Multi-variate, Time-varying, and Comparative Visualization with Contextual Cues

Jonathan Woodring and Han-Wei Shen

Abstract— Time-varying, multi-variate, and comparative data sets are not easily visualized due to the amount of data that is presented to the user at once. By combining several volumes together with different operators into one visualized volume, the user is able to compare values from different data sets in space over time, run, or field without having to mentally switch between different renderings of individual data sets. In this paper, we propose using a volume shader where the user is given the ability to easily select and operate on many data volumes to create comparison relationships. The user specifies an expression with set and numerical operations and her data to see relationships between data fields. Furthermore, we render the contextual information of the volume shader by converting it to a volume tree. We visualize the different levels and nodes of the volume tree so that the user can see the results of suboperations. This gives the user a deeper understanding of the final visualization, by seeing how the parts of the whole are operationally constructed.

Index Terms—multi-variate, time-varying, comparative, focus + context

1 Introduction

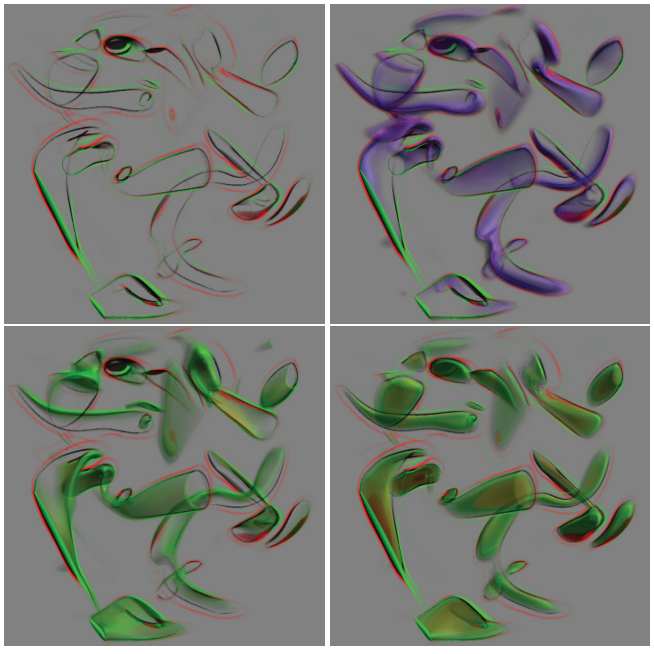

Fig. 1. From left to right, top to bottom: a set intersection operation result with two volumes A **and** B of clear, A **out** B, B **out** A, and B **in** A. The silhouette context of the volumes A, B, and A **in** B is rendered as well.

Scientific visualization has moved from the exploration of single scalar data sets to multi-variate, time-varying, and comparative data [10, 11]. In the realm of time-varying data, animation is a common way of exploring a time series data. While animation is adequate for expressing time evolution, it requires effort from the user to analyze space-time-value relationships. For example, if the user wants to see the relationship between time step A and B, she has to fast-forward and reverse the animation, or jump between animation frames. Even then, the relationships between value and space in time are not entirely clear, due to the loss of positional information. The same is true for comparative visualization, where we compare several different views of the data at the same time.

An alternative method to explore multi-field, time-varying, and comparative data is to combine all data into one volume [1]. High dimensional projection [21, 22] for time series data allows the user to combine multiple time steps into a single volume. Unfortunately, the user is limited to how many time steps can effectively be visualized at the same time because of color composition techniques used. We use the basic primary colors, i.e., red, blue, and green, for three fields, then the volume intersections from high dimensional projection appear as cyan, magenta, yellow, or white, depending on the region. Using more sophisticated color schemes, it might be possible to expand beyond three fields. However, the hues that humans can distinguish are rather limited. Furthermore, the user cannot use the transfer function to visualize values at the same time as volume intersections, because color is reserved to indicate intersection regions.

The number of time steps that can be visualized at once is limited due to space filling and occlusion, and it might not be possible to even see volume intersection areas. An alternative technique for combining multiple field data into one volume uses a different conceptual framework [4]. They are able to solve some of the occlusion problems by adding on volume inclusion and exclusion operations. These are based upon the alpha channel where both operations determine the output opacity of a point. They do not consider all possible set operations that could be performed between volumes. Other set operations could be added on as special cases, but that would be an non-integrated solution.

Both methods mentioned above combine volumes with alpha composition using the transfer function from the field data. These do not allow the user to have exact control of the final appearance of a data point. Since a point in space is the result of the combination of colors of several transfer functions, it cannot be directly mapped to one value in one transfer function. Additionally, both methods only show the final visualization of multiple data fields, but not the intermediate results that lead to the final image. For example, the user may wish to see the intersection of multiple fields in space, thereby excluding the data that are not included in the intersection. If the user generates

- *Jonathan Woodring is with the Department of Computer Science and Engineering at the Ohio State University, email: woodring@cse.ohio-state.edu*
- *Han-Wei Shen is with the Department of Computer Science and Engineering at the Ohio State University, email: hwshen@cse.ohio-state.edu*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

a complex composition involving all available data, she may be presented with an image that is completely blank. The user does not have any contextual information on what went wrong in her visualization, especially if she does not know what the original data look like.

To solve these problems, we propose a flexible multi-variate, time-varying, and comparative visualization scheme that is able to handle an arbitrary number of variables and time steps. We achieve this by allowing the user to specify a volume shader expression combining data volumes with operations. In our operators, we use an expanded series of set operations that take into account all regions between multiple volumes, allowing the user to locate specific data regions and solving occlusion problems. The set operations are implemented with a general framework that can handle an arbitrary number of data fields, which is also efficient and easy to understand. While constructive volume geometry (CVG) [5] describes a set algebra for boolean operations, we provide a mathematical framework that makes it simple to implement with the state-of-the-art hardware volume rendering techniques. In addition, we can perform numerical and statistical operations in the same framework. With multiple data fields and complex volume shader expressions, the user needs to see the individual parts of the data, the suboperations of the final result, and how her result was generated. In response, we convert the volume shader expression into a tree graph, called the *volume tree*. In a volume tree, operations and volumes are nodes in the tree, and edges are inputs and outputs. This is realized through the visualization of the suboperations, levels, and nodes of the volume tree along with the final result. Leveraging this visualization, the user is able to make an better informed visualization and gain a greater understanding of the data.

