# Distribution-Driven Visualization of Volume Data

C. Ryan Johnson and Jian Huang

Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN

**Abstract**—Feature detection and display are the essential goals of the visualization process. Most visualization software achieves these goals by mapping properties of sampled intensity values and their derivatives to color and opacity. In this work, we propose to explicitly study the local frequency distribution of intensity values in broader neighborhoods centered around each voxel. We have found frequency distributions to contain meaningful and quantitative information that is relevant for many kinds of feature queries. Our approach allows users to enter predicate-based hypotheses about relational patterns in local distributions and render visualizations that show how neighborhoods match the predicates. Distributions are a familiar concept to non-expert users, and we have built a simple graphical user interface for forming and testing queries interactively. The query framework readily applies to arbitrary spatial datasets and supports queries on time variant and multifield data. Users can directly query for classes of features previously inaccessible in general feature detection tools. Using several well-known datasets, we show new quantitative features that enhance our understanding of familiar visualization results.

**Index Terms**—Volume visualization, volume rendering, multivariate data, features in volume data

## 1 Introduction

A primary research goal of volume visualization is to develop methods that, for a given problem, can discriminate between relevant and irrelevant data. By isolating features of interest, users can more quickly make important scientific and medical decisions. To this end, visualization and feature detection have become inseparably linked in the task of understanding volume data. For thorough investigation, feature detection must be customizable, interactive, and performed in terms of the problem domain.

The visualization field has made a continuing effort to expand the types of physical properties that are considered in describing features. As a result, more classes of features can be visualized and examined. So far, the set of properties used has grown to include intensity, shape, curvature, orientation, size, and so on [17, 12, 13, 23, 2]. This collection of primitives allows users access to a rich set of features relevant for many applications.

Several researchers have taken steps toward defining features based on statistical information derived at each voxel. For example, Lundström et al. [18] detect features by considering material frequency within a neighborhood and Jänicke et al. [10] use a measure of local statistical complexity to automatically find probabilistically unique features. These works, discussed in greater detail later, have shown the power of defining features according to statistical data.

In this work, we incorporate the arbitrary use of statistical data into feature querying. Our key concept is to query for features in terms of the most general form of statistics—the probability distribution function. In particular, we incorporate dynamic, user-driven, and statistically-based feature queries that enable interactive investigation of volume data.

Describing features in terms of distribution functions is much more complex than typical scalar-based methods. This complexity becomes acute when multiple variables, and hence multiple distribution functions, are studied together to understand their combined effects. To address this challenge, we have developed a compact, expressive query language called SeeDQ (for "See Distribution Query"). In SeeDQ, users filter distributions into relevant value intervals and compose a series of boolean predicates that relate statistical properties of these intervals. These predicate clauses define the features of interest. Each neighborhood in the volume has its statistical properties computed and is scored according to the degree to which the clauses are satisfied. The resulting query volume is rendered so that users can see how neighborhoods matched.

We avoid making a binary classification of matching and non-matching neighborhoods; instead, the degree of match is computed as a scalar-valued score. This score serves as a fuzzy metric which can be used as input to a traditional transfer function. A second input is the neighborhood's type of match, which indicates which predicate clauses in a compound query were matched and which were not. Currently, SeeDQ modulates opacity using a neighborhood's score and assigns color according to the combination of matched clauses.

Our method enables users to specify in succinct terms the statistical frequency patterns to visualize. Patterns in the distribution correlate with many practical features of interest, and by using the distribution as the basis for feature identification, we allow classes of features previously inaccessible to be visualized, offering a novel and quantifiable perspective of the data.

Some examples of feature queries that are best defined using distributions include finding regions in which two materials are mixed in a certain proportion, filtering neighborhoods that maintain some degree of homogeneity between timesteps, or discovering how multiple medical scans relate to each other. In all these cases, sample-specific properties do not provide sufficient information to adequately answer the query, while spatial properties are sensitive to orientation changes.

Distribution queries yield descriptive visualizations that can be used to test quantitative hypotheses, but they can also be used to enhance our understanding of familiar features. For example, a query for the presence of an isovalue in a neighborhood can be combined with a query on covariance in order to learn how variables interact around an isosurface. Alternatively, the output of a query can be used to augment existing feature detection mechanisms and create more powerful transfer functions and segmentation algorithms.

As a test of our distribution-based feature extraction method, we have been able to visualize previously inaccessible features that enhance our understandings of familiar results from several widely-used research datasets, including the 100-timestep Shockwave, 99-timestep Vortex, Visible Male head, and Chest datasets.

The remainder of the paper is organized as follows. Previous work related to our research is discussed in Section 2. In Section 3, we describe our concept of local distribution, and define the types of predicates that are supported. In Section 4, approaches to efficiently compute and render the resulting query volume are presented. Results and discussion follow in Section 5.

## 2 Related Work

Our method of multidimensional feature detection operates on the distribution of volume attributes in the neighborhood around a voxel.

• *E-mail: {cjohnson,huangj}@cs.utk.edu.*

*Manuscript received 31 March 2007; accepted 1 August 2007; posted online 2 November 2007.*

*For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.*

Feature queries are custom-tailored by the user using flexible boolean predicate-based criteria. This framework lends itself directly to comparing and visualizing multivariate data. Thus, the related work falls into four areas: (i) high-dimensional feature extraction, (ii) neighborhood-based visualization, (iii) query-driven visualization, (iv) multifield visualization.

### 2.1 High-dimensional Feature Extraction

In order for features to be detected and visualized, they must first be described in terms of the physical properties available. Density and gradient strength, for example, are often sufficient to characterize major boundary surfaces between bodily structures of living organisms [17]. Surfaces form a useful and perception-oriented class of features, but these may communicate artificial boundaries that do not necessarily exist in less-structured datasets [21].

Transfer functions have served as a primary means for interactively performing feature detection. The limits of one-dimensional transfer functions have long been known, and numerous multidimensional techniques have been studied. Tzeng et al. [28] describe a training-based transfer function design that uses scalar value, gradient magnitude, 6-neighbor values, and spatial location to discriminate between features and non-features. Kindlmann et al. [13] consider local surface curvature to classify samples. Many techniques examine derivatives and their zero-crossings [14, 9] to isolate surfaces and oriented structures.

These projects illustrate the importance of the local domain in feature extraction. However, surface- and gradient-based transfer function designs only indirectly consider a sample's neighboring voxels. In contrast, our approach makes consideration of the surrounding region explicit and adds support for multivariate volumes. In this way, we accommodate visualizing different classes of features that are not necessarily surface-based.

Lundström et al. [19] use sorted histograms to perform two-dimensional classification. In their work, a traditional 1-D histogram along a chosen dimension is displayed, but each bin is sorted and colored along the frequency axis according to a second dimension. Users can alter optical properties for selected regions on the histogram. Possible second dimensions that can be used for sorting include frequency and variance of a value range within each voxel's local neighborhood.

Kniss et al. [15] augment volume rendering by visualizing the statistical risk of choosing a particular surface given multiple competing segmentations. The actual process of feature definition is not a part of their work. Instead, they use a sample's vector of segmentation probabilities to alter the appearance of uncertain boundaries. Our system allows for a form of uncertainty visualization using approximate matching to feature classes, but we do not retain specific information on samples' partial membership in each feature class.

Isosurfaces form another class of features, and the task of finding interesting isosurfaces can benefit from higher-dimensional information. For example, spectra of surface characteristics may be computed and displayed, guiding users to scalar values that yield an isosurface having a certain volume, surface area, or curvature [2, 22]. Tenginakai et al. [26] present a similar idea but use neighborhoods' statistical moments to determine salient isovalues. We support the use of statistical moments like variance and skewness in our distribution query to allow for more general feature detection.

### 2.2 Neighborhood-based Visualization

Using neighborhood data directly to augment input to a function has been discussed in several works. Laidlaw et al. [16] determine the mixtures of materials within a single voxel by sampling the voxel numerous times, forming a histogram of the results, and probabilistically choosing which materials best comprise the distribution. Their work aims at solving the partial volume effect and is dependent on an understanding of which materials are contained in the volume.

Image processing techniques that rely on neighborhood data have been employed for many applications. Fang et al. [6] integrate convolution and other neighborhood techniques in the rendering pipeline to sharpen, filter, and perform edge detection in volume data.

Lundström et al. [18] define the term *range weight* as the frequency at which a material occurs within a neighborhood. They use range weights for two applications: (a) decomposing the global histogram into its unknown constituent materials by using the neighborhoods' range weights to determine what intensity ranges combine in smaller regions and (b) competitive classification, in which a neighborhood's range weights for several known tissue types are combined with domain knowledge to determine which tissue type occurs at a sample. Our work is similar to this second application in our leveraging of statistics of local value intervals to perform feature detection. A primary difference is that Lundström et al. arbitrate between features by locating range weights within static confidence intervals derived from domain knowledge and combining the resulting confidence levels. In contrast, we provide a more general framework in which users define features by comparing range weight and other statistical primitives to expressions of arbitrary complexity. These expressions need not be static—they may include other statistics of the neighborhood.

### 2.3 Query-driven Visualization

The process of designing a transfer function can be likened to querying a database. A subtle difference between the two is that many lookup table-based transfer function designs ask the user to assign optical properties to the whole dataspace, while query-based approaches allow the user to directly describe the values of returned records. Our distribution-based feature detection design fits a more query-based model.

An example of a query-based visualization application is VisDB [11], which displays the results of queries on large multivariate databases according to how relevant data items are to a query. The motivation behind this tool is that traditional queries are dependent on precisely known keys, an assumption that precludes a more general, approximate search. Users do not always know what they are looking for, and a less strict querying process is needed with strong visual feedback to indicate how close records are to fully meeting query predicates. VisDB is not applied to volume rendering directly, but is used to visualize the relevance of the results returned from querying of an arbitrary database.

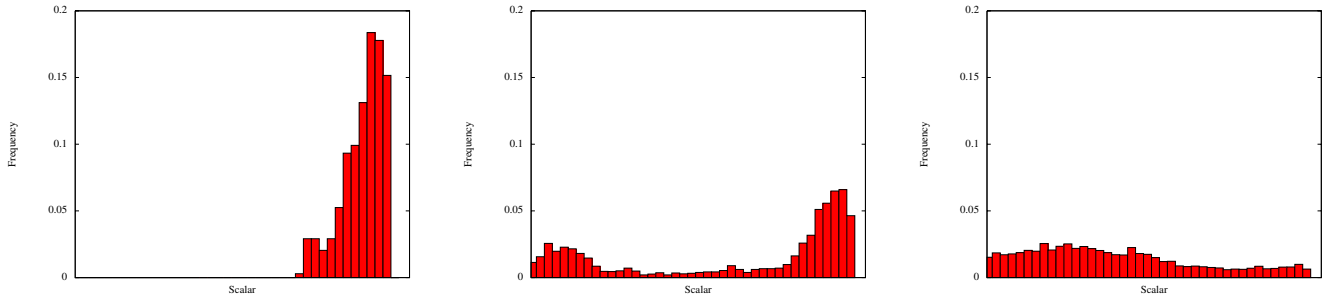
Doleisch et al. [5] describe a feature querying framework that lets users make multidimensional range selections in a brushing interface. Multiple range selections can be combined in arbitrary ways using logical operations. Our method is similar, but we operate within the local neighborhood and add the ability for users to specify probabilistic criteria on the value ranges.

The Scout project [20] supports a form of scripted querying directly on the GPU in its shader framework. Users can form boolean predicates to filter out value intervals for further processing.

### 2.4 Multifield Visualization

Many transfer functions are multidimensional in the sense that they take scalar value plus derived attributes like gradient magnitude as input. But visualizing data that is already multidimensional in the acquisition stage poses a significant roadblock to clear understanding. Cai and Sakas [3] describe a taxonomy of visualization methods for intermixing multivariate or multiple volumes. Most techniques of displaying multivariate data involve combining noticeably different textures, color, and glyphs into a single visualization. Stompel et al. [25] use non-photorealistic techniques to help users decipher multivariate computational fluid dynamics and time-varying data. They note that most scientific visualization is by definition non-photorealistic in that variables like pressure, electron density, and temperature offer no photographic model.

Woodring and Shen [29] introduce a method to composite and visualize multiple volumes. To facilitate understanding of the resulting visualization, each intermediate composition is visualized in a volume tree. By navigating this tree, the users receive feedback at each step in the composition and can quickly locate errors. Examining suboperations in an expression is not unlike assessing individual predicates of a compound query, and we employ similar techniques to indicate which predicates match in our distribution-based query.




(a) Neighborhood radius = 5 (b) Neighborhood radius = 10 (c) Neighborhood radius = 20

Fig. 1. Example distributions for three differently-sized, spherical, and concentric neighborhoods centered on a voxel in timestep 99 of the Jet dataset. In (a) one material dominates a small spatial region, 5 voxels in radius. In the slightly larger neighborhood shown in (b), a bimodal distribution reveals two prevalent materials. In (c), the distribution loses local detail and closely matches the more uniform global histogram.

Jänicke et al. [10] use a measure called local statistical complexity to reduce multifield data down to a complexity field in which regions are marked according to their degree of information. The complexity field is then visualized. Their method of feature detection is automatic, with features defined according to the dataset but in an application-independent manner. Drawing from a similar motivation, we support detection of features defined on multiple variables. Our measure of feature importance, however, is dynamically computed and based on users' distribution-based feature queries.

Correlation fields have been demonstrated as another means of exploring multifield volumes. Sauber et al. [24] condense a field's variables at each location into a measure of similarity. All combinations of variables are examined hierarchically using a graph structure. Gosink et al. [7] project the correlation field between two variables on an isosurface, possibly defined on a third variable. Potentially interesting isosurfaces are detected using distribution functions.

## 3 Distribution-based Feature Detection with SeeDQ

SeeDQ is a query-based mechanism for extracting knowledge directly from local frequency distribution data. Designed as an expressive query language, SeeDQ is intended for exploring previously unknown relationships in the data, particularly those that require unified studies of multiple variables. To this end, SeeDQ expands the current vocabulary available for quantifying features in visualization. It can also be used for studying the statistical behavior of many familiar kinds of features, such as isosurfaces.

In SeeDQ, the local domain is defined as the neighborhood centered around each individual voxel. Users select intervals in attribute space to filter out value ranges of interest in each neighborhood. For example, a user may be curious how intervals of high density, moderate temperature, and low pressure compare. Neighborhoods of interest are then described using a predicate-based query that describes features in terms of statistical metrics of the selected intervals. The query is evaluated by computing the statistical metrics for all neighborhoods and determining how closely they meet the target metrics. Matching neighborhoods are rendered according to their degree of match and combination of matched clauses.

### 3.1 Local Distributions

The key structure of SeeDQ queries is the local distribution of intensity values. We consider the distribution of each voxel's local neighborhood and examine its statistics.

#### 3.1.1 Neighborhoods

We define a neighborhood $H$ around voxel $v$ as a set of voxels $v_i$ surrounding $v$. $H$ serves as a sample space, and each $v_i$ is an elementary event in space $H$.

The simplest and default neighborhood is the set of all voxels within a Euclidean distance $d$ of $v$. Some applications, however, may benefit from a sample space tailored to the features of interest. In SeeDQ, we allow users to define custom sample spaces as a collection of 3-D coordinates relative to the origin. These coordinate masks are flattened into a 1-dimensional array of voxel address offsets the neighborhood's central voxel. This design allows flexible handling of spaces of all radii, shapes, distance functions, and dimensionalities.

The shape and distance function used for a sample space are important application-dependent considerations. For example, spherical neighborhoods align well with normally distributed phenomena, while an ellipsoidal or even curved neighborhood may offer a more suitable sample space for features oriented in a known direction. When voxels are anisotropic, the neighborhood sizes may be adjusted according to the disparity among the differently-spaced axes. The mask used for a query can be selected at runtime in SeeDQ.

Every voxel serves as the origin of one neighborhood, and in constrast to the work of Lundström et al. [18], we allow neighborhoods to overlap. We do not fit neighborhoods to voxels at volume edges, however, where a full neighborhood cannot be formed. Partial neighborhoods contain smaller numbers of voxels. These reduced sample spaces are not easily compared to the full, internal neighborhoods and are omitted from the query.

#### 3.1.2 Frequency Distribution

After a neighborhood structure is defined, we examine every neighborhoods' frequency distribution of values. When restricted to small neighborhoods (e.g., a radius of 1) such as those used to compute gradient with central differencing, distributions may not contain significant information. Alternatively, overly large neighborhoods may mask the regional cues that denote features. Accordingly, neighborhoods should be sized so that their distributions contain enough voxels to adequately evaluate the query. Figure 1 illustrates how distributions change with size for several concentric neighborhoods.

Scalar metrics derived from the local distribution have been shown to discriminate between features [18, 16]. Herein, we allow the user to query directly on the distribution function, allowing for a more diverse set of features to be studied. The distribution offers the user a rich description of the underlying data for finding features. For example, in the distribution of a medical dataset, the frequency relationship between soft tissue and bone is immediately perceived and the relationship can be described quantitatively. Alternatively, neighborhoods that are homogeneous have very concentrated unimodal distributions, while heterogeneous neighborhoods have modeless distributions with high variance. A gap in a distribution indicates a hard boundary between materials, while a skewed distribution tells us that a value range is asymmetric about its mean. Many physical phenomena can be described using distribution primitives, and relationships between such phenomena can be described quantitatively using these primitives. This fact forms the foundation of our feature detection scheme.

Note that histograms, which are discretized distributions, have long played a role in 1-dimensional and 2-dimensional transfer function de-

signs [12, 14]. Our use of distributions is different, however, in that our system uses distributions based on neighborhoods rather than the entire volume, and instead of using the global histogram to influence the transfer function, users query for features directly on the local distributions. Also, as explained in Section 3.2, we do not convert local frequency distributions into conventional discretized histograms, but operate on the original data values to query a neighborhood distribution.

### 3.2 Predicate-Based Querying

There are two parts to a distribution query in SeeDQ: bins and predicate clauses. In specifying bins, users establish intervals of interest that are used to filter the scalar values' distributions for each neighborhood. A query is then composed by specifying a series of boolean predicates that statistically relate features of the neighborhood's distribution intervals. For example, a query may be formulated by specifying that neighborhoods contain twice as many high density values (e.g., bone) as low density values (e.g., soft tissue), or that pressure in one timestep should be comparable in mean value to pressure in another timestep.

#### 3.2.1 Bin Specification

First, the user decides what value intervals in the volume are of interest for querying neighborhoods. The user restricts the domain of the distributions to intervals that are relevant to the features of interest and discards the rest. All standard inequalities are supported in defining intervals, and intervals for different bins may overlap.

To examine and compare multivariate datasets, the user specifies the variable over which a bin interval spans. For each interval within each neighborhood, we compute statistics like relative frequency, mean, variance, standard deviation, and skewness. We also support joint intervals that link two variables for each location in order to query covariance.

The bin is filled by traversing a neighborhood $H$ and "dropping" each voxel in if its value falls within the interval. The result is a set of values, as described in Equation 1. As the neighborhood's voxels are processed, we calculate the bin's statistics in an online fashion. For example, relative frequency, mean, and variance for an arbitrary $bin_i$ in neighborhood $H$ are calculated in the conventional way in Equations 2-4, with the other statistics for standard deviation, skewness, and covariance computed according to their definitions.

$$bin_i = \{v : v \in H \text{and } min_i \leq v \leq max_i\} \quad (1)$$

$$freq(bin_i) = \frac{|bin_i|}{|H|} \quad (2)$$

$$mean(bin_i) = \frac{1}{|bin_i|} \times \sum_{v \in bin_i} v \quad (3)$$

$$var(bin_i) = \frac{1}{|bin_i|} \times \sum_{v \in bin_i} (v - mean(bin_i))^2 \quad (4)$$

The bin descriptions are formulated by the user using a tool described in Section 4.1, which produces an XML-based query file. We show a clearer and equivalent textual representation of several bin descriptions in Figure 2(a).

#### 3.2.2 Clause Specification

Once the intervals of interest are specified, the user specifies a compound query about how the intervals should relate. This is done through a series of boolean predicates that compare bin variables to other quantities. An example query specification is shown in Figure 2(b).

Together with the bins specified in Figure 2(a), we use Figure 2(b) to illustrate how a query is evaluated. The simple predicate `freq(bin 0) > 50%` matches neighborhoods where a majority of the voxels fall in the interval specified by bin 0. To require that bin 0 not contain any voxels, the predicate becomes: `freq(bin 0) == 0`.

```
bin 0 {
   attribute 0
   range 0.9 to 1.0
}
bin 1 {
   attribute 1
   range < 0.5
}
bin 2 {
   attribute 4
   range >= 0.9
}
```

(a) Bins

```
clause 0 {
   freq(bin 0) > 50%
   weight 50%
   dropoff 0.1
}
clause 1 {
   var(bin 2) < var(bin 1)
   weight 25%
   dropoff 10
}
clause 2 {
   freq(bin 2) > 2 * freq(bin 1)
   weight 25%
   dropoff 10
}
```

(b) Clauses

Fig. 2. An example query comprised of (a) bins and (b) clauses. Here, the user is interested in three value intervals: $[0.9, 1.0]$ for variable 0, $[0, 0.5)$ for variable 1, and $[0.9, 1]$ for variable 4. The compound query specification in (b) contains three predicates. Weight values like 50% and 25% indicate the importance of each predicate to the overall query. The dropoff values of 0.1 and 10 are parameters used to allow approximate matches as described in Section 3.3.

In addition to relating bin variables to constant values, users can refer to other bin variables. For example, the predicate `var(bin 2) < var(bin 1)` finds neighborhoods that have a more homogeneous subset of voxels in bin 2's interval than in bin 1's interval by comparing variances. The bin statistics variance and standard deviation are useful when neighborhoods contain a significant number of voxels and the dispersion of values is relevant to features of interest. For instance, variance is high in neighborhoods containing boundaries.

Arithmetic expressions are supported in each clause to allow more complex relationships between variables. A predicate of `freq(bin 2) > 2 * freq(bin 1)` finds neighborhoods that have over twice as many voxels in bin 2's interval than in bin 1's interval.

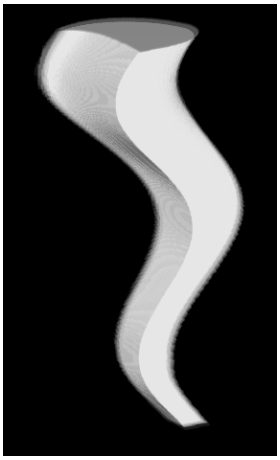
The right-hand side of the inequality can be any standard algebraic expression operating on constants and bin variables. It represents the target value of the clause, and the inequality operator indicates how the bin statistic on the left-hand side must relate for the neighborhood to match. If the inequality is satisfied, the neighborhood matches the predicate.

### 3.3 Approximate Matching

Displaying more neighborhoods than just those that strictly match every predicate often leads to more insight into the dataset. In this way, users receives feedback even for queries that return few neighborhoods. Additionally, users may be interested in neighborhoods that match certain clauses or match the entire query to a certain degree. A binary cutoff between matching and nonmatching neighborhoods does not allow users to specify vaguely determined queries, and the resulting visualization may contain distractingly sharp edges. To facilitate more flexible queries and improve feedback, we support two methods of displaying approximately matching neighborhoods: fuzzy scoring and predicate-signature classification.

#### 3.3.1 Fuzzy Scoring

To illustrate our method of fuzzy scoring, consider the Tornado dataset [4] shown in Figure 3. Here, the user is searching for the intersection where the velocity magnitudes along all three axes are positive and in high concentration. This query can easily be formulated in terms of the frequency distribution. The rendering on the left shows the exact location of the intersection, but by allowing fuzzy matches to be shown with less opacity, the user also sees regions where the query is nearly met in the right image. In this case, the three adjacent, differently-colored areas are where only one or two of the velocity components are in high concentration.



(a) Non-fuzzy matching

(b) Fuzzy matching

```
bin 0 {
   attribute 0
   range > 0.01
}
bin 1 {
   attribute 1
   range > 0.01
}
bin 2 {
   attribute 2
   range > 0.01
}
```

(c) Bins

```
clause 0 {
   count(bin 0) >= 50%
   weight 33%
   dropoff 0.37
}
clause 1 {
   count(bin 1) >= 50%
   weight 33%
   dropoff 0.37
}
clause 2 {
   count(bin 2) >= 70%
   weight 33%
   dropoff 0.37
}
```

(d) Query specification

Fig. 3. A single query on the Tornado dataset, with fuzzy matches rendered in (b) but not in (a). The opaque white intersection is comprised of neighborhoods where all three velocity components are positive and in high concentration. The three translucent regions in (b) consist of neighborhoods where only $x$- and $z$-velocities match, then $y$ and $z$, and finally just $z$.

To support fuzzy matching, we evaluate each predicate's inequality so that a value is returned indicating how far a neighborhood is from satisfying the inequality. This value is the predicate's *distance*, or $\Delta$. When the inequality is completely satisfied, the distance is 0. Otherwise, we compute the distance as the absolute value of the difference between the left- and right-hand sides of the predicate's inequality.

For instance, using the first predicate from Figure 2(b), if `freq(bin 0)` does comprise over 50% of the neighborhood, it is assigned a distance $\Delta$ of 0. If only 40% of the neighborhood is in bin 0, then $\Delta$ is $|40\% - 50\%| = 0.1$.

The user chooses how loosely a query can be matched by specifying a dropoff parameter for each clause, as seen in Figure 2(b). The value of the parameter indicates the distance which should be assigned a score of 0.5. The dropoff parameter and $\Delta$ are then fed into an exponential decay function shown in Equation 6 to obtain the clause's matching score.

$$D = \frac{dropoff_i^2}{\ln(.5)} \tag{5}$$

$$score_i = \exp(\frac{\Delta^2}{D}) \tag{6}$$

We use the scoring function in Equation 6 rather than a linear ramp because it offers better control over scoring various degrees of non-matching. Additionally, this function is defined for any possible $\Delta$, which is in $[0,\infty)$, and the function has a range of $[0,1]$. This range is useful since the score values can then map directly to color or opacity during volume rendering. Larger dropoff parameters tolerate more loosely matching neighborhoods. Figure 4 shows how scores are computed for various dropoff values.

Both $\Delta$ and $dropoff_i$ are values in the same units as those returned by the bin statistic on the left-hand side of each predicate. If the predicate's statistic is `freq`, then the units represent a percentage of the neighborhood. If the statistic is `mean` or `stdev`, then the units are defined by the bin's attribute, and `var` is in square units. Normalization to a standard domain is avoided since distance is defined on an open-ended interval.

As in [11], we recognize that different predicates in the query may be of differing importance to the user. Therefore, in addition to a dropoff parameter, each clause is assigned a weight indicating its importance to the overall query, as shown in the Figure 2(b). This weight is used to directly modulate the clause's score and can be set to any positive or negative value. Negative weights serve as the negation operator in our matching scheme, used to penalize neighborhoods that match a clause. A weight of 0 is also legal; this indicates that no partial scoring is permitted and that the neighborhood must perfectly satisfy the clause. In effect, a 0-weight restricts the query to regions where the specified condition is met, i.e., the other clauses become statements of conditional probability.

The score of the entire query is computed as the weighted sum of the scores of all clauses, as shown in Equation 7:

$$score = \text{clamp}(0, 1, \sum_{i \in Q} weight_i \times score_i) \tag{7}$$

Here, $Q$ is the set of clauses forming the compound query. $score_i$ is the fuzzy matching score for clause $i$, defined by Equation 6, and $weight_i$ is the weight for clause $i$.

The weighted sum *score* is clamped between 0 and 1 to form a neighborhood's final score. The sum must be clamped because weights do not necessarily sum to unity, which allows users to execute different kinds of queries. For instance, a user may assign two predicates each a weight of 100% to look for neighborhoods that meet either predicate in a disjunctive relationship. If two predicates are each given a 50% weight, however, then both must be met in order for the final score to reach 1. This is a conjunctive query.

An alternative method of computing the neighborhood's final score is to avoid clamping and normalize the scores according to the sum of weights. However, this approach limits the types of clauses that can be expressed. The weights would have to sum to a non-zero value, i.e., two clauses with weights 1 and $-1$ would not be possible. Additionally, disjunctive clauses would not be supported, as a neighborhood matching one clause but not another would not receive a full score.

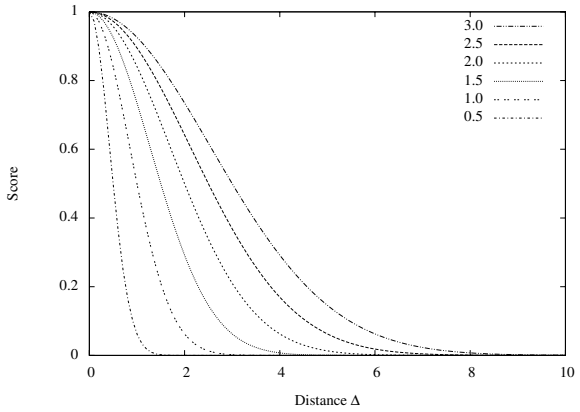
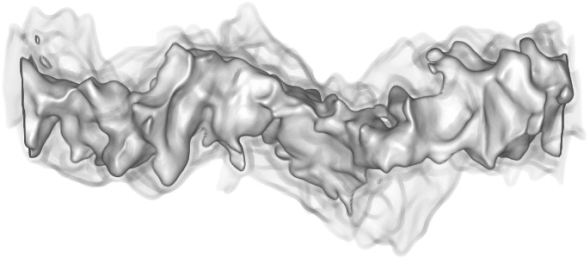
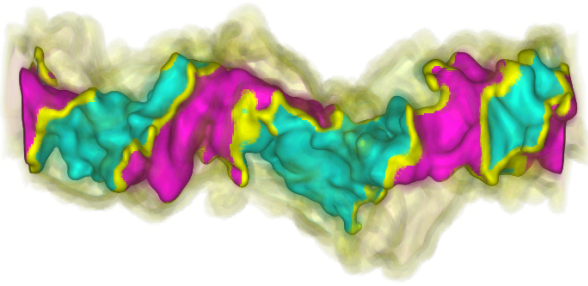



Fig. 4. Several degree-of-match functions with differing dropoff parameters. The dropoff parameter indicates which distance is assigned a score of 0.5. Thus, a low dropoff will tolerate only closely matching neighborhoods, while a high dropoff offers a more approximate match.



(a) Score Volume



(b) Query Volume

```
bin 0 {
   attribute 0
   range > 200
}
bin 1 {
   attribute 1
   range > 200
}
```

(c) Bins

```
clause 0 {
   freq(bin 0) >= 80%
   weight 50%
   dropoff 0.83
}
clause 1 {
   freq(bin 1) >= 80%
   weight 50%
   dropoff 0.83
}
```

(d) Clauses

Fig. 5. A query on a volume containing timesteps 80 and 99 of the Jet dataset. The grayscale score volume in (a) is opaque where the query is fully met, i.e., both timesteps have 80% of their values in the correct interval. Elsewhere, scores dropoff to 0. In (b), the entire query volume is additionally colored according to its predicate signatures to indicate which clauses were matched.

Obviously, combinations of weights should be tailored according to application needs. The final neighborhood score is saved with each voxel and can be used to modulate a sample's opacity in volume rendering. An example score volume can be seen in Figure 5(a).

#### 3.3.2 Predicate-signature Classification

The neighborhood score gives a succinct measure of how closely a neighborhood matches the overall query, but it does not contain enough information to determine how a compound query was matched. To allow for more in-depth feedback, we implement a system of *predicate signatures*. This technique is inspired by VisDB [11], in which a user can inversely determine which predicates of a compound query were matched by simply examining the results' color values.

The predicate signature of a neighborhood is a represented compactly by a bitfield for each voxel, with each bit corresponding to exactly one clause. Each clause whose inequality is fully satisfied (i.e., $\Delta$ is 0) has its bit turned on, and all remaining bits are turned off. When interpreted as an integer, the bitfield has a value between 0 and $2^{|Q|}-1$, where $Q$ is the set of clause predicates.

For typical queries, composed of 8 clauses or fewer, the bitfield fits comfortably in a single byte per voxel. Our approach readily accommodates larger bitfields, but the predicate signature is less useful when the number of predicates is high due to limits of human perception.

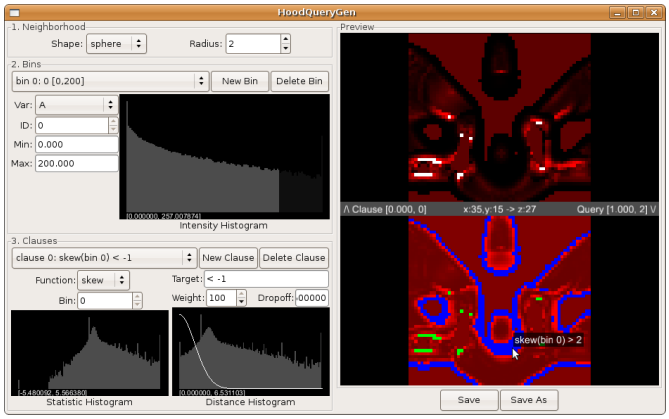



Fig. 6. A screenshot of our tool for quickly generating neighborhood queries. Controls for setting neighborhood shape, creating bins, and creating clauses appear in the left pane. A single slice of the queried volume is rendered in the right pane. The slice is scored and rendered according to both the currently selected clause (top) and the entire query (bottom) for maximum clarity of how the query is being matched. Color indicates which combination of clauses matched and intensity indicates score. The color mapping is described further in Section 4.3. The query shown focuses on the interval $[0,200]$ of the Neghip dataset and contains two clauses, one highlighting neighborhoods of positive skewness (blue) and the other negative (green).

## 4 Query Volume Rendering

Users generate queries using an interactive graphical interface designed to give maximal feedback on changing query parameters. After the query has been adjusted to the user's liking, a query volume is generated and rendered using a GPU volume renderer.

### 4.1 Generating a Query

The ability to formulate arbitrary queries on statistical information means users can investigate any number of probabilistic hypotheses on their data. However, users often need guidance in selecting and iteratively refining query parameters, and, if neighborhood sizes are large, actually performing the query is quite computationally expensive. To guide users in setting parameters and to facilitate quick feedback, we have implemented an easy-to-use graphical tool for formulating queries and previewing their results.

Figure 6 contains a screenshot of our query formulation tool. The left pane contains GUI controls and histograms related to the query parameters. Users formulate a query by: 1) selecting neighborhood shape and size, 2) creating bins to filter out scalar ranges, and 3) creating clauses that relate the bins' statistical metrics. The clause histograms show for all neighborhoods of the volume the distribution of the selected clause's statistical value and the distances between the actual neighborhood values and the inequality's target value. In the query depicted in the figure, distributions of the skewness of intensity range $[0,200]$ and the distances from the target value of $-1$ are shown. Users can see how the score drops off with distance in the distance histogram.

The right pane renders a two-dimensional slice of the query volume. Both the currently selected clause and the total query are rendered. Users hover over a voxel with the mouse to see which queries it's neighborhood satisfies and how it is scored. They can interactively step through slices along any object-aligned axis and adjust query parameters for immediate feedback. When finished editing, users save the query to an XML file, which serves as input to a separate tool that queries the entire 3-D volume.

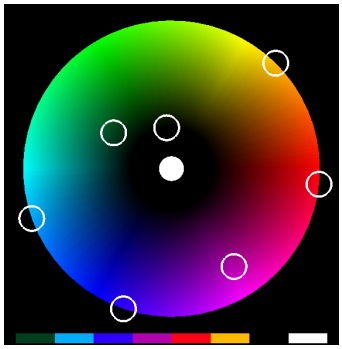
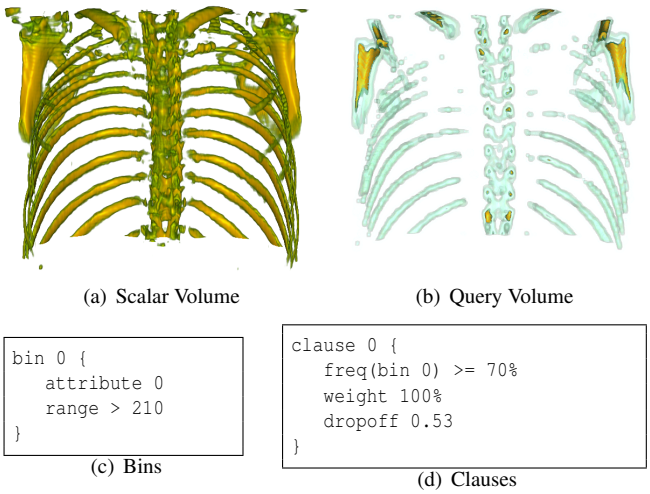


Fig. 7. The color lookup table interface. Each possible combination of matched clauses is represented by a predicate signature, with the white circles centered on the colors associated with the signatures. A circle's distance from center indicates the opacity used for its predicate signature. The resulting colormap texture is displayed at the bottom.




(a) Scalar Volume (b) Query Volume

(c) Bins (d) Clauses

Fig. 8. A query for large bone structures in the Chest dataset. (a) A 1-D transfer function can only distinguish between bone and not bone. (b) Our distribution-based query finds large bones in the spine and shoulder. Smaller bones have lower scores and are shown with limited opacity. A spherical neighborhood of radius 3 was used as the sample space.

### 4.2 Generating Query Volume

To visualize a 3-D query volume, our search tool first loads the XML query file generated by the query formulation tool. The clause inequalities are lexically analyzed and parsed using structured grammars and code generated by the tools `flex` and `bison`. Since the inequalities contain neighborhood-specific terms that can only be evaluated after neighborhoods' bins have been filled, each boolean predicate is compiled into a postfix expression that can be evaluated quickly once its terms are known.

Each neighborhood can be processed independently of all other neighborhoods, so we have implemented our querying algorithm in parallel using MPICH. For a query on a cluster using $n$ nodes, the input volume is first sliced into $n$ subvolumes of approximately equal size. Slicing is done along the major axis for fast decomposition. To allow nodes to access all voxels in a neighborhood, some overlap of slices between nodes is necessary. Each node in the cluster processes all fully-contained neighborhoods in its subvolume, and the resulting scores and predicate fields are gathered back to the root node. This query volume is saved to disk for rendering.

### 4.3 Transfer Function

With the scores and signatures computed for each neighborhood, we render the query volume using these values to assign color and opacity. A sample's color is determined by its predicate signature, and its shading and opacity by the scores.

Essentially, the predicate signatures form a segmented or tagged volume. The signatures are scaled and biased so that they index into the center of a tag's cell in a colormap texture. An example tag colormap is shown in Figure 7. Effectively, this colormap translates each possible combination of matching clauses to a different color value, allowing the user to interpret how regions matched the overall query.

In the example colormap, there are three clauses in the overall query, with $2^3$ possible predicate signatures. The center white circle represents the maximal predicate signature, where all clauses are fully matched. The other seven circles center on colors that are made sufficiently distinguishable by sampling the outer ring of the HSV hexcone at equidistant intervals. The resulting colormap is displayed at the bottom of the interface. To change colors, the user can shift or "dial" the equidistant color set around the hexcone, or set colors manually.

In the example in Figure 7, the colormap contains a color for every possible bitfield value. In reality, not all colors may be used since only a few combinations may be present in a volume.

### 4.4 Lighting Calculation

Using this approach, multifield volumes or multiple volumes are visualized by means of a query volume, in which scores and predicate signatures are used for the final rendering. In the spectrum of multifield visualization methods [3], our approach falls under the category of data-level intermixing.

A key issue of visualizing datasets of multiple variables is how to define the gradient at each sample to compute local illumination. Gradients are unlikely to correspond across variables. Furthermore, the composite volume has surface characteristics independent of the original variables, and it's unlikely that any combination of gradients could be used without introducing misleading surface cues.

We believe our query-based scoring approach leads to a convenient solution for determining the gradient. At each sample, we compute the central difference across neighboring scores. Scores gradate coherently from feature to feature, leading to well-defined and coherent surfaces around the matching neighborhoods. Since scores represent a weighted blend of the clauses of the user's query, we make gradients a meaningful user-driven attribute for volume rendering.

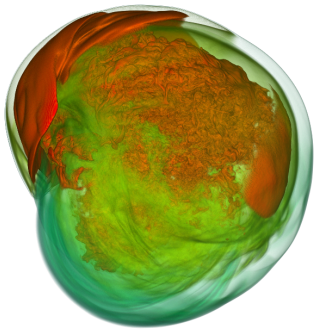
### 4.5 Visualizing Queries

We render the query volume using a GPU-accelerated volume renderer. View-aligned slices are clipped against the volume's bounding box and rendered, invoking a fragment program to determine each fragment's color using the scores and predicate signatures to index into the color and opacity textures.

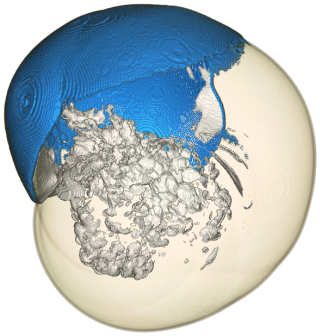
Gradients are computed directly on the graphics card using the neighboring scores. Six additional texture lookups are required to compute the central difference at a sample, but the extra cost incurred by these lookups is minimal given the limited number of texture channels available.

Tagged volume rendering poses several difficulties that we must address. Shading without the original data is overcome by using the scores in the query volume. By defining the gradient across scores instead of predicate signatures, smoother shading is achieved. Additionally, we must avoid linearly interