encounter samples from each interval in the entire data range for a single pixel. Thus, the number of Gaussians in the mixture for each ray can be estimated from the number of features a ray intersects or the number of spikes/peaks in the RAF values for that pixel. We scan the values in the RAF and extract the amplitude A and mean μ for each peak. We can either use a fixed standard deviation or estimate it from the original RAF. We estimate the σ for each Gaussian in the mixture by clustering the RAF values. Each cluster includes the corresponding peak along with the neighboring RAF values. The values in a cluster cannot overlap any neighboring peaks. This strategy is easy to implement and works well for a small number of bins in an attenuation function (say, 16 values). For a RAF containing a larger number of bins, say 64 or more, we use the k-means algorithm to partition the RAF values into clusters. The Gaussian parameters are obtained as the mean and standard deviation of each cluster.

We achieve compression by storing the parameters for each Gaussian in the mixture, instead of the original RAF. Later, when the data is visualized, we reconstruct the RAF by estimating each value in the RAF from the stored parameters for the mixture of Gaussians. Fig. 5 shows the reconstruction error (avg. pixel difference) between the images reconstructed from the original RAF vs. the mixture of Gaussians, as we reduce/increase opacity of all intervals for the OH and vorticity variables of a combustion simulation. We use a transfer function with 7 distinct peaks. Opacity modulation of 0 corresponds to no opacity change and is the base error. While the error stays very small, we see a smooth increase in error as opacity is reduced/increased further. Fig. 5(b) and (c) shows the result of fitting a mixture of Gaussians for a single RAF, corresponding to the pixel with coordinates (100, 300) in the second image for fixed (b) and estimated (c) c. The accuracy of image reconstruction using the RAF with opacity modulation, as compared to the ground truth volume rendered images, is studied in [TCM10b].

To use one example, the size of our RAF representation, corresponding to the second image in Fig. 5(a) is 16 values per pixel for an 800×800 image. Storing two values per Gaussian (fixed σ) results in approximately $3.37\times$ compression and storing three values per pixel (variable σ) results in approximately $2.25\times$ compression, without a major loss of quality. The data can be further compressed using the data reduction and image compression methods described above.

4.2. Non-parametric Model Fitting

As opposed to parametric techniques, non-parametric model fitting methods can be used with arbitrary distributions. One of the most commonly used non-parametric techniques is *kernel density estimation (KDE)* [Sco92, Sil98, WJ95]. In kernel density estimation, the contribution of each data point is smoothed out over a local neighborhood of that data point, according to the following formula:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x-x_i) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x-x_i}{h}), \qquad (16)$$

where K is the kernel (we use a Gaussian kernel) and h is a smoothing parameter, or bandwidth.

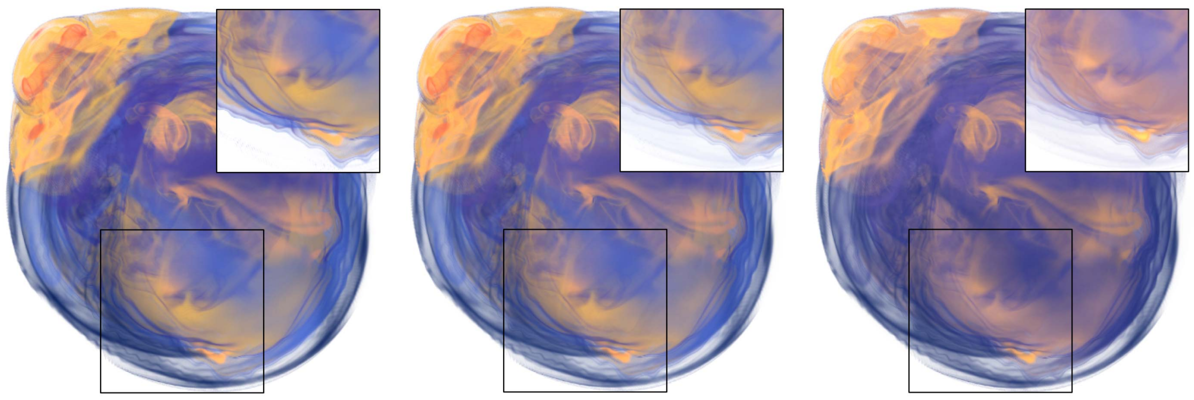

Figure 6: *Comparison of image reconstruction from the original RAF (a) vs. KDE estimates (b) and (c). (c) Image reconstruction from only 8 KDE estimates, which results in* $2\times$ *space savings without a major loss in quality.*

We use KDE to compute an alternative RAF representation (*N* estimates), which more closely represents the distribution of samples along a ray. We use constant bandwidth, for simplicity. Figure 6 compares the results of image reconstruction from the original RAF vs. KDE estimates. In Figure 6(c), we reconstruct an image using only 8 KDE estimates, which is $2\times$ less than the number of values used in (a) and (b). There is no major loss of quality. Using KDE becomes necessary for larger data sets and for transfer functions of higher frequency.

5. Limitations and Future Work

The IAF computation algorithm, presented in this paper, is general. It does not make any assumptions about the data and can be adopted to different scientific volumetric representations. Currently, we assume that volume data is stored in a regular grid, as it is common in scientific simulations. However, our technique also applies to less structured data, such as volumes discretized in unstructured meshes, or particle data, often used in smoothed particle hydrodynamics. The additional challenge in RAF computation for these types of data is handling ray traversals through the data efficiently.

Similar to the limitations described in [TCM10b], the degree of exploration available to users is bounded by the amount of information that can be encoded in the RAF. Due to attenuation, entire intervals in a volume may be poorly represented in the RAF, in which case it is not possible to retrieve the occluded features by means of attenuation modulation. A possible solution is to detect regions where attenuation reaches zero, similar to the opacity peeling techniques. This is the focus of our future work.

Attenuation functions are computed for a given view, so the intermediate representation is only valid for a given viewpoint. However, due to the compactness of the representation, it is now possible to store several RAF for a number of views. The visualization system can then offer a user the ability to change the orientation at interactive rates.

6. Conclusion

New approaches to large data visualization are needed to address the upcoming peta-scale data challenges. Although previous work has made strides towards computing intermediate representations of volume data, the existing approaches often do not scale to large data sets. In this paper, we present a parallel algorithm for computing compact intermediate representations of large volume datasets in an efficient manner. We show that we can achieve parallelism in two ways. Similarly to the way parallelism is exploited in contemporary graphics processing units, we process every pixel in an image in parallel. The second, less trivial part, is the exploitation of the associative property of attenuation along the viewing rays. We demonstrate that a block-based decomposition of the data helps us compute attenuation functions for large volume data sets efficiently. Moreover, we demonstrate a general mechanism for computing attenuation functions in parallel for less structured data. Our results demonstrate that the algorithm can produce compact representations of volume data that can be used for exploration, without much compromise on the accuracy of the rendering results. We also present a method for computing an alternative compact data representation, which more closely encodes the distribution of samples along the rays, and is more suitable for larger data sets and for transfer functions of higher frequency. Convinced by the results of our algorithm, we believe that the preview and exploratory techniques similar to ours will become widespread, because they can be easily adopted to different scientific data representations and deployed in today's in-situ and remote visualization settings.

7. Acknowledgments

This work has been sponsored in part by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777, and by the U.S. National Science Foundation through grants OCI-0325934, OCI-0749217, CNS-0551727, CCF-0811422, CCF-0808896, OCI-0749227, OCI-0950008, and CCF-0938114. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. Department of Energy under contract DE-AC04-94-AL85000. We also acknowledge computing resources of the National Center for Computational Sciences (NCCS) at ORNL, which has been supported by the Office of Science of the U.S. Department of Energy under contract DE-AC05-00OR22725. Data sets courtesy of Jacqueline H. Chen of Sandia National Laboratories and the Terascale Supernova Initiative.

References

[AP98] Ahrens J. P., Painter J. S.: Efficient sort-last rendering using compression-based image compositing. In *Proc. of Eurographics Workshop on Parallel Graphics and Visualization* (1998), pp. 145–151. 3

[CCdS*09] Chen J. H., Choudhary A., de Supinski B., DeVries M., Hawkes E. R., Klasky S., Liao W. K., Ma K. L., Mellor-Crummey J., Podhorszki N., Sankaran R., Shende S., Yoo C. S.: Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science & Discovery 2* (2009). 5

[DH01] Ding C., He Y.: A Ghost Cell Expansion Method for Reducing Communications in Solving PDE Problems. In *Proc. of Supercomputing* (2001). 5

[EP07] Eilemann S., Pajarola R.: Direct send compositing for parallel sort-last rendering. In *Proc. of Eurographics Symposium on Parallel Graphics and Visualization* (2007), pp. 29–36. 3

[HHKP96] He T., Hong L., Kaufman A., Pfister H.: Generation of transfer functions with stochastic search techniques. In *Proc. of IEEE Visualization* (1996), pp. 227–ff. 3

[LP03] LaMar E., Pascucci V.: A multi-layered image cache for scientific visualization. In *Proc. of IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), pp. 61–68. 3

[MAB*97] Marks J., Andalman B., Beardsley P. A., Freeman W., Gibson S., Hodgins J., Kang T., Mirtich B., Pfister H., Ruml W., Ryall K., Seims J., Shieber S.: Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proc. of SIGGRAPH* (1997), pp. 389–400. 3

[Max95] Max N.: Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (1995), 99–108. 4

[MCEF94] Molnar S., Cox M., Ellsworth D., Fuchs H.: A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications 14*, 4 (1994), 23–32. 3

[MCP91] Ma K.-L., Cohen M. F., Painter J. S.: Volume seeds: A volume exploration technique. *Visualization and Computer Animation 2* (1991), 135–140. 3

[MMG07] Malik M. M., Möller T., Gröller M. E.: Feature peeling. In *Proc. of Graphics Interface* (2007), pp. 273–280. 3

[MPHK93] Ma K.-L., Painter J. S., Hansen C. D., Krogh M. F.: A Data Distributed, Parallel Algorithm for Ray-Traced Volume Rendering. In *Proc. of Parallel Rendering Symposium* (1993), pp. 15–22. 3

[MWYT07] Ma K.-L., Wang C., Yu H., Tikhonova A.: In situ processing and visualization for ultrascale simulations. *Journal of Physics: Conference Series (also in Proc. of DOE SciDAC Conference) 78*, 012043 (2007). 3

[Neu94] Neumann U.: Communication costs for parallel volume-rendering algorithms. *IEEE Computer Graphics and Applications 14*, 4 (1994), 49–58. 3

[PGR*09] Peterka T., Goodell D., Ross R. B., Shen H.-W., Thakur R.: A configurable algorithm for parallel image-compositing applications. In *Proc. of Supercomputing* (2009). 3

[RBG07] Rautek P., Bruckner S., Groller M. E.: Semantic layers for illustrative volume rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1336–1343. 3

[RPSH08] Ropinski T., Prassni J.-S., Steinicke F., Hinrichs K. H.: Stroke-based transfer function design. In *Proc. of IEEE/Eurographics International Symposium on Volume and Point-Based Graphics* (2008), pp. 41–48. 3

[RSK06] Rezk-Salama C., Kolb A.: Opacity peeling for direct volume rendering. *Proc. of Computer Graphics Forum 25*, 3 (2006), 597–606. 3

[SCM03] Srivastava V., Chebrolu U., Mueller K.: Interactive transfer function modification for volume rendering using compressed sample runs. In *Proc. of Computer Graphics International Conference* (2003), pp. 8–13. 3

[Sco92] Scott D. W.: *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, Inc., 1992. 8

[Sil98] Silverman B. W.: *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1998. 8

[SLSM06] Shareef N., Lee T.-Y., Shen H.-W., Mueller K.: An image-based modeling approach to GPU-based unstructured grid volume rendering. In *Proc. of Volume Graphics* (2006), pp. 31–38. 3

[TCM10a] Tikhonova A., Correa C. D., Ma K.-L.: Explorable images for visualizing volume data. In *Proc. of IEEE Pacific Visualization Symposium* (2010). 3

[TCM10b] Tikhonova A., Correa C. D., Ma K.-L.: An exploratory technique for coherent visualization of time-varying volume data. *Computer Graphics Forum 29*, 3 (2010), 783–792. 2, 3, 4, 8, 9

[WJ95] Wand M., Jones M.: *Kernel Smoothing*. Chapman & Hall, 1995. 8

[WQ07] Wu Y., Qu H.: Interactive transfer function design based on editing direct volume rendered images. *IEEE Transactions on Visualization and Computer Graphics 13*, 5 (2007), 1027–1040. 3

[YWG*10] Yu H., Wang C., Grout R. W., Chen J. H., Ma K.-L.: In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications 30*, 3 (2010), 45–57. 3

[YWM08] Yu H., Wang C., Ma K.-L.: Massively Parallel Volume Rendering Using 2-3 Swap Image Compositing. In *Proc. of Supercomputing* (2008). 3, 6