2 Related Work

There are different techniques that combine several volumes together into one volume. We have previously mentioned Woodring et al., Chen and Tucker's, and Cai and Saka's techniques for combining volumes together, and their problems that we address in this paper [4, 5, 21, 22]. It might be possible to render multi-field data with higher dimensional projections, slicing, and transfer functions [1, 10, 13, 20], however, we completely avoid going to higher dimensions by working in a shading framework.

Shading languages are ubiquitous today, and many implementations of shading languages exist, which can be attributed to Hanrahan and Lawson [9, 18]. More recently McCormick et al. [16] have developed a procedural volume shading language that generates and compiles code for the CPU and GPU for visualization. Porter and Duff [17] presented a unified technique and novel mathematics for image composition. Our set operations are inspired by their ideas. We also attempt to extract interesting features for the user, from our operations which have been worked on in the past [13, 14].

Our paper uses ideas of selection sets of multiple volumes and the combination of selection sets. Bruckner and Gröller [2] provide a system for volume illustration of selection sets. We expand upon their method utilizing multi-field data, and also generalizing and simplifying the method of volume combinations without using intersection transfer functions. Doleisch et al. [7] provide methods for feature set selection in visualization.

The volume tree is derived from the shader expression that describes how volumes are joined together to form a comparative visualization. For user interaction and construction of the volume tree, we utilize visualization and image spreadsheets [12, 15]. In the context rendering of the volume tree, we utilize non-photorealism where we use line drawing techniques [3, 6, 8, 19] to create silhouettes, and express the existence of volume data without explicitly rendering the entire volume.

3 Overview of Our Approach

Our fundamental principle for comparative visualization is that the data fields should be rendered together within the same space for user comparison. The visualization can extract the comparative information and present how data differ at a point in space, without asking the user to compare the data manually. In this paper, we present a comparative visualization technique where per-point operations are performed on the data. Assuming that each data field lies in the same coordinate system, the renderer applies a user defined volume shader expression.

The volume shader is an expression composed of operations and data fields. Such an example is given in Equation 1, where o is the output volume derived from a user expression consisting of operations op and input data volumes v. Note that a Backus-Naur Form (BNF) can be derived for our volume shader expression. For brevity, we do not express a BNF in this paper, but we refer the reader to shading language and expression literature [9, 18] for the detail of using BNFs. The final rendering result o is a volume that represents the application of the volume shader at every point in space. The user has complete flexibility in terms of what they want to visualize, provided that a sufficient toolbox of operations for data comparison is given.

$$o = op_1(op_2(op_3(v_1, v_2, ...), ...), op_n(...), ...) \quad (1)$$

In the context of multi-field, time-varying, and comparative visualization, we specify two kinds of operations on the data, *set operations* and *numerical or statistical operations*. The set operators give the capability of a "visual database search" of the data. It allows the user to visualize examples such as, given a pressure p and a temperature t, showing all points where $p_0 < p < p_1$ and $t_0 < t < t_1$. This gives the user a way to isolate data to a specific area of interest. Furthermore, it helps us cope with space filling and occlusion issues. Traditional techniques of rendering two volumes intersecting in space make it difficult for us to observe the intersections, because they are often occluded by the non-intersecting portions. Using set operations, the user can make a query like "just show me the intersecting data in space". The user can also extract specific regions from data by defining a volume mask, where the set operations cull the data accordingly.

Other methods have used the appearance of a volume after the application of the transfer function to composite volumes together. A point in a composite volume may be the combination of colors from two or more transfer functions. The user loses control over the appearance of the data. Moreover, she also loses the ability to see what value a point has, because the final color may be different than the transfer function mapping. We provide *value-based* set operations that output data volumes rather than color volumes, in addition to *appearance-based* set operations. Thus, the user can control the visual appearance of a data point, since the data will be mapped by one transfer function, rather than the color combination result of several transfer functions.

Numerical or statistical operations can be used in the volume shader expression to generate a new data field based on the volume inputs. Rather than preprocessing and storing all analyses as a separate data field, the user can perform numerical operations on the fly and make data-to-data comparisons. There might not be the time or the forethought to create all numerical or statistical tests on data. It seems natural to include numerical operations, and they allow the user to extract more information as needed.

With long shader expressions and operator nesting, it can be a daunting task to understand the results. This difficulty arises due to the reason that the user does not see how the final visualization is created from the individual parts. We ease the understanding by providing more information about the volume shader. This is done through the visualization of the suboperations and input data along with the final result. We convert the volume shader to a volume tree. Figure 2 is a example of a volume tree that is derived from a volume shader expression and a linked visualization spreadsheet [12], which is described in section 6.1. The volume tree consists of operations and volumes as nodes, and inputs and outputs as edges. By the visualization of tree levels and nodes along with the visualization spreadsheet, the user see how the sources change progressively through the volume tree to the final result. Observing each operation in context with the final result, the user gains insight that may not be entirely clear or present in the result. This volume tree representation also gives the user the opportunity to fine tune the results. By adjusting each suboperation of the volume shader, the user can generate a more precise visualization. It

even allows the user to locate errors in her visualization if she has made a mistake at some stage of the volume shader.

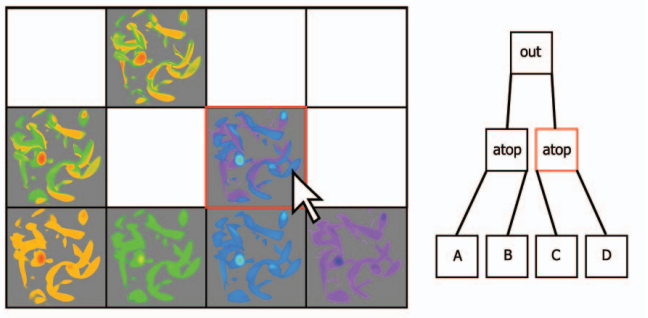

Fig. 2. An example of a volume tree and visualization spreadsheet displayed for the user. Leaf nodes are input data fields and internal nodes are operations. Edges indicate input and outputs. The user constructs the volume shader by selecting two or more data sets and choosing the operation to apply.

In addition to the visualization spreadsheet, we provide two methods of rendering the volume tree context: *level-of-detail* and *animation*. For the level-of-detail method, we render selected suboperations and input at a lower detail level along with the final result. Lower detail levels can be realized through transparency, shells, and line drawing [3, 6, 8], so that the final result is not completely occluded. The animation method allows the user to move through the levels of the volume tree so that she can see how the data evolves in the same space.

4 Set Operators

A *set operator* is a binary operator that takes two data fields as inputs, and it outputs a field resulting from performing per point set tests. We give the user set operations to extract the interesting portions of the data. Our set operators and tests are inspired by the Porter and Duff compositing operators [17]. Recall that with their compositing operators, given two input images, it simply states how two images can be composited together in a mechanical, unified way. Although the graphics community knew how to perform compositing before their operators came out, it was largely ad-hoc and turn-key solutions. Porter and Duff unified image compositing and made it very simple to specify arbitrary composition. High dimensional projection [21, 22] does not allow for an arbitrary combination of operators, only one operation is applied for all data. The multi-volume rendering in [4] only considers inclusion and exclusion and not in a straightforward manner. CVG [5] provides boolean algebra, but does not specify implementation mathematics. In response to these, we provide a unified, mathematical method for any arbitrary combinations of data. The method also conforms to using hardware volume rendering techniques.

In our set operators, we avoid using N-dimensional transfer functions [13] for the data. Considering the case of two fields, we wish to show the intersection of data between two interval volumes. This is quite simple to do with a 2D transfer function. Once we progress to four fields or more, it becomes difficult for the user to define the transfer function, besides trying to implement it in graphics hardware. In our method, we do not actually have to consider the transfer function to perform set operations. This allows us, in certain cases, to ignore the transfer function until rasterization. This also allows us to draw a distinction between two types of set operations: *appearance-based set operations* and *value-based set operations*. The former applies the transfer function and combines data together based upon their color volumes. The latter performs volume set operations in data space and applies the transfer function later.

4.1 Appearance-Based Set Operator

In [17], we have Equation 2 that specifies the output color at a point c_O, given two pre-multiplied alpha input colors c_A and c_B. The type of compositing operation performed is determined by a lookup table which determines the values of F_A and F_B from the alpha values of c_A and c_B. The compositing equation is simple to implement given the lookup table, and it covers all composition possibilities.

operation	F_A	F_B
clear	0	0
A	1	0
B	0	1
A **over** B	1	$1-w_A$
B **over** A	$1-w_B$	1
A **in** B	w_B	0
B **in** A	0	w_A
A **out** B	$1-w_A$	0
B **out** A	0	$1-w_B$
A **atop** B	w_B	$1-w_A$
B **atop** A	$1-w_B$	w_A
A **xor** B	$1-w_B$	$1-w_A$

Table 1. Lookup table for F_A and F_B to determine set operations

$$c_O = c_A F_A + c_B F_B \tag{2}$$

For our volume composition, the appearance-based set operator is the same as Equation 2. It operates on two color volumes, where the transfer function is applied to data volumes. c_A and c_B are pre-multiplied alpha color points in the color volumes A and B. c_O is an output color at a point in the resulting volume of the set operator. In addition, we have a lookup table for the set operations that we wish to perform, for F_A and F_B in Table 1. Examples of applying appearance-based set operators on two volumes can be seen in Figures 3, 4, 5, and 6. We use a hurricane simulation data set. Set A is a data range defined on the pressure field from the hurricane simulation, while set B a set defined on the precipitation field.

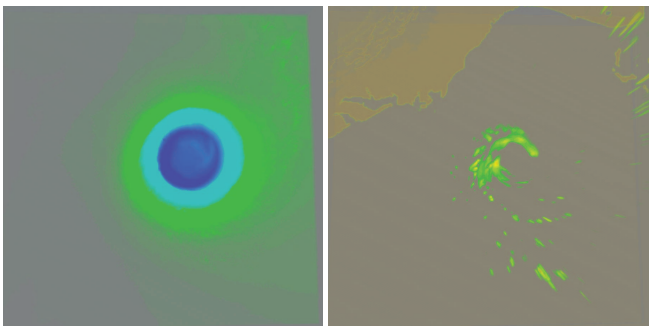

Fig. 3. Two volume sets with A on the left and B on the right.

Note that instead of F_A and F_B being determined by the alpha values of c_A and c_B, as in [17], we introduce two new scalar values, w_A and w_B. Here, w is the weight scalar in the range of [0, 1] for a point. A weight is defined for each point in each volume A or B. It indicates the membership for a point in the input volume sets which participate in the set operation [2]. A w of zero means that a point does not belong to the set of its volume, while any value greater than zero means that a point does belong to the set of its volume. A weight of less than one allows the user to determine the blending factor between color volumes. With fractional set weights, the user can specify that only 50% of a volume, for example, blends with any other volume at most, independent of the transfer function.

Every volume has a set defined within it. The weighting function needs to be specified for every volume, which maps data points to weights. Methods outlined in the reference material [2, 7] can be used to generate the weighting function. The weighting function can also

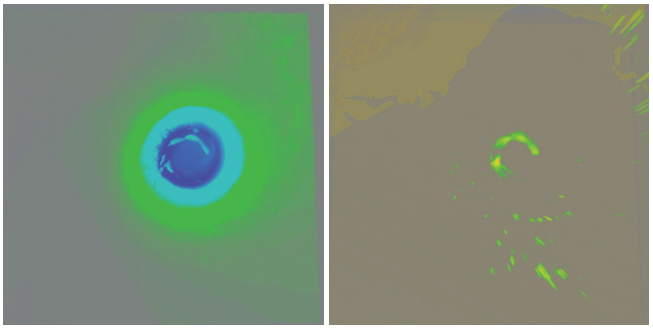

Fig. 4. A **out** B on the left, and B **out** A on the right.

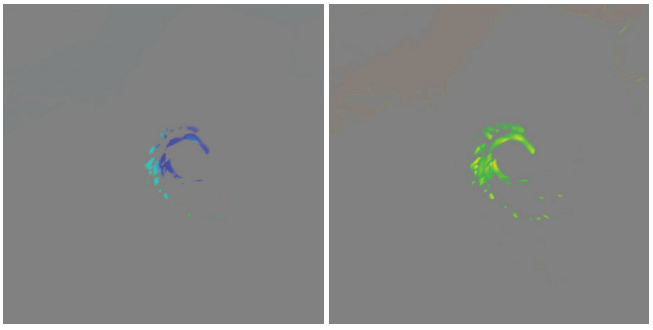

Fig. 5. A **in** B on the left, and B **in** A on the right.

be used to define the volume sets based on spatial regions, rather than on values. The reason that we do not use the alpha value of c_A and c_B for F_A and F_B determination is to allow the user to specify the input sets independently of the transfer function. The input volume set determination for a set operator should be orthogonal to the appearance of volume. Using the alpha channel as the determinant for F values, we assume that the data selection being operated on is the same as the the points that have opacity greater than zero. With the weighting function, the user can utilize it to determine what portion of a data field is part of an operational set independent of the transfer function. For example, the user is able to completely define the appearance of a data field, but then is able to select a portion of it to be operated on. Therefore, she can even select points that are transparent as part of a volume set by using the weighting function.

An appearance-based set operator works on color volumes. When applying a transfer function to a volume and transforming it to a color volume, a weight function is also applied to the volume. In our color volumes, a color volume vector at a point in space is a 5-tuple of (r,g,b,a,w). The output weight w_O for a color point from the input colors is Equation 3, where *clamp* is a function that clamps scalars to [0, 1]. In practice, however, an output weight is not necessarily stored as intermediate data. This is because at run time, the source weights in the volume shader are looked up and intermediate weights are stored as local variables during point processing. They are then discarded after the final color is produced.

$$w_O = clamp(F_A + F_B) \tag{3}$$

4.2 Value-Based Set Operator

The second type of set operator is value-based. The value-based set operators can be used in two additional scenarios. When the user performs a numerical operation on data, she wants to work in the original data space, rather than the color space. Second, if the user wishes to make a precise specification of the appearance of data points in the final result, she wants to use value-based set operations. Appearance-based operations can blend colors together, and therefore, the one-to-one mapping of color to value is lost. Using value-based set operations, we are able to disambiguate this problem, since the color at a point in space is determined by exactly one transfer function. This is the same argument as using pre-classification interpolation versus post-classification interpolation. A transfer function may not be linearly defined, therefore when blending colors, the final color may not be an accurate representation of the value. If we generate data volumes by value interpolation and blending, then apply the transfer function later, we can have a more accurate representation of the value. We introduce two types of value-based set operators we have developed.

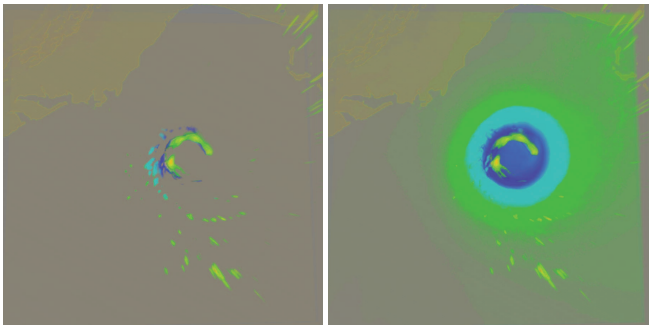

Fig. 6. A **atop** B on the left, and A **xor** B on the right.

The first type of value-based set operator performs value blending, as shown in Equation 4. Here, the constraint is that the two input data fields have to be of the same type for value blending. In Equation 4, d_O is the output data point resulting from the two input data points d_A and d_B. We use the same Table 1 as in appearance-based set operations, to determine the values of F_A and F_B from the function we apply for the set operation.

$$d_O = (d_A F_A + d_B F_B)/(F_A + F_B) \tag{4}$$

Instead of performing a linear combination of colors, we perform a linear combination of values to generate an output point. There is also a re-normalization step in Equation 4. This is to ensure that the value of d_O is not scaled out of the range of the original data. If the fractions sum to zero, then the output data is a not-a-number (NAN) value, which indicates that it has no value. Furthermore, we use Equation 3 to determine the output weight of the data point. This follows that points in a data volume become 2-tuples of (d,w), where d is the data and w is the weight. Though, likewise as in the appearance-based set operator, a calculated weight is not explicitly stored. It is only used and calculated at run time, and then discarded at rasterization. Moreover, the user needs to select the transfer function that is eventually applied to the data during rasterization, or to be used when the data is passed through an appearance-based set operation. The transfer function can come from either input field, because both fields are of the same type. Also, since one transfer function is used, we avoid some of the problems of value ambiguity due to color blending. Although, it is not completely resolved since values are linear combinations of data fields.

In our second value-based set operator, instead of value blending, we output the value that is the input value with the maximum fraction F, as shown in Equation 5. Like previous value-based set operators, if both fractions sum to zero, then the output value is a NAN value.

$$d_O = if\ F_A > F_B\ then\ d_A\ else\ d_B \tag{5}$$

This value-based set operator provides two benefits. First, the output volume can have mixed data types, since we do not attempt to perform value blending and we choose the input value that has the maximum F value. In that way, the user can perform an operation, for example, where she finds the intersection of tensor values with scalar values, without applying the transfer function and appearance-based

set operations. Second, it allows the user to specify the appearance of a data point in the final result completely and exactly. Since the output is a binary decision of which value is the output value from the two inputs, the selected data value's transfer function can be used for the appearance during rasterization. We keep track of this as every output data point will have an index reference i that maps to its transfer function. The output data field has output data points that are 2-tuples of (d,i), where d is the data and i is the transfer function index. During transfer function application, i is used to locate the correct transfer function to apply to the data value. It is also used to look up its weighting function.

4.3 Normal Calculation

One final note to mention is the calculation of normals for correct lighting. We postpone gradient calculation until the final operation or rendering takes place. The gradient is based upon visible points, and not the original data. For example, if a volume is carved out by another volume from a set operation and the normals used are from the original data, then the lighting will not appear correct. We could use the reversed normals from the other volume. However, that only considers the bordering points between the two volumes, and further interior points still have the incorrect gradients. A more general solution for solving this problem is to calculate the gradient in alpha space after the transfer function is applied. To calculate the normal, we apply the volume shader expression to the neighborhood of points to obtain samples. After applying the transfer function for the sample points and obtaining the alpha values, we calculate the gradient on the alpha space and normalize it for the lighting normal. An image showing the correct lighting after a volume A is carved out by a volume B with A **out** B can be seen in Figure 7.

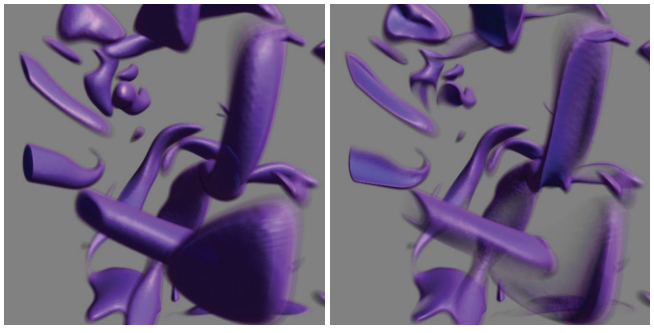

Fig. 7. Correct lighting after a set operation carves away a volume. The volume before operation is shown on the left. The volume after operation, having being carved away, is shown on the right.

5 Numerical or Statistical Operators

For completeness in our model, numerical or statistical operators are included for use in time-varying and comparative scenarios, where the user is comparing volumes of the same data type. With numerical operators, the user can perform a series of computations in her volume shader expression, along with set operators to isolate regions of the data. Numerical operators give the user the capability of additional analyses without any particular forethought of preprocessing the data and storing it as a separate data field. As long as a numerical or statistical operation is applied per point for the input data, it can then be implemented as an operation for a volume shader. Unlike set operators, the user needs to specify a transfer and weighting function on the operator, since it creates a new data field. A set operator derives the appearance and weight from the inputs, while a numerical operation cannot.

Any type of numerical operation that can be used in analysis of a series of data points in space can be implemented as a numerical operator. For scalar data, a max operator finds the maximum of a data point over a series of input data fields. This is useful in time series data if we want to find the point of maximum gain over time. This is also useful in the analysis of comparative data sets. In this case, the user can find areas of extreme deviations from the rest of the data. For vector data, a dot product operator would allow the user to find areas of directional differences. In time-varying vector flow field data, it would show where flow changes direction over time. In a comparative flow scenario, it would show how different models have different flows. For geometry tensor data, we could use a unary operator that transforms the tensor into a more easily-visualized form, such as primary and secondary curvature scalars without preprocessing.

6 User Interaction and Contextual Rendering

The primary method that we use to show the context of the visualization is by converting the volume shader expression into a *volume tree*. Operators and input data fields form nodes and input and output connections between operators form edges. The visualization result is the root of the tree, and all input fields are leaves of the tree. One operational step forward in forming the final result is indicated by one level of the tree. Every subtree forms a suboperation that is part of the final result. We define the context of the volume tree as the suboperations that form the final result. Our goal is to allow the user to construct the volume shader and navigate through the context of volume tree.

6.1 Graphical Representation

The user can directly specify a volume shader through a shader expression. While adequate in functionality, there is a disconnect from the expression and the visualization of the data. To ease the construction of the volume shader for the user, we provide a visualization spreadsheet [12] and volume tree interface, seen in Figure 2. This allows the user to browse all of the available data fields and the results of every step of the volume shader operation. It also gives the user the ability to construct the volume shader in a bottom-up fashion and gives her an interface to adjust the volume shader expression.

The user starts by importing all of her data that she wishes to use into the spreadsheet, which appears in one row, the bottom row, of the spreadsheet. We also display each data set as a volume tree graph node, all at the same vertical level spread out horizontally, mirroring the spreadsheet layout. The user can manipulate the transfer function or weighting functions for a data set by selecting a cell or node. By selecting one or more cells or nodes, the user can direct the system to operate on them. The resulting operation is displayed in a new spreadsheet cell showing the user the results of the operation. The operation also creates a new node in the volume tree, connecting the source data nodes by edges to the new node.

The new graph node attempts to place itself in an appropriate manner by positioning itself one vertical level above the highest source node. Its horizontal position is the horizontal average of the source nodes. If that position is already occupied, it places itself in the nearest free horizontal position to its desired destination. The new corresponding spreadsheet cell attempts to automatically position itself to mimic the layout of the volume tree graph. The spreadsheet coordinates can be seen as grid structuring of the volume tree graph. When a new node is created, the graph space is transformed to spreadsheet coordinates and the new cell is placed there. Likewise, if that spreadsheet cell is occupied, the new cell places itself in the closest column to its desired destination. The user is allowed to reposition nodes or cells, but we do not adjust current node or cell positions when placing new nodes and cells. If we repositioned them, the user would get lost or confused, because we have broken from their current mental image of her progress in the visualization.

We also provide a linked view of the volume tree to the visualization spreadsheet by highlighting linked cells or nodes when the user hovers over one. The rotation and translation of the cells are linked as well. The volume shader is run for the current cell the user is interacting with so she has real time updates. When the user stops interacting with that cell, the spreadsheet runs the volume shader for all the other cells to update them. By selecting a node or cell, the user can also change the operation, the weighting function, or the transfer function to apply to that data. This allows the user to fine tune the visualization.

It also provides a method for the user to keep the same volume shader but change the source data cells and nodes. Additionally, the user can select nodes, levels, or branches of the volume tree and order the system to render that data as contextual information in the root node.

Now we have a framework to be able to explore the volume shader. The next section we focus on how to render or display volume tree contextual information with the final result of the volume shader. We provide two ways of visualizing the volume tree nodes. The first is a level-of-detail method that shows the suboperations rendered in conjunction with the final result. The suboperations are rendered with less information or detail, so that they do not completely occlude the final result. In the second method, we animate through different portions of the volume tree to show how input data are filtered through the volume shader for the purpose of data highlighting. We also show how to explore the parameter space of the volume shader.

6.2 Level-of-Detail

As mentioned previously, the level-of-detail context rendering method displays suboperations in the volume tree in conjunction with the final result at a lower detail. We have several different notions of lower detail that we will further explain later in this section. The lower detail is provided to give a hint of the space and value of the suboperational results, without completely occluding the final result. If the user wishes to focus entirely on one suboperation because more information is needed, she can select that suboperation in the volume tree as the final result of the visualization, and terminate the rest of the visualization past that point. Which level-of-detail to use for a suboperation is determined by the distance from the root node in the volume tree. A suboperation has a lower detail that it is displayed at the further that it is from the final result node in the volume tree. Therefore, things that have more relevance to the final result are shown at a higher detail than suboperations that are further down the tree.

We present three methods for rendering a lower level-of-detail. The first is modulation of the opacity based on the distance from the root. With modulation of the opacity, the suboperations further away from the final result in the volume tree are much more transparent, that is seen as an example in Figure 8. The second method is to use volume shells by using a 3D edge detector to extract the shell. These are also opacity modulated by their volume tree distances. Finally, we can render line depictions of volumes, using silhouettes and feature lines, as in Figure 1. These are also opacity modulated by their distances. By using an exponential opacity modulation function, we can automatically limit the suboperation distance from the root to reduce visual clutter. Rendering too many levels of the volume tree at once would be difficult for the user to extract relevant information. By isolating the suboperation context rendering to one branch of the volume tree, we can emphasize the filtering and evolution from the perspective of an individual data field.

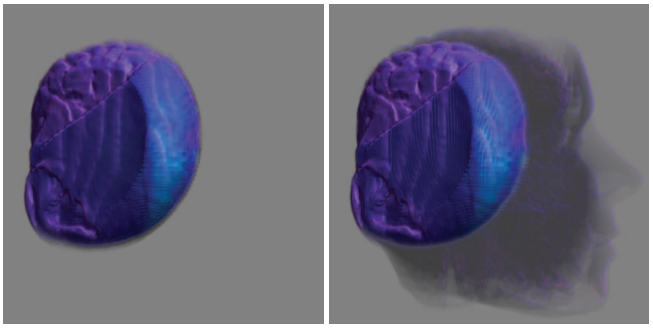

Fig. 8. The end result of an volume shader, A **in** B, is shown on the left. The context of the shader down one branch, how volume B is changed, is shown at a lower opacity on the right. This is also an example of two frames of context animation; from not showing the context to showing the context.

There is also a need for rendering priority order when displaying suboperation data points that coincide at the same 3D location. We can determine a priority order of what to display at a point in space by using a breadth-first-search from the root. The first suboperation of the volume tree that has an opacity greater than zero is the displayed point in space. The reasoning for this is that when rendering several levels of the volume tree together, it is assumed that there is going to be color depth blending. If we do not perform 3D blending between suboperations that intersect in the same space, then opacity does not accumulate as fast as it normally would in depth blending. Furthermore, if there is only one type of color blending due to depth blending, then the color at a point is closer to its original color in the transfer function and eliminates some value ambiguity.

6.3 Contextual Animation

The level-of-detail rendering cannot show all contexts at once, because the screen space is limited and the depth complexity of a volume tree is arbitrary. To solve this problem, we propose additional methods to browse the context of a volume tree. We provide animation through the volume tree. This allows the user to see all the contextual information or the evolution of the data, depending on the types of animation we use.

The first method of animation that we use is changing the root node or final rendering of the volume tree. This is done by *walking* the tree to a suboperation and displaying that as the final rendering result. The volume tree walk can be to a child, parent, or sibling of an operation. The path that is taken by walking the tree describes the context of the volume tree in different ways. By walking up or down one branch of the volume tree, the user can see how one data field evolves operationally into the result. Such an example can be seen in Figure 8. On the other hand, by walking from sibling to sibling, the user can see all the inputs that go to one suboperation of the volume tree.

When moving from one node to the next, the change in suboperation rendering is alpha blended over time so that the target suboperation fades into view, while the old suboperation fades out of view. The animation can be combined with the level-of-detail context rendering. Walking the rendering focus to a new suboperation would cause more information to be displayed for that node. All suboperations that are now closer to the new root from the walk have higher level detail as well. The suboperations now further away from the new root would have less information displayed.

If there is a very deep or wide volume tree and many nodes are displayed, it may be difficult to see individual results. Our second way of animating the volume tree is to fluctuate the opacity of the levels in the tree for the level-of-detail context rendering. This allows the visualization to remove occluding data and the user is able to see individual suboperations better. The fluctuation of the volume tree levels peels away the context information, showing just the final results of the visualization.

We can also animate the data by providing positional or transfer function animation to different regions of the data. The color animation or motion acts as a filter to human perception. In this scenario, whatever is animated stands out in comparison to the static data, and this can be important when many different volumes are shown at once. When we have several levels rendered together, it may be difficult for the user to tell which suboperation corresponds to a particular spatial region in the visualization. The user can select the volume tree suboperation that she is interested in. The selected data then move in a periodic fashion or color cycle. This lets the user know exactly what portion of the volume tree she is viewing by highlighting the data with animation.

Additionally, we provide animation to the parameters of the volume tree operators. Through animation, the user can explore the parameter space of her visualization. She can see how changing the various parameters affect the final result, without the manual change of parameters by herself. Figure 9 is an example where the operator is changed during the animation to show several different operation possibilities and the results.

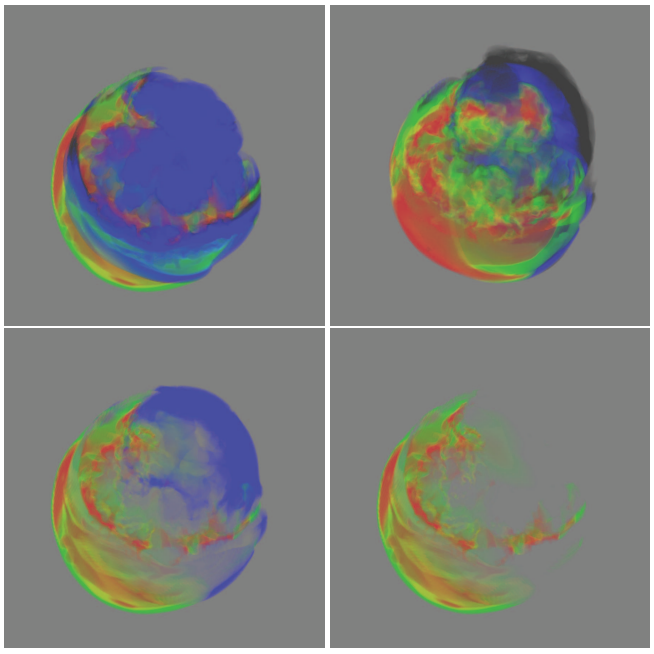

Fig. 9. The operation between two data fields is changed over time. From left to right, top to bottom, it shows B **atop** A, A **atop** B, A **xor** B, and B **out** A.

7 Volume Shader Implementation

Our method of multi-field visualization allows itself to be easily implemented in modern graphics hardware. By using 3D texture slicing to implement our volume renderer, the volume shader is translated into a fragment shading program using GLSL. When the source data are sliced using rendering quadrilaterals, the volume shader is automatically applied per point. The volume shader is a parallel data program that can be applied to the source data start to finish per fragment without any intermediate storage. For data or programs that cannot fit into graphics hardware, a CPU implementation can be written that applies the volume shader program per voxel position from the source data.

In our hardware implementation, the data fields are uploaded to the graphics hardware in the form of 3D textures. We interleave the fields in the RGBA channels, so that one data field is stored in a channel. That means up to four fields can be stored in one texture. One texture unit is used to store the transfer function and weighting functions in a packed 1D texture. Most graphics hardware have a minimum of four texture units, therefore, we can upload 12 (4 fields $*$ 3 textures) data fields simultaneously to the graphics hardware. On a Pentium 4 2.0GHz and an nVidia GeForce 6600 GT graphics card, we achieve over 10 fps without lighting turned on. When we turn on real-time normal calculation for lighting or NPR calculations, which need the gradient and use a 3D edge detector, the speed drops significantly. Times tend to be under 1 frame per second, ranging from 1 to 5 seconds per frame. Multiple texture lookups significantly slow down the fragment processing speed. Positional and color table animations can be done in real time on hardware, because the animation parameters are a small uniform or transfer function upload to the fragment shader.

We have provided many different examples of data sets that our technique can be used with throughout the paper. Figures 1 and 8 are examples of a time-varying computational fluid dynamic simulation of a vortex data set. These figures show how the data interacts in space over time. Figures 4, 5, 6, 7, and 10 are from a time-varying, multi-variate hurricane simulation. Our examples use the multi-variate fields of the data to locate interesting portions of the data set that exist between fields. Figure 11 is multi-variate, time-varying supernova simulation from the Tera-scale Supernova Initiative. We use three different fields and time steps to locate values in space over field over time. We also use a non-variate data set in Figure 9. The UNC brain data set is visualized in comparison with a distance field. The distance field is used to cull away data from the brain. While we do not have examples using comparative data sets, one can easily imagine using our technique to compare data from different simulation runs or measurements.

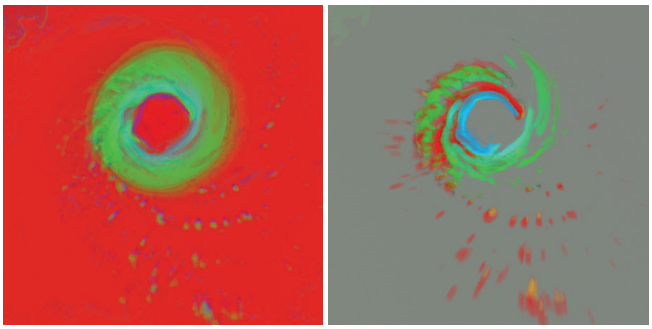

Fig. 10. Multi-field hurricane data set visualizing $(A$ **xor** $B)$**xor**$(C$ **xor** $D)$ on the left. $(B$ **in** $A)$**xor**$(D$ **in** $C)$ can be seen on the right. Each data field has its own transfer function and there is no color blending. Value and field can be determined from the color.

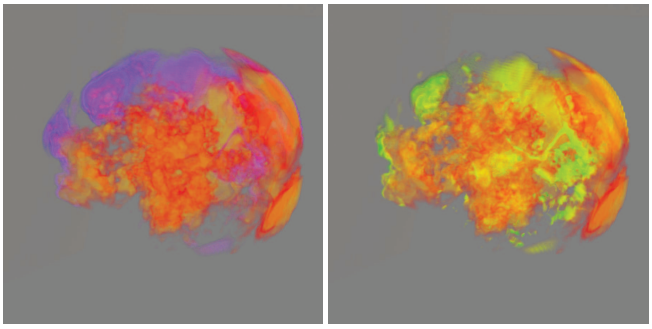

Fig. 11. Tera-scale Supernova Initiative data set visualizing C **atop**$(A$ **in** $B)$ on the left. C **atop**$(B$ **out** $A)$ can be seen on the right. Each data field has its own transfer function and there is no color blending. Value and field can be determined from the color.

8 Conclusion

We have presented a method for the comparison of different data fields. These data fields can be from time-varying, multi-field or comparative data sets. The method of comparison is through the expression of a volume shader that composes data fields together with operations. The operations can be either set or numerical, and they allow the user to easily express comparisons between her data that she wants to see. Our implementation of the set operations allows us to easily implement volume shaders of arbitrary complexity. With a distinction between appearance-based and value-based set operations, we can be more sensitive to color mapping and value preservation. By converting the volume shader to a volume tree, the user can see how her data combines together to form the final result. This is accomplished through the visualization of the volume tree in conjunction with a visualization spreadsheet. By observing how data are changed through the tree, the user can have a greater understanding of the data, and she can make more informed decisions on how to configure her visualization.

For future work, we would like to explore additional methods of feature extraction and presentation. There are some feature lines of interest that can be presented, such as intersection lines. Other animation techniques could be introduced to expose the context. There may be additional ways to render the context, such as finding operations

to apply in the contextual space. Additionally, we need to complete a user study to investigate how many levels of context the user can effectively visualize at one time, and what methods are best to show context in different scenarios. Finally, we may be able to analyze the data and generate automatic volume shaders that make best guesses to describe data which the user could be looking for.

ACKNOWLEDGEMENTS

This work was supported by NSF ITR Grant ACI-0325934, NSF RI Grant CNS-0403342, DOE Early Career Principal Investigator Award DE-FG02-03ER25572, NSF Career Award CCF-0346883, and Oak Ridge National Laboratory Contract 400045529. The TSI data set was provided by John M. Blondin (NCSU), Anthony Mezzacappa (ORNL), and Ross J. Toedte (ORNL). Thanks go to Kwan-Liu Ma for providing the the vortex data set made available through the NSF ITR project. Hurricane Isabel data were produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR and the U.S. National Science Foundation. Finally, the authors would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] C. Bajaj, C. Pascuci, G. Rabbiolo, and D. Schikore. Hypervolume Visualization: A Challenge in Simplicity. In *Proceedings of IEEE Symposium on Volume Visualization '98*, pages 95–102, 1998.

[2] S. Bruckner and M. E. Gröller. VolumeShop: An Interactive System for Direct Volume Illustration. In *Proceedings of IEEE Visualization '05*, pages 671–678, 2005.

[3] M. Burns, J. Klawe, S. Rusinkiewicz, A. Finkelstein, and D. DeCarlo. Line Drawings from Volume Data. In *Proceedings of ACM SIGGRAPH '05*, pages 512–518, 2005.

[4] W. Cai and G. Sakas. Data Intermixing and Multi-volume Rendering. In *Proceedings of EuroGraphics '99*, pages 359–368, 1999.

[5] M. Chen and J. V. Tucker. Constructive Volume Geometry. *Computer Graphics Forum*, 19(4):281–293, 2000.

[6] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive Contours for Conveying Shape. In *Proceedings of ACM SIGGRAPH '03*, pages 848–855, 2003.

[7] H. Doleisch, M. Gasser, and H. Hauser. Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data. In *Proceedings of the Symposium on Data Visualization '03*, pages 239–248, 2003.

[8] M. Fruehauf. Combining Volume Rendering with Line and Surface Rendering. In *Proceedings of EuroGraphics '91*, pages 21–32, 1991.

[9] P. Hanrahan and J. Lawson. A Language for Shading and Lighting Calculations. In *Proceedings of ACM SIGGRAPH '90*, pages 289–298, 1990.

[10] A. J. Hanson and R. A. Cross. Interactive Visualization Methods for Four Dimensions. In *Proceedings of IEEE Visualization '93*, pages 196–203, 1993.

[11] A. J. Hanson and P. A. Heng. Four-Dimensional Views of 3D Scalar Fields. In *Proceedings of IEEE Visualization '92*, pages 84–91, 1992.

[12] T. J. Jankun-Kelly and Ma K.-L. Visualization and Exploration via a Spreadsheet-Like Interface. *IEEE Transactions on Visualization & Computer Graphics*, 7(3):275–287, 2001.

[13] J. Kniss, G. Kindlmann, and C. D. Hansen. Multidimensional Transfer Functions for Interactive Volume Rendering. *IEEE Transactions on Visualization & Computer Graphics*, 8(3):270–285, 2002.

[14] W. Krueger. Volume Rendering and Data Feature Enhancement. In *Proceedings of IEEE Symposium on Volume Visualization '90*, pages 21–26, 1990.

[15] M. Levoy. Spreadsheets for Images. In *Proceedings of ACM SIGGRAPH '94*, pages 139–146, 1994.

[16] P. S. McCormick, J. Inman, J. P. Ahrens, C. D. Hansen, and G. Roth. Scout: A Hardware-Accelerated System for Quantitatively Driven Visualization and Analysis. In *Proceedings of IEEE Conference on Visualization '04*, pages 171–178, 2004.

[17] T. Porter and T. Duff. Compositing Digital Images. In *Proceedings of ACM SIGGRAPH '84*, pages 253–259, 1984.

[18] K. Proudfoot, W. R. Mark, S. Tzvetkov, and P. Hanrahan. A Real-Time Procedural Shading System for Programmable Graphics Hardware. In *Proceedings of ACM SIGGRAPH '01*, pages 159–170, 2001.

[19] P. Rheingans and D. Ebert. Volume Illustration: Nonphotorealistic Rendering of Volume Models. *IEEE Transactions on Visualization & Computer Graphics*, 7(3):253–264, 2001.

[20] J. J. van Wijk and R. van Liere. Hyperslice. In *Proceedings of IEEE Visualization '93*, pages 119–125, 1993.

[21] J. Woodring and H.-W. Shen. Chronovolumes: A Direct Rendering Technique for Visualization Time-Varying Data. In *Proceedings of International Workshop on Volume Graphics '03*, pages 27–34, 2003.

[22] J. Woodring, C. Wang, and H.-W. Shen. High Dimensional Direct Rendering of Time-Varying Volumetric Data. In *Proceedings of IEEE Conference on Visualization '03*, pages 417–424, 2003.