Time-Varying Data Visualization Using Functional Representations

Yun Jang, *Member, IEEE,* David S. Ebert, *Fellow, IEEE,* and Kelly Gaither

Abstract—In many scientific simulations, the temporal variation and analysis of features are important. Visualization and visual analysis of time series data is still a significant challenge because of the large volume of data. Irregular and scattered time series datasets are even more problematic to visualize interactively. Previous work proposed functional representation using basis functions as one solution for interactively visualizing scattered data by harnessing the power of modern PC graphics boards. In this paper, we use the functional representation approach for time-varying datasets and develop an efficient encoding technique utilizing temporal similarity between time steps. Our system utilizes a graduated approach of three methods with increasing time complexity based on the lack of similarity of the evolving datasets. Using this system, we are able to enhance the encoding performance for the time-varying datasets, reduce the data storage by saving only changed or additional basis functions over time, and interactively visualize the time-varying encoding results. Moreover, we present efficient rendering of the functional representations using binary space partitioning tree textures to increase the rendering performance.

Index Terms—Basis Functions, Functional Representation, Time-Varying Data, Volume Rendering.

1 Introduction

Currently, many scientists are able to simulate large complex time-varying phenomena. These simulations have grown increasingly complex in both the space and time dimensions. However, visualization and visual analysis of this time-varying data is still a significant challenge because of the large amount of volume data to process. Moreover, if the time-varying data is irregular, unstructured, or scattered, the visualization and analysis is even more problematic. In past years, researchers have proposed several approaches for functional representations in order to represent volumetric data [13], [14], [19], [44], [45]. Using these functional representations, many types of volumetric datasets, including scattered point data, are functionally represented and data values at any point in the volume can be calculated from the functional representations.

The functional representations are formed as the weighted sum of all basis functions and the general mathematical form is presented in the following equation.

$$f(\vec{x}) = \sum_{j=1}^{M} \lambda_j \phi_j(\vec{x}) \quad (1)$$

where ϕ is the basis function, λ is the weight, and M is the number of basis functions. Note that parameters are λ and ones from the basis function ϕ. This functional representation can be evaluated and visualized directly using modern PC graphics boards by storing all function parameters in textures. However, all of these functional representations are performed with single-time-step datasets and there has been no sufficient method proposed to represent time-varying datasets functionally, other than computing all time steps individually. The individual computation of all time steps is very time consuming and the size of the functionally represented datasets for all time steps can be quite large.

In this work, we extend the functional representation approach from Jang et al. [13] and develop an efficient functional representation technique for time-varying data. Our approach generates a unified representation with functions for any type of volumetric data including scattered point data. Since most time-varying datasets have temporal coherence between consecutive time steps, we use the similarity between time steps to determine the best functional representation methods. Figure 1 shows four consecutive steps of the convection simulation dataset in each row. The top row includes the 51st, 52nd, 53rd, and 54th steps and the bottom row shows the 75th, 76th, 77th, and 78th steps. As shown in this figure, consecutive time steps show very similar volumetric patterns. Note that we term *functional representation* as *encoding* in this paper. For time series encoding, as an extension to [13], we can encode each time step independently. However, since the functional representation requires expensive computation to generate a better compression ratio and rendering performance, the process to encode all time steps independently would be very time consuming. Instead of encoding each step independently, we develop an approach similar to the MPEG encoding algorithm. In our system, we *currently* set two different frame types, which are the intra-frame (*I-frame*) and the forward predictive frame (*P-frame*). In order to decide the frame type, we utilize temporal similarity

- *Y. Jang is with the Department of Computer Science at ETH Zürich, Switzerland. E-mail: jangy@inf.ethz.ch*
- *D. S. Ebert is with the School of Electrical and Computer Engineering at Purdue University, IN, 47906. E-mail: ebertd@purdue.edu*
- *K. Gaither is with the Data & Information Analysis Visualization and Data Analysis at Texas Advanced Computing Center in The University of Texas at Austin , TX, 78758. E-mail: kelly@tacc.utexas.edu*

Digital Object Indentifier 10.1109/TVCG.2011.54

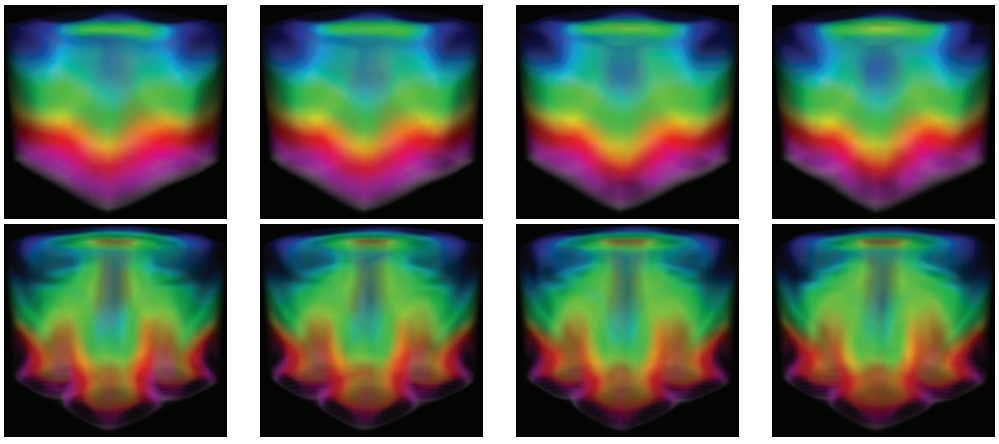


Fig. 1. These images show 4 time steps of natural convection data. From left, 51st, 52nd, 53rd, 54th on the top row and 75th, 76th, 77th, 78th on the bottom row. As shown here, consecutive steps show similar patterns in the volume.

between two consecutive steps and errors from the previous encoding result to determine an encoding method over time. If two consecutive steps are very similar and the errors from the previous encoding result is small, we generate a *P-frame*; otherwise, we generate an *I-frame* as a new encoding regardless of the previous encoding result. In this way, we need to store only a small number of additional basis functions for the *P-frame* since it only has locally influencing basis functions. This *P-frame* encoding uses local errors, which accelerates the evaluation and rendering performance, as well. Therefore, the total encoding time and the total encoding data size is decreased compared to the traditional independent encoding process.

To render the encoding results, we store the parameters of all basis functions for all time steps in 2D textures and evaluate the data values in a fragment program. For the *I-frame*, we evaluate all basis functions, whereas, we evaluate only additional basis functions for the *P-frame* using a *render to 3D texture* technique. In order to accelerate the evaluation and rendering performance, we use the hierarchical spatial structure proposed by Jang et al. [14]. In this work, however, we store the hierarchical structure in an additional 3D texture to search for the texture coordinates of the 2D textures instead of tracking and finding all intersections between spatial cells and slices for the volume rendering on CPU.

Our major contributions in this work are:

- The efficient and fast functional representation for time-varying volumetric data using similarity between time steps
- The generation of a much smaller data size for time series data from large amount of input time-varying data
- The interactive volume rendering of the time-varying data using the 3D textures of the hierarchical structure

In addition to these contributions, we present comparisons of two volume rendering techniques, slice based and ray casting volume rendering, and comparisons of the hierarchical spatial structures using octree and binary space partitioning tree.

We first review previous work in Section 2 and describe our encoding process in Section 3. We then introduce details of our interactive evaluation and visualization of the encoding results and present results produced by our system in Section 4 and 5. Finally conclusions and future directions are described in Section 6.

2 Related work

The large volume of time-varying data makes visualization a challenging problem. Many techniques for volume rendering of time-varying data have been proposed and these techniques enable the visualization of large time-varying datasets. One approach is to use data coherency between consecutive time steps to speed up volume rendering [1], [2], [34], [35], [50]. Another approach is to encode and compress the time-varying data appropriately for volume rendering [18], [21], [22], [39], [46]. Shen and Johnson [35] propose an algorithm that exploits data coherency between time steps and extracts the differential information for biomedical and computational fluid dynamics datasets. Shen et al. [34] show the time-space partitioning (TSP) tree. This TSP structure improves the rendering speed and reduces the amount of volumetric data I/O. Younesy et al. [50] propose a differential time-histogram table for an efficient data update in order to minimize the amount of data that needs to be loaded from disk. For the temporal compression approach, Westermann [46] proposes a memory minimizing algorithm based on multi-resolution representations of both the spatial distribution and the time evolutions. Ma and Shen [22] present quantization and octree encoding of time-varying datasets and reduce the rendering speed over time. Lum et al. [21] show temporal encoding using discrete cosine transform (DCT) and accelerate the rendering speed using the graphics hardware. Sohn et al. [39] present a compression scheme for encoding time-varying isosurface and volume features. They encode only the significant blocks using a block-based wavelet transform. Recently Ko et al. [18] proposed large time-varying data visualization using video-based compression, such as MPEG. However, all of these techniques are for regular grid datasets and prove difficult to extend to scattered time-varying datasets.

Bernardon et al. [1], [2] present an interactive volume rendering of unstructured grids for time-varying scalar fields and use multi-resolution techniques, such as time-varying level-of-detail with coefficient of variation (COV). Leeuw and Liere [7], [8] propose an algorithm to estimate and analyze motion in time-varying volumetric data. They use local block matching to extract a velocity field for the motion to track motion features [7] and use mass conservation to estimate the motion globally [8]. Silver et al. [5], [15], [37], [38] present a tracking and visualization algorithm of important features over time. Tzeng and Ma [43] also show the extraction and tracking of time-varying flow data by using machine learning algorithms to directly extract and track features in a high-dimensional space. Muelder and Ma [25] use a prediction-correction method and coherently extract actual feature of interest.

Previously, researchers used radial basis functions to interpolate and approximate scattered data [4], [9], [24], [28], [29], [33], [42] and they study functional interpolation and approximation for surface and volume datasets. Jang et al. [13], [14] and Weiler et al. [45] propose a functional representation approach for interactive volume rendering. Their approaches are designed for any scattered datasets and directly volume render the basis functions without resampling. Moreover, using ellipsoidal basis functions, they improve the functional representation statistically and visually [13]. Recently, Ledergerber et al. [19] applied a moving least square to interpolate the volumetric data and Vuçini et al. [44] reconstruct non-uniform volumetric data by B-splines.

For the use of radial basis functions in the time-varying data, most previous work focuses on solving partial differential equations [16], [17], [32], [36]. These approaches are mostly for generating meshless data instead of data approximation. In flow modeling, Pighin et al. [31] use radial basis functions by combining the Eulerian and Lagrangian representations and advected radial basis functions. They model local flow properties of the fluid and used the model for editing the three-dimensional flow. Park et al. [30] propose an approach that represented time-varying 3D scattered data as static 4D scattered data and in their work, they directly rendered time-orthogonal three-dimensional hyperplanes. Zhou et al. [51] present real-time smoke rendering with radial basis functions and they utilize *render to 3D texture* to evaluate the radial basis functions.

3 Functional Representation System of Time-Varying Data

Our system is designed to generate the functional representations for time-varying data. Instead of encoding each time step individually, our system produces correlated results between consecutive time steps using the previous encoding results. Figure 2 shows a diagram of our time-varying data encoding system.

The input and output of the encoding system for the time-varying data are defined as follows.

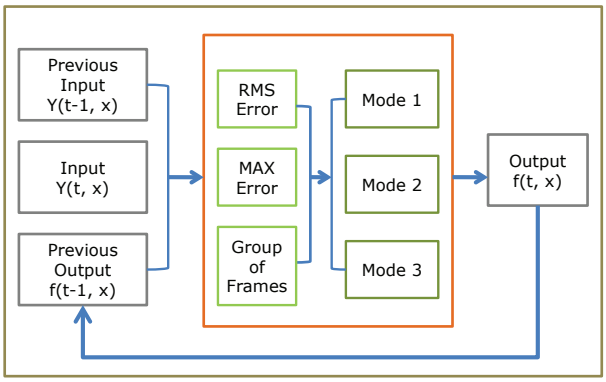


Fig. 2. Our time-varying data encoding system. There are three inputs and one of the inputs is taken from the previous time output. Each mode is automatically chosen based on the errors and group of frame (GOF) information.

$$\text{Input} = \{Y(0,\vec{x}), Y(1,\vec{x}), \ldots, Y(t-1,\vec{x}), Y(t,\vec{x})\}$$
$$\text{Output} = \{f(0,\vec{x}), f(1,\vec{x}), \ldots, f(t-1,\vec{x}), f(t,\vec{x})\}$$

where $Y(t,\vec{x})$ is input *scattered* data at time t and $f(t,\vec{x})$ is the functional representation of $Y(t,\vec{x})$. The time-varying functional representation is defined as the following.

$$f(t,\vec{x}) = \sum_{T=T_I(t)}^{t-1} \sum_{j=1}^{M(T)} \lambda_j(T)\phi_j(T,\vec{x}) + \sum_{j=1}^{M(t)} \lambda_j(t)\phi_j(t,\vec{x}) \quad (2)$$

where $T_I(t)$ is the *I-frame* time index and all other parameters are similarly defined as in Equation 1. The functional representation at time t is a sum of the previous encoding results from the nearest *I-frame* ($T_I(t)$) and the current encoding result of residual errors from the previous encoding results. For the encoding, we have three different modes according to our measurements including errors from the previous time step encoding result $f(t-1,\vec{x})$ and group of frame (GOF) information which is similar to group of pictures used in MPEG. In this work the GOF is defined as the number of frames between *I-frame*s.

3.1 Measurements for temporal encoding

One measurement is the residual of the current time step from the previous time step results. Let $Y(t,\vec{x})$ be the data at the current time step and $f(t-1,\vec{x})$ be the encoding result from the previous time step. The residual error from the previous encoding result for the current time step is defined as the following.

$$r(t,\vec{x}) = Y(t,\vec{x}) - f(t-1,\vec{x}) \quad (3)$$

We then measure the maximum and RMS errors from the previous result. The maximum error from the previous result is defined as the following.

$$e_{max}^{prev}(t) = \max_{i=1,\ldots,N} \left| r(t,\vec{x}_i) \right| \quad (4)$$

where N is the number of input data points. The RMS error from the previous result is defined as the following.

$$e_{RMS}^{prev}(t) = \sqrt{\frac{\sum_{i=1}^{N} r^2(t,\vec{x}_i)}{N}} \tag{5}$$

For the termination of the encoding system, we measure the maximum error from the current result. The maximum error from the current result is defined as the following.

$$e_{max}^{curr}(t) = \max_{i=1,\ldots,N} \left| Y(t,\vec{x}_i) - f(t,\vec{x}_i) \right| \tag{6}$$

The RMS error from the current result is computed to provide the quality of the encoding and the RMS error from the current result is shown as the following.

$$e_{RMS}^{curr}(t) = \sqrt{\frac{\sum_{i=1}^{N} \left(Y(t,\vec{x}_i) - f(t,\vec{x}_i)\right)^2}{N}} \tag{7}$$

Note that we normalize the errors by the maximum data values, $Y_{max}(t) = \max_{i=1,\ldots,N} Y(t,\vec{x}_i)$.

We utilize the values defined in Equation 4 and 5 to design a system encoding volumetric time-varying data and the value from Equation 6 to terminate the functional representation.

3.2 Encoding modes

We build our time-varying encoding system utilizing the maximum error (Equation 4) and the RMS error (Equation 5) based on the functional representation system presented in [13], [14], [45]. Details of the functional representation system are summarized in Section 3.3. In this work, we consider both the encoding time and the encoding results for the interactive rendering. Our focus is on developing faster time series encodings, compared to independent time step encoding, and storage saving for the encoding results. In addition, faster volume rendering of the time-varying encoding results is also considered.

As mentioned above, in our system there are three different encoding modes. We choose each mode based on the maximum error and the RMS error. Let $f_r(t,\vec{x})$ be the additional or different set of basis functions from the previous encoding results with $f(t-1,\vec{x})$ being the encoding for the current time step. Note that $f_r(t,\vec{x})$ is the encoding result only for $r(t,\vec{x})$ from Equation 3. The functional representation in the time step t for *I-frame* is $f(t,\vec{x})$, whereas, the functional representation for *P-frame* is $f(t-1,\vec{x}) + f_r(t,\vec{x})$.

A brief summary of the encoding modes is as follows:

- Mode 1 (*P-frame*): Encode only the residual, $r(t,\vec{x})$, from Equation 3, and add the results into the previous results
- Mode 2 (*I-frame*): Encode current data $Y(t,\vec{x})$ using previous results, $f(t-1,\vec{x})$, and stop optimizing $f(t-1,\vec{x})$ from the first sets of parameters when max error, $e_{max}^{curr}(t)$, is less than our error threshold (e_{th})
- Mode 3 (*I-frame*): Encode current data $Y(t,\vec{x})$ independently

Mode 1 is used when the RMS error $e_{RMS}^{prev}(t)$ is less than $\alpha \times e_{th}$ and the maximum error $e_{max}^{prev}(t)$ is greater than e_{th}. The α is set as 0.6 and e_{th} is set as 5% of the maximum input data value. In this mode, we encode the residual $r(t,\vec{x})$ until $e_{max}^{curr}(t)$ is less than e_{th}. This Mode 1 generates *P-frame* since $f(t,\vec{x})$ is dependent on $f(t-1,\vec{x})$.

Mode 2 is used when the maximum error $e_{max}^{prev}(t)$ is less than e_{th}. In this case, $f(t-1,\vec{x})$ fits all data values sufficiently, i.e. too many basis functions for this time step. Therefore, we adjust the parameters of $f(t-1,\vec{x})$ one set by one set until $e_{max}^{curr}(t)$ is less than e_{th}. This mode produces *I-frame* because some of parameters in $f(t-1,\vec{x})$ are changed; therefore, we are not able to obtain $f(t,\vec{x})$ from $f(t-1,\vec{x})$.

Mode 3 is used when the RMS error $e_{RMS}^{prev}(t)$ is greater than $\alpha \times e_{th}$ and the maximum error $e_{max}^{prev}(t)$ is greater than e_{th}. Since there is no coherence between two steps, we encode the data independent of $f(t-1,\vec{x})$. This mode generates *I-frame*. In addition to this condition, Mode 3 is selected based on the group of frames (GOF) in order to reset the temporal encoding and we select Mode 3 at every GOFth frame.

The number of *I-frame*s is influenced by the α and GOF. The α indicates how much error we tolerate for *P-frame*s and smaller α generates more *I-frame*s. The GOF is used to reset the encoding mode and smaller GOF generates more *I-frame*s.

3.3 Encoding Detail

Once the encoding mode is determined, we encode either $r(t,\vec{x})$ for Mode 1 or $Y(t,\vec{x})$ for Mode 2 and 3. The basis function ϕ_i used in Equation 2 is set as an axis aligned ellipsoid $\lambda \cdot \ exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2} - \frac{(z-\mu_z)^2}{2\sigma_z^2}\right)$ in this work but we can easily replace the basis function with other basis functions, such as Gaussian or arbitrary directional ellipsoids. We insert basis functions one by one at the largest error points and optimize the basis function parameters such as λ, $\mu_{x,y,z}$, and $\sigma_{x,y,z}$, using a nonlinear optimization method, Levenberg Marquardt [23] with H1-norm [10] cost function defined as the follow.

$$\begin{aligned}\psi(t) = \frac{1}{2}\sum_{i=1}^{N}\Big\{ &(f(t,\vec{x}_i) - Y(t,\vec{x}_i))^2 + \left(\frac{\partial f(t,\vec{x}_i)}{\partial x} - \frac{\partial Y(t,\vec{x}_i)}{\partial x}\right)^2 + \\ &\left(\frac{\partial f(t,\vec{x}_i)}{\partial y} - \frac{\partial Y(t,\vec{x}_i)}{\partial y}\right)^2 + \left(\frac{\partial f(t,\vec{x}_i)}{\partial z} - \frac{\partial Y(t,\vec{x}_i)}{\partial z}\right)^2 \Big\}\end{aligned} \tag{8}$$

The termination condition (e_{th}) is set as 5% of the maximum data values. The e_{th} controls the number of basis functions generated from our encoding system and smaller e_{th} indicates more basis functions in the results. Since we use the H1-norm cost functions for the nonlinear optimization, the error metric for the basis functions is the H1-norm although we terminate our encoding system when e_{max}^{curr} is less than e_{th}. The $\mu_{x,y,z}$ can be other than the data point locations and the $\sigma_{x,y,z}$ (kernel sizes) vary after the nonlinear optimizations. Therefore, the number of basis functions is not fixed and tends to be much smaller

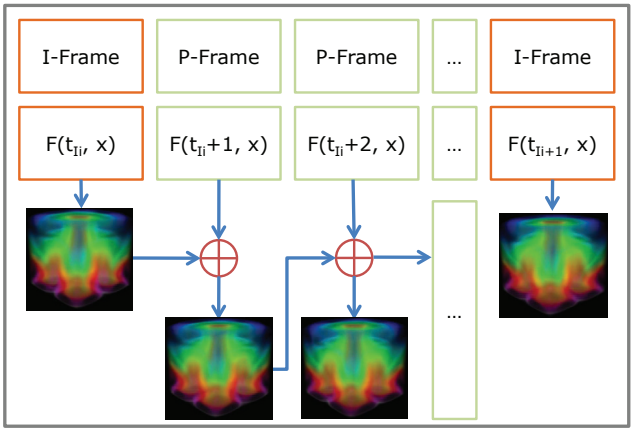


Fig. 3. Our rendering system for the functional representations of time-varying data. In this diagram, two *I-frames* and *P-frames* between them are shown. The *P-frames* are rendered using the previous *P-frames* and *I-frame*.

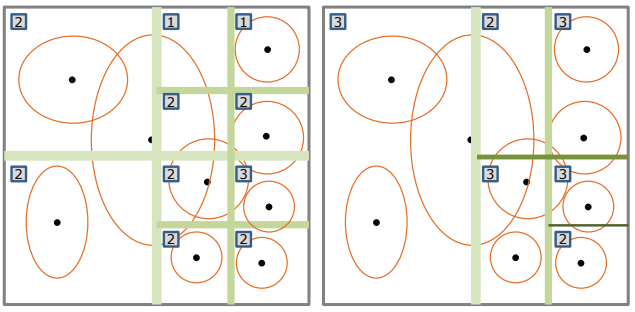
than the number of input data points. Since we develop our temporal encoding system based on the work by Jang et al. [13], we refer the reader to that work for more details.

4 Evaluation and Rendering of Functional Representation

In the previous section, we introduced our encoding system for the time-varying data. The encoding results are directly used for the rendering. In this section, we present the evaluation and rendering of the functional representations using the GPU. We store all parameters for all time steps in textures and evaluate the data values using a spatial hierarchical decomposition with a binary space partitioning (BSP) tree. In this work, we evaluate and render the functions with both slice based and ray casting volume rendering techniques. Our rendering system for the time dependent functional representations is illustrated in Figure 3. We have two different rendering modes according to the frame type, either *I-frame* or *P-frame*.

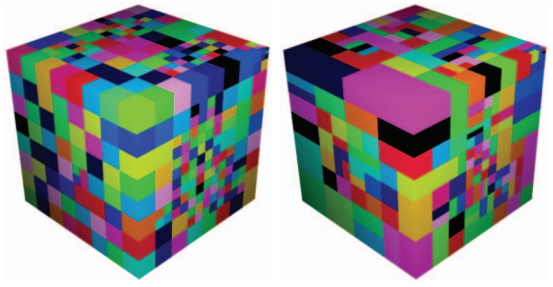
4.1 Spatial hierarchy using BSP tree

While evaluating the functional representations, we need to use all basis functions in a volume to obtain the data values. If we ignore very small data values from the basis functions, such as less than 0.001, then we can compute the influence ranges of the basis functions. However, some basis functions influence only small regions and the previous work by Jang et al. [14] suggests using an octree for spatial decomposition to accelerate the rendering performance. Each subdivision generates 8 subcells, although some of the regions do not need to be split, and more texture memories are consumed to store the decomposed basis functions. In this work, we propose using an axis aligned binary space partitioning (BSP) tree instead of an octree to decompose the basis functions. The volume is subdivided along an axis, and the axis is chosen such that the smallest number of basis functions are generated after splitting. Note that the axis must pass through the center of a cell. Then we



(a) Using quad tree (b) Using BSP tree

Fig. 4. Example of our spatial decomposition. Black dots are the centers of basis functions and red curves are the influence ranges of basis functions. Green lines are splitting axes and the order of decomposition are presented as the thickness (thick to thin) and color (light green to dark green) of the lines. $M_{max-percell}$ is set as 3 in this example. (a) illustrates the spatial decomposition using a quad tree in 2D case proposed by [14]. Total number of basis functions sent to GPU is 19 in this case. (b) is the spatial decomposition with a binary space partitioning (BSP) tree. The number of basis functions sent to GPU is 16.



(a) Octree (b) BSP tree

Fig. 5. Comparison of octree and BSP tree hierarchy showing basis function locality in the 1^{st} timestep of the vortex data. (a) shows the spatial structure using the octree (2,192 cells and 163,715 basis functions), whereas, (b) is the spatial hierarchy with the BSP tree (925 cells and 83,108 basis functions).

subdivide the volume until the number of basis functions in a cell is smaller than a certain number of basis functions ($M_{max-percell}$). Figure 4 presents a comparison of a quad tree and a 2D BSP tree after the spatial decompositions. Note that the quad tree is used for this 2D case. Figure 4 (a) is the spatial hierarchy using a quad tree and we obtain 10 subcells with total 19 basis functions. On the other hand, (b) shows our BSP tree based decomposition. We only generate 6 subcells with total 16 basis functions. Note that the number of basis functions for each cell is placed in a box. One disadvantage of this BSP tree is that we need to evaluate more basis functions for some volume locations compared to the octree based decomposition. From Figure 4, in the upper right regions, if we use the BSP tree, we have to evaluate 3 basis functions compared to 1 basis function. However, as long as the number of basis functions does not exceed the number $M_{max-percell}$, the rendering performance is not changed because other subcells have more than 1 basis function. Figure 5 shows a comparison of octree and BSP tree spatial hierarchies. In this example, we use the 1^{st} time step of vortex data and 3,807 basis functions are generated from our encoding

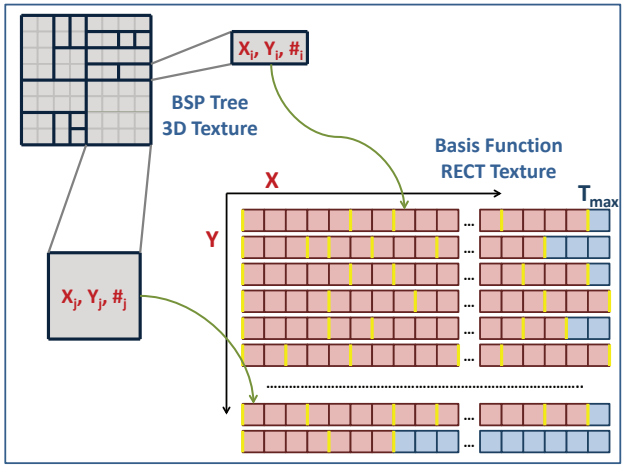
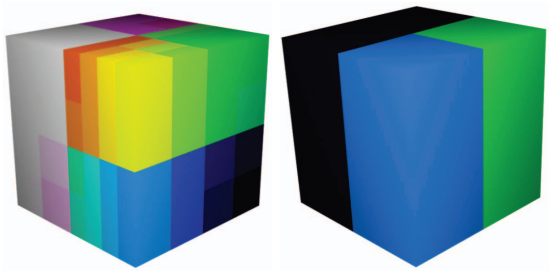


Fig. 6. Texture layouts of the hierarchical BSP tree texture and RECT textures for the functional representations. The hierarchical texture is a 3D texture (shown here as 2D) and it contains texture coordinates (X and Y) of RECT textures and the number of basis functions (#). The RECT texture is composed of lists of basis functions and a list of basis functions for one hierarchical cell is placed between two yellow bars. Note that red indicates that texture cells are filled and blue indicates that texture cells are empty in the RECT texture. T_{max} is the maximum RECT texture size. In this figure, as an example, we show two hierarchical cells and their corresponding RECT texture cell positions with green arrows.

system. (a) is the spatial hierarchy using the octree and the octree decomposition produces 2,192 spatial cells and 163,715 basis functions. Note that basis functions can be duplicated. (b) shows the spatial hierarchy using the BSP tree, producing 925 spatial cells and 83,108 basis functions. As presented in the figure, the spatial decomposition using the BSP tree generates much fewer number of spatial cells and duplicated basis functions. Therefore, the data size stored on the GPU is reduced compared to using the octree decomposition. In the worst case, the maximum data size using the BSP tree is the same as the data size using the octree, but generally the data size using the BSP tree is much smaller than using the octree. Note that there is no performance difference between the octree and BSP tree decomposition since the maximum number of basis functions per spatial cell is the same.

4.2 Textures and evaluation

For storing the functional representations, we use GPU textures and simply store all parameters of the basis functions in several RECT textures in the similar way as the work in [13], [14], [45]. The RECT textures store the basis functions and the order of RECT texture packing is based on the lists of basis functions from the hierarchical BSP tree structure. A list of basis functions for one hierarchical cell is placed together and then another list of basis functions for the next hierarchical cell is inserted next to the previous list. Since there is a maximum size (T_{max}) of RECT texture according to graphics hardware, the X texture coordinate can not exceed this maximum size. Therefore, if the accumulated number of basis functions is larger than T_{max} along X, we simply move the new list of the basis functions to the



(a) Independent encoding (b) Time dependent encoding

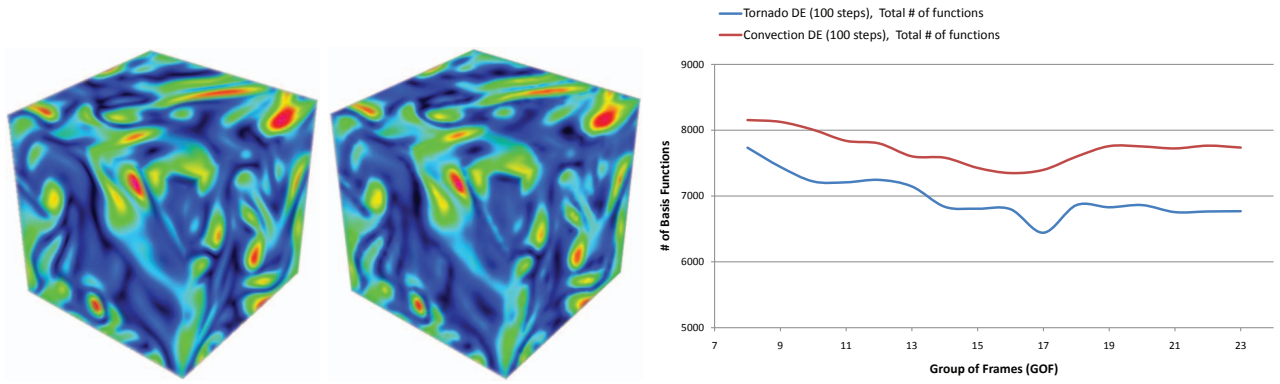
Fig. 7. Comparison of the BSP tree 3D textures for the spatial hierarchy showing basis function locality in the 94^{th} timestep of the convection data. (a) shows the spatial structure of the independent encoding, whereas, (b) is the *P-frame* spatial hierarchy.

next Y position. This texture packing strategy is illustrated in Figure 6. In the RECT texture, blue texture cells indicate that they are empty because the accumulated number of basis functions to be placed is larger than T_{max}; therefore, we leave the texture cells as empty. The number of RECT textures varies according to the number of parameters. Two RECT textures are needed for the axis aligned ellipsoidal basis functions since there are 7 parameters.

However, when the functions were evaluated in the previous work, it required a search for all of the intersections of the subcells with the slices or rays, and, if there are many subcells, it decreases the evaluation performances. Instead, inspired by Lefebvre et al. [20], we propose to use one 3D texture for the BSP tree in order to find the texture coordinates of the RECT parameter textures efficiently. We set the maximum level (L_{max}) of the hierarchy and the 3D texture size is determined by $2^{L_{max}}$. Figure 5 and 7 illustrates examples of the 3D textures. The general evaluation of the functional representation is straightforward as shown in Figure 6. We first fetch the 3D BSP tree texture by using the position of either slice or ray as the texture coordinate. The value fetched from the 3D texture is used to locate the texture coordinate of the RECT parameter textures. Using the 3D texture allows us to have a large number of subcells, and the performance is not affected by the number of subcells as long as we have enough GPU memory to store the all basis functions.

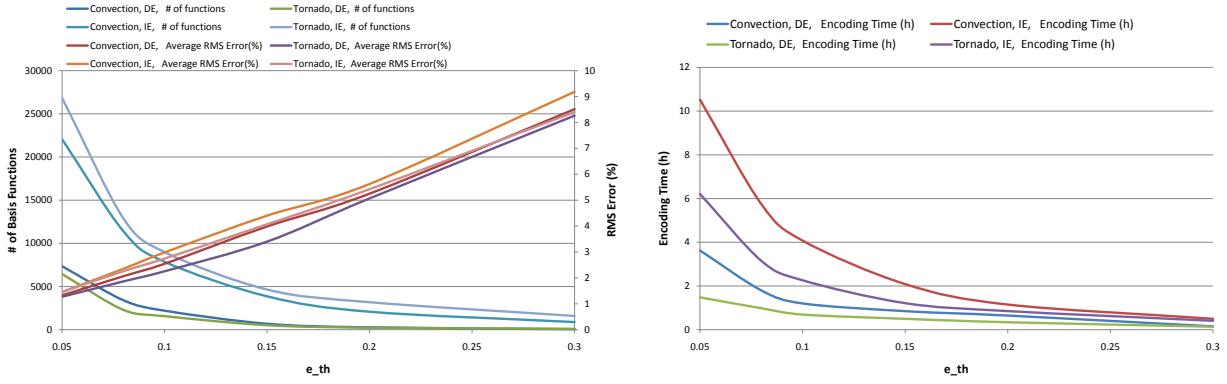
4.3 *I-frame* and *P-frame* rendering

From the proposed time-varying data encoding, our encoding system generates two different frame types, *I-frame* and *P-frame*. The *I-frame* is independent of the previous time steps, which is a new result, whereas, the *P-frame* can be evaluated only with the previous time steps including the previous *I-frame* and these are illustrated in Figure 3. The rendering of the *I-frame* is straightforward as described in Section 4.2. However, the *P-frame* needs more attention since all the previous time steps should be evaluated together backward by the last *I-frame*. Since we store all time steps of the functional representations in the textures, we can easily find the parameters of the previous time steps. However, this approach might degrade the rendering performances because of possibly large number of basis



(a) Original vortex data (b) Our functional representation (c) Comparison of the number of basis functions for the Isabel data

Fig. 8. Quality of our functional representation using the vortex data and number of basis functions obtained from time dependent encoding results according to the group of frame (GOF). (a) is the 47^{th} step of the original vortex data and (b) is our functional representation of the vortex data. In the graph (c), 15 - 17 GOFs provide smaller basis functions compared to other GOFs. DE indicates the time dependent encoding.



(a) Comparison of the number of basis functions and RMS errors (b) Comparison of the encoding time

Fig. 9. Number of basis functions and RMS errors (a), and encoding time (b) according to e_{th} using the convection and tornado datasets.

function to be evaluated. Therefore, we use *render to 3D texture* as an alternative approach to store the results for the previous steps and the *render to 3D texture* discards the evaluation of the previous *P-frames* and the previous *I-frame*. We only need to evaluate the current *P-frame* and add the 3D texture values to the current evaluations for the final values. This increases the rendering performances because each *P-frame* has a small number of additional basis functions. One disadvantage of the *render to 3D texture* is that data values are pre-sampled in a grid structure using the frame buffer object and the grid size depends on the original input data.

5 Results and Discussion

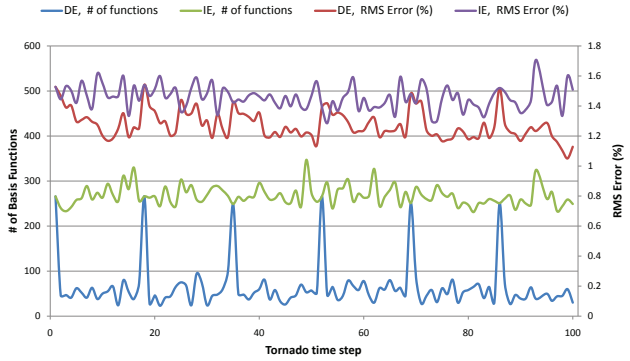
We have tested our encoding using 8 Intel Nehalem Core 2.5GHz processors and rendering system using Core 2 Quad CPU 2.4GHz processor with NVIDIA GeForce GTX 295 graphics hardware respectively. As volume rendering techniques, we use both slice based rendering [48] and ray casting [11] in this work to compare the rendering performances.

We tested our techniques with five different datasets including the tornado, natural convection, turbulent vortex, Isabel data, and galaxy data. The tornado dataset was generated by the original program developed by R. Crawfis, from the Ohio State University and we generate 100 time steps of 32,768 points in a volume. The natural convection dataset simulates a non-Newtonian fluid in a box, heated from below, cooled from above, with a fixed linear temperature profile imposed on the side walls. The simulation was developed by the Computational Fluid Dynamics Laboratory at The University of Texas at Austin. We use 100 out of 6000 time steps from the natural convection datasets, where each time step is composed of 48000 tetrahedral elements with 68,921 points. The turbulent vortex dataset is a fluid flow simulation that has been used widely in [25], [37], [49] and each time step is composed of 2,097,152 points. The Isabel dataset is the IEEE Visualization 2004 Contest dataset and each time step has 25,000,000 data points. We use velocity magnitudes of the Isabel data for the functional representations. 80 and 48 time steps are used for the turbulent vortex and the Isabel datasets respectively. Note that since these turbulent vortex and Isabel datasets were captured coarsely over time and our system produced all independent encodings for these two datasets, we created smoother time steps using a linear interpolation between time steps in order to test our technique. The galaxy data is the IEEE Visualization 2008 Contest dataset [47] and each time step has 36,902,400 data points. We use the logarithm

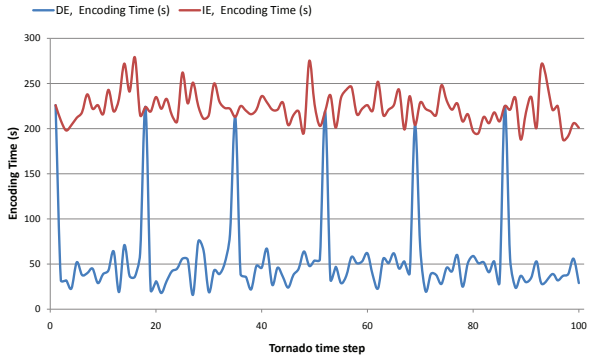
TABLE 1

Data statistics and our encoding results. IE indicates the independent encoding without considering temporal coherence and DE indicates the time dependent encoding proposed in this work. The numbers represent whole time steps.

Dataset	number of time steps	input size (MB)	number of functions from IE	number of functions from DE	storage from IE (MB)	storage from DE (MB)	encoding time from IE (hours)	encoding time from DE (hours)
Tornado	100	88	26,840	6,442	0.75	0.18	6.2	1.4
Convection	100	177	22,060	7,349	0.62	0.21	10.5	3.6
Vortex	80	671	339,748	71,916	9.51	2.01	586.4	135.4
Isabel	48	4,800	46,924	14,645	1.31	0.41	695.6	204.5
Galaxy	72	10,628	64,803	39,811	1.81	1.11	912.7	499.1



(a) Comparison of the number of basis functions and RMS errors for the tornado data



(b) Comparison of the encoding time for the tornado data

Fig. 10. Comparisons of the number of basis functions and RMS errors (a), and the encoding time (b) for the 100 time steps of the tornado data.

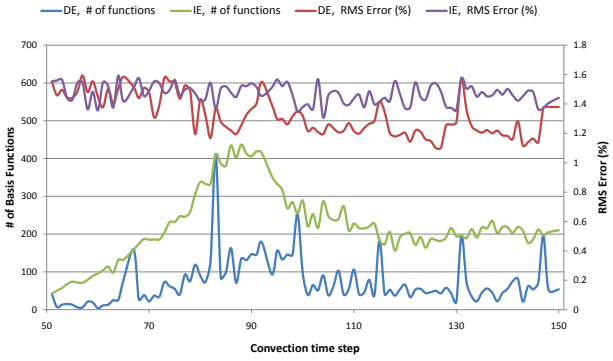
TABLE 2

Comparison of octree and BSP tree for the spatial hierarchy with 80 time steps of the vortex data.

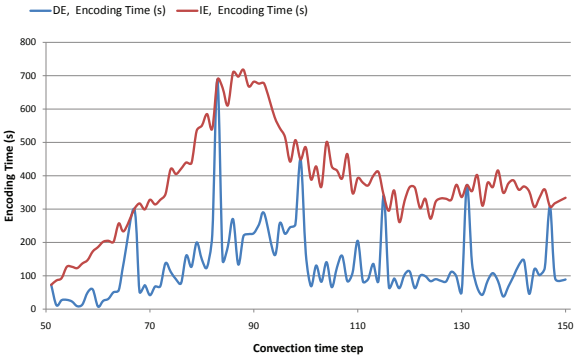
	total number of hierarchical cells	total number of basis functions stored on GPU
Octree	26,533	1,816,677
BSP tree	11,182	932,696

of density values for the functional representations and 72 time steps are used in this work. The datasets are summarized in Table 1.

First, we obtain the functional representations for the four datasets using the proposed encoding approach introduced



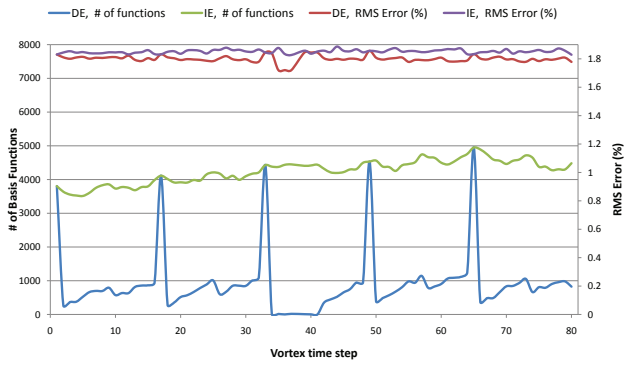
(a) Comparison of the number of basis functions and RMS errors for the convection data



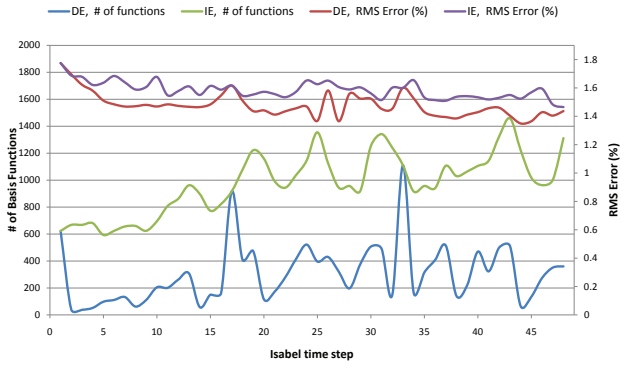
(b) Comparison of the encoding time for the natural convection data

Fig. 11. Comparisons of the number of basis functions and RMS errors (a), and the encoding time (b) for the 100 time steps of the natural convection data.

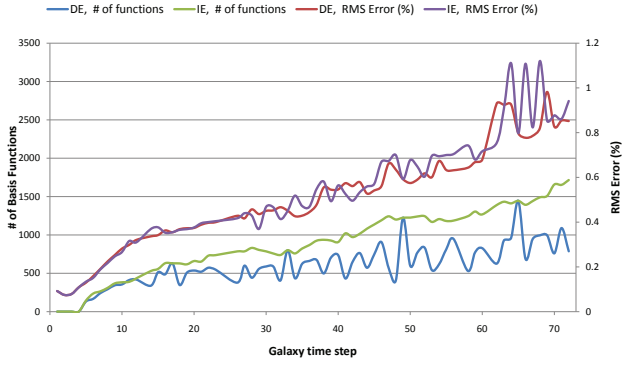
in Section 3. For the comparison, we generate both time independent encoding (IE) and time dependent encoding (DE). Figure 8 (a) and (b) show a comparison between the original vortex data and our functional representation. As presented, our result appears similar to the original data. In order to determine the proper group of frames (GOF), we measured the number of basis functions generated over several GOFs as shown in Figure 8 (c). As seen in the figure, reseting the time encodings at every 15 to 17th steps produces the best results. If we have a small GOF, there are too many IEs and if we have a large GOF, the number of basis functions tends to increase for DEs (see Figure 12(a)). Figure 9 shows the encoding results according to e_{th}. As seen in the figure, the number of basis functions and the encoding time decrease as e_{th} increases, whereas, the RMS



(a) Comparison of the number of basis functions and RMS errors for the vortex data



(b) Comparison of the number of basis functions and RMS errors for the Isabel data



(c) Comparison of the number of basis functions and RMS errors for the galaxy data

Fig. 12. Comparisons of the number of basis functions and RMS errors for the vortex (a), Isabel (b), and galaxy (c) data.

errors increase as e_{th} increases.

In Table 1, we show the encoding results including the number of basis functions, data storage for the encoding results, and the timing of the encoding. As seen in the table, our proposed DE provides very significant reduction in the storage and the encoding timing compared to the IE. The overall compression ratio is between 333:1 and 11,707:1. The encoding time is generally proportional to the input data size. Since our encoding uses nonlinear optimization, we have to evaluate all data points in the optimization, therefore, more input data points indicate longer optimization time and encoding time. Entire encoding comparisons for the tornado and natural convection datasets are found in

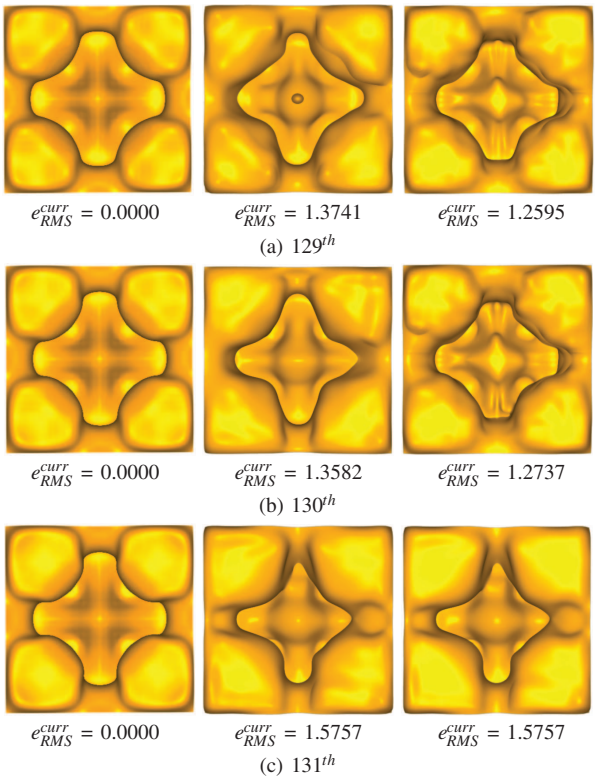


Fig. 13. Comparisons of rendering qualities of the natural convection data for the original (Left column), the time independent encoding (Middle column), and time dependent encoding (Right column). 129^{th} and 130^{th} are *P-frames* and 131^{st} is *I-frame*. RMS errors (%) are presented below images.

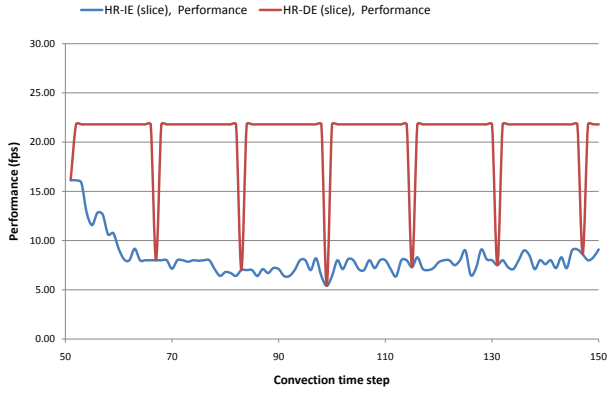
Figure 10 and 11 as graphs. The top rows are the number of basis function comparisons with RMS errors and the bottom rows are the timing comparison. The timing corresponds well with the number of basis functions. Figure 12 presents the number of basis function comparisons with RMS errors for the turbulent vortex, Isabel, and galaxy datasets. As seen in the graphs, our DEs produce much smaller number of basis functions and much shorter encoding time. Moreover, the errors are slightly reduced by our DE since our DE approximates more localized residuals. Figure 13 shows comparisons of the original (Left column), IE (Middle column), and DE (Right column) for three steps of the natural convection data. 129^{th} and 130^{th} are *P-frame*s and 131^{st} is *I-frame*. As seen in the figure, our DE captures major shapes of the data in the *P-frames*. However, there is a sudden change between the last *P-frame* and the following *I-frame* when there is a large RMS difference between them as the sudden changes are visible in other video compression techniques. This is observed in Figure 13 (b) and (c) by looking at the graphs in Figure 11 (a).

Our approach is more favorable than handling the time-varying data independently since many time steps have a similar volumetric pattern. Another advantage of the time dependent encoding is that we encode only the residual data and the residual influences only local area, which means we can easily decompose the encoding results into the hierarchical structure introduced in Section 4.1. Figure 7 shows that the residual basis functions are well

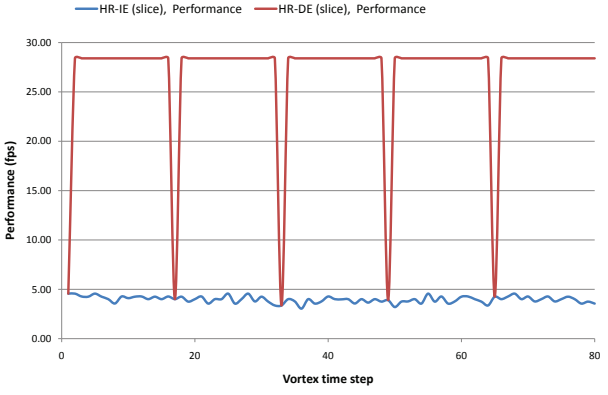
TABLE 3

Performance statistics for the four datasets. The performances are corresponding to Figure 15 and 16. GR indicates the global rendering, HR the hierarchical rendering, DE the time dependent encoding, IE the time independent rendering. The timing is measured on viewport 600x600 with 256 slices and 256 ray samplings.

Dataset	Time step	Frame type	number of functions DE	number of functions IE	GR-DE (slice) (fps)	HR-DE (slice) (fps)	GR-DE (raycasting) (fps)	HR-DE (raycasting) (fps)	HR-IE (slice) (fps)	HR-IE [13] (fps)
Tornado	16^{th}	*P*	38	330	5.3	21.7	7.2	32.5	5.8	2.1
	69^{th}	*I*	251	251	1.5	5.6	1.8	5.6	5.6	3.2
Convection	80^{th}	*P*	90	338	4.0	21.8	6.2	32.1	5.3	2.6
	99^{th}	*I*	254	254	1.3	5.4	1.5	5.3	5.4	2.9
Vortex	68^{th}	*P*	489	4,593	1.2	28.4	1.3	32.5	3.5	0.6
	49^{th}	*I*	4,533	4,533	0.1	3.9	0.1	5.1	3.9	0.6
Isabel	34^{th}	*P*	164	916	4.9	31.2	5.3	64.2	10.6	4.6
	1^{st}	*I*	623	623	1.5	12.8	2.3	16.1	12.8	8.3
Galaxy	30^{th}	*P*	589	786	0.5	21.2	0.9	32.2	7.1	3.1
	65^{st}	*I*	1,445	1,445	0.3	3.4	0.3	3.6	3.4	2.0



(a) Rendering performance for the convection data

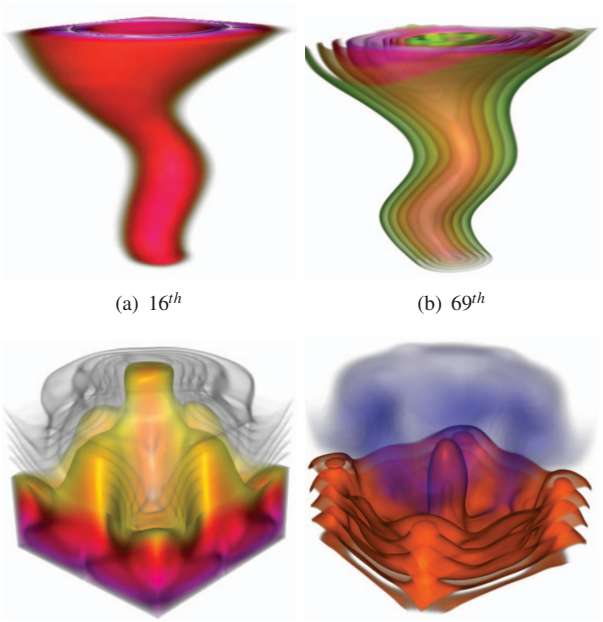


(b) Rendering performance for the vortex data

Fig. 14. Comparisons of the rendering performances for IE and DE using the convection (a) and vortex datasets (b).

localized as shown in (b) and there is no need to split the volume into smaller subcells because there are not many large influencing basis functions. Therefore, a smaller hierarchy level is sufficient to decompose the volume into interactively renderable subcells. As presented in Table 2, moreover, our BSP tree based decomposition provides a much smaller number of cells and basis functions compared to the octree based spatial decomposition; therefore, we can save graphics memory. As a hierarchical data representation for comparison, there is an approach, wavelet [3], [6], [12], [26], [27], [39], [40], [41]. In the area of volume encoding, most work [12], [26], [27] has been performed for regular grids and shows effective 3D compressed volumes and rendering. Recently, this research has been extended to irregular grids organized using polygonal meshes [6], [40], [41]. The wavelet encoding of irregular grids is usually performed using irregular sampling and adaptive subdivision. However, the wavelet for the irregular data set is based on polygonal meshes. Therefore, it is not easily extended to arbitrary scattered volume data. Sohn et al. [39] present a compression scheme for encoding time-varying isosurface and volume features. They encode only the significant blocks using a block-based wavelet transform. However, all of these techniques are for regular grid datasets and it is difficult to extend to scattered time-varying datasets. On the other hand, our encoding is able to handle any scattered datasets regardless of grid structure.

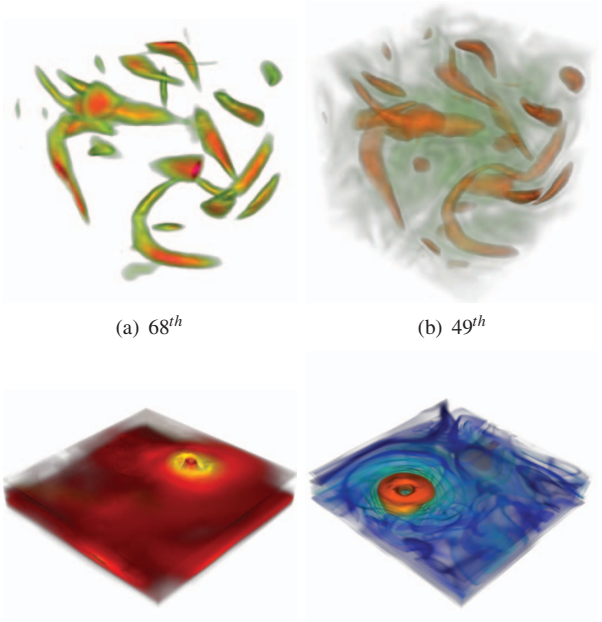
Figure 14 shows rendering performance comparisons for IE and DE using the convection (a) and the vortex (b) datasets over the time steps. The rendering performance is measured on a 600x600 viewport with 256 slices and 256 samplings per ray. As seen in the figure, DE provides good rendering performances compared to IE. The constant performance in DE is obtained since the residual encoding fits very well in our spatial data structure. Table 3 presents our rendering performance results for two time steps of each dataset. In this work, we implement both slice based and ray casting volume rendering for comparisons. Moreover, we compare two encoding schemes (IE and DE) and the efficiency of our hierarchical spatial structures. The performances presented in Table 3 corresponds to the images shown in Figure 15, 16, and 17. We select one *P-frame* and one *I-frame* for each dataset. In general, *P-frames* have a much smaller number of basis functions compared to *I-frames* because the *P-frame* data is already represented by the previous *P-frames* and the *I-frame*.



(a) 16^{th} (b) 69^{th}

(c) 80^{th} (d) 99^{th}

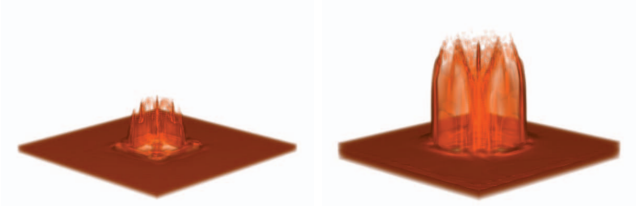
Fig. 15. Screen shots of the tornado (a and b) and convection (c and d) data. *P-frames* are placed in the left column and *I-frames* are in the right column.



(a) 68^{th} (b) 49^{th}

(c) 34^{th} (d) 1^{st}

Fig. 16. Screen shots of the vortex (a and b) and Isabel (c and d) data. *P-frames* are placed in the left column and *I-frames* are in the right column.



(a) 30^{th} (b) 65^{th}

Fig. 17. Screen shots of the galaxy data. *P-frames* are placed in the left column and *I-frames* are in the right column.

This fact is directly related to our global rendering (GR), which means that we evaluate all basis functions for all volumes, and the smaller number of basis functions is used to accelerate the rendering speed. If we use the hierarchical spatial structure, we can further increase the performance because we need only to evaluate the basis functions that influence a location in the volume. However, this hierarchical rendering (HR) depends on the locality of the basis function and if the majority of basis functions influence the entire volume, we are not able to obtain a significant performance improvement. Fortunately, our encoding system mostly generates more narrow influencing basis functions than broad influencing basis functions due to the characteristic of the residual data. Therefore we are able to achieve a large performance increase with the HRs. This is observed in the HR columns compared to the GR columns in the tables. As a comparison between the slice based volume rendering and the ray casting volume rendering, the tables provide the performance comparisons. When we render the volume with more transparency, the ray casting does not provide a significant performance increase. On the other hand, in Figure 15 (a) and (c), we increase opacity and we can see an increased speedup compared to other more transparent volumes. Also we compare the efficiency of using the 3D texture for the hierarchical structure to search for the texture coordinates in the parameter 2D textures with the previous work [13]. In the previous work, all subcells were traced to find the slice intersections on CPU and if there are too many subcells, this degrades the performance. This is observed between the last two HR-IE columns of both tables especially for the vortex data. Since we can find the texture coordinates easily by fetching the 3D texture, the large number of subcells does not affect the performances. As introduced in Section 4.3, we use 3D textures to store the previous *P-frames* and *I-frames* using the *render to 3D texture* technique. This technique discards the evaluation for the previous *P-frames* and *I-frames* for the current time step. The column HR-DE (slice) and HR-IE (slice) show that using our time dependent rendering gives us much better performances. However, as mentioned in Section 4.3, this could cause a problem when we have very high resolution datasets since this is similar to resampling of the data with the basis functions. We will investigate this issue in the future.

6 Conclusion and Future Work

In this paper, we presented time-varying data visualization using functional representations. Instead of a naive extension of the previous functional representations, we have developed a more efficient encoding scheme using the temporal similarity between the time steps. This encoding approach dramatically speeds up the encoding timing while significantly reducing the number of basis functions, since we need to encode only the residual from the previous functional representations. We presented our

results using time independent representations, showing that our approach has significant advantages for functionally representing and compressing time-varying data. We also presented an evaluation and visualization technique for our proposed encoding approach. This work demonstrates that time-varying functional representations for time-varying data can be evaluated quickly compared to handling all time steps independently. Moreover, we compared various rendering techniques including new slice based and ray casting volume rendering, as well as previous work [13]. As future work, we would like to study more aggressive techniques, such as prediction-corrections, for datasets which are not much coherent between time steps. Another future work will be to reduce the sudden changes at *I-frames*. Moreover, we will investigate more efficient rendering methods for the *P-frames* since the *render to 3D texture* approach could degrade the data precision due to the texture dimensions.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for many helpful comments. This work was supported in part by the Swiss National Science Foundation under grant 200021_124642.

REFERENCES

[1] F. F. Bernardon, S. P. Callahan, J. L. D. Comba, and C. T. Silva. Interactive volume rendering of unstructured grids with time-varying scalar fields. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 51–58, 2006.

[2] F. F. Bernardon, S. P. Callahan, J. L. D. Comba, and C. T. Silva. An adaptive framework for visualizing unstructured grids with time-varying scalar fields. *Parallel Computing*, 33(6):391–405, 2007.

[3] M. Bertram, M. A. Duchaineau, B. Hamann, and K. I. Joy. Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization. In *IEEE Visualization 2000*, pages 389–396, October 2000.

[4] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 67–76, August 2001.

[5] J. Chen, D. Silver, and M. Parashar. Real time feature extraction and tracking in a computational steering environment. In *Proceedings of the High Performance Computing Symposium, HPC2003, Society for Modeling and Simulation International*, pages 155–160, 2003.

[6] I. Daubechies, I. Guskov, P. Schröder, and W. Sweldens. Wavelets on irregular point sets. *Phil. Trans. R. Soc. Lond. A*, 357(1760):2397–2413, 1999.

[7] W. de Leeuw and R. van Liere. BM3D: motion estimation in time dependent volume data. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 427–434. IEEE Computer Society, 2002.

[8] W. de Leeuw and R. van Liere. MCMR: a fluid view on time dependent volume data. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 149–156. Eurographics Association, 2003.

[9] R. Franke and H. Hagen. Least squares surface approximation using multiquadrics and parametric domain distortion. *Computer Aided Geometric Design*, 16(3):177–196, March 1999.

[10] R. Grosso and T. Ertl. Mesh optimization and multilevel finite element approximations. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics*, pages 19–30. Springer Verlag, Heidelberg, 1998.

[11] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. H. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[12] I. Ihm and S. Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18(1):3–15, 1999.

[13] Y. Jang, R. P. Botchen, A. Lauser, D. S. Ebert, K. P. Gaither, and T. Ertl. Enhancing the interactive visualization of procedurally encoded multifield data with ellipsoidal basis functions. *Computer Graphics Forum*, 25(3):587–596, 2006.

[14] Y. Jang, M. Weiler, M. Hopf, J. Huang, D. S. Ebert, K. P. Gaither, and T. Ertl. Interactively visualizing procedurally encoded scalar fields. In *EG/IEEE TCVG Symposium on Visualization VisSym '04*, pages 35–44, 339, 2004.

[15] L. Jiang, H. Liu, M. Parashar, and D. Silver. *Rule-Based Visualization in a Computational Steering Collaboratory*. Lecture Notes in Computer Science, Computational Science - ICCS 2004. Springer Berlin / Heidelberg, 2005.

[16] E. J. Kansa. Volumetric radial basis functions methods applied to gas dynamics. In *International Workshop on MeshFree Methods 2003*, 2003.

[17] E. J. Kansa, H. Power, G. E. Fasshauer, and L. Ling. A volumetric integral radial basis function method for time-dependent partial differential equations. i. formulation. *Engineering Analysis with Boundary Elements*, 28(10):1191–1206, 2004.

[18] C.-L. Ko, H.-S. Liao, T.-P. Wang, K.-W. Fu, C.-Y. Lin, and J.-H. Chuang. Multi-resolution volume rendering of large time-varying data using video-based compression. In *IEEE Pacific Visualization Symposium 2008*, pages 135–142, 2008.

[19] C. Ledergerber, G. Guennebaud, M. Meyer, M. Bacher, and H. Pfister. Volume mls ray casting. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1539–1546, 2008.

[20] S. Lefebvre, S. Hornus, and F. Neyret. Octree textures on the GPU. In M. Pharr, editor, *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 37, pages 595–613. Addison Wesley, Mar. 2005.

[21] E. B. Lum, K. L. Ma, and J. Clyne. Texture hardware assisted rendering of time-varying volume data. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 263–270. IEEE Computer Society Press, 2001.

[22] K.-L. Ma and H. Shen. Compression and accelerated rendering of time-varying volume data. In *Proceedings of the Workshop on Computer Graphics and Virtual Reality*, 2000.

[23] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems, July 1999.

[24] B. S. Morse, T. S. Yoo, P. Rheingans, D. T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings Shape Modeling International*, pages 89–98, Genova, Italy, May 2001.

[25] C. Muelder and K.-L. Ma. Interactive feature extraction and tracking by utilizing region coherency. In *PACIFICVIS '09: Proceedings of the 2009 IEEE Pacific Visualization Symposium*, pages 17–24, Washington, DC, USA, 2009. IEEE Computer Society.

[26] S. Muraki. Approximation and rendering of volume data using wavelet transforms. In *VIS '92: Proceedings of the 3rd conference on Visualization '92*, pages 21–28. IEEE Computer Society Press, 1992.

[27] K. G. Nguyen and D. Saupe. Rapid high quality compression of volume data for visualization. *Computer Graphics Forum*, 20(3):49–56, 2001.

[28] G. M. Nielson. Scattered data modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, 1993.

[29] G. M. Nielson, T. A. Foley, B. Hamann, and D. Lane. Visualizing and modeling scattered multivariate data. *IEEE Computer Graphics and Applications*, 11(3):47–55, 1991.

[30] S. Park, L. Linsen, O. Kreylos, J. D. Owens, and B. Hamann. A framework for real-time volume visualization of streaming scattered data. In M. Stamminger and J. Hornegger, editors, *Proceedings of Tenth International Fall Workshop on Vision, Modeling, and Visualization 2005*, pages 225–232. DFG Collaborative Research Center, Nov. 2005.

[31] F. Pighin, J. M. Cohen, and M. Shah. Modeling and editing flows using advected radial basis functions. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 223–232. Eurographics Association, 2004.

[32] A. L. Rocca, A. H. Rosales, and H. Power. Radial basis function hermite collocation approach for the solution of time dependent convectiondiffusion problems. *Engineering Analysis with Boundary Elements*, 29(4):359–370, 2005.

[33] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[34] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 371–377. IEEE Computer Society Press, 1999.

[35] H.-W. Shen and C. R. Johnson. Differential volume rendering: a fast volume visualization technique for flow animation. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 180–187, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[36] Z. Shi, Y. Tamura, and T. Ozaki. Nonlinear time series modelling with the radial basis functions-based state-dependent autoregressive model. *International Journal of Systems Science*, 30(7):717–727, 1999.

[37] D. Silver and X. Wang. Tracking and visualizing turbulent 3D features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):129–141, 1997.

[38] D. Silver and X. Wang. Tracking scalar features in unstructured datasets. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 79–86. IEEE Computer Society Press, 1998.

[39] B.-S. Sohn, C. Bajaj, and V. Siddavanahalli. Feature based volumetric video compression for interactive playback. In *VVS '02: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 89–96. IEEE Press, 2002.

[40] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. Wavelets for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.

[41] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM Journal on Mathematical Analysis*, 29(2):511–546, 1998.

[42] G. Turk and J. F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics (TOG)*, 21(4):855–873, 2002.

[43] F.-Y. Tzeng and K.-L. Ma. Intelligent feature extraction and tracking for visualizing large-scale 4D flow simulations. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 2005.

[44] E. Vuçini, T. Möller, and M. E. Gröller. On visualization and reconstruction from non-uniform point sets using b-splines. *Computer Graphics Forum*, 28(3):1007–1014, 2009.

[45] M. Weiler, R. P. Botchen, S. Stegmeier, T. Ertl, J. Huang, Y. Jang, D. S. Ebert, and K. P. Gaither. Hardware-assisted feature analysis of procedurally encoded multifield volumetric data. *Computer Graphics and Applications*, 25(5):72–81, 2005.

[46] R. Westermann. Compression domain rendering of time-resolved volume data. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, pages 168–175,450. IEEE Computer Society Press, 1995.

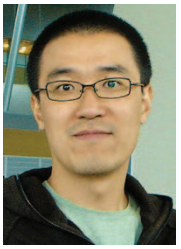
[47] D. Whalen and M. L. Norman. Competition data set and description, 2008. In 2008 IEEE Visualization Design Contest, http://vis.computer.org/VisWeek2008/vis/contests.html.

[48] O. Wilson, A. V. Gelder, and J. Wilhelms. Direct volume rendering via 3D textures. Technical Report UCSC-CRL-94-19, University of California at Santa Cruz, Santa Cruz, CA, USA, 1994.

[49] J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumetric data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 55, Washington, DC, USA, 2003. IEEE Computer Society.

[50] J. Younesy, T. Moller, and H. Carr. Visualization of time-varying volumetric data using differential time-histogram table. *International Workshop on Volume Graphics*, 0:21–29, 2005.

[51] K. Zhou, Z. Ren, S. Lin, H. Bao, B. Guo, and H.-Y. Shum. Real-time smoke rendering using compensated ray marching. *ACM Transactions on Graphics*, 27(3):36, 2008.



Yun Jang is a post doctoral researcher at ETH Zürich, Switzerland. His research interests include interactive visualization, volume rendering, and data representations with functions. Jang received his masters and doctoral degree in Electrical and Computer Engineering from Purdue University in 2002 and 2007 respectively, and received his bachelors degree in Electrical Engineering from Seoul National University, South Korea, in 2000. Contact him at jangy@inf.ethz.ch.

David S. Ebert is the Silicon Valley Professor of Electrical and Computer Engineering at Purdue University, a University Faculty Scholar, a Fellow of the IEEE, and Director of the Visual Analytics for Command Control and Interoperability Center (VACCINE), the Visualization Science team of the Department of Homeland Security's Command Control and Interoperability Center of Excellence. Dr. Ebert performs research in novel visualization techniques, visual analytics, volume rendering, information visualization, perceptually-based visualization, illustrative visualization, mobile graphics and visualization, and procedural abstraction of complex, massive data. Ebert has been very active in the visualization community, teaching courses, presenting papers, co-chairing many conference program committees, serving on the ACM SIGGRAPH Executive Committee, serving as Editor in Chief of IEEE Transactions on Visualization and Computer Graphics, serving as a member of the IEEE Computer Society's Publications Board, serving on the IEEE Computer Society Board of Governors, and successfully managing a large program of external funding to develop more effective methods for visually communicating information. Contact him at ebertd@purdue.edu.

Kelly Gaither Director of Data & Information Analysis at the Texas Advanced Computing Center (TACC), is leading the scientific visualization, data management & collections, and data mining & statistics programs at TACC while conducting research in scientific visualization and data analysis. Gaither, a research scientist, also serves as the area director for visualization in the National Science Foundation funded TeraGrid project. Gaither received her doctoral degree in Computational Engineering from Mississippi State University in May, 2000, and received her masters and bachelors degree in Computer Science from Texas A&M University in 1992 and 1988 respectively. While obtaining her Ph.D., she worked full time at the Simulation and Design Center in the National Science Foundation Engineering Research Center as the leader of the visualization group. Gaither has a number of refereed publications in fields ranging from Computational Mechanics to Supercomputing Applications to Scientific Visualization. She has given a number of invited talks. Over the past ten years, she has actively participated in the IEEE Visualization conference, and served as the IEEE Visualization conference general chair in 2004. She is currently serving on the IEEE Visualization and Graphics Technical Committee. Contact her at kelly@tacc.utexas.edu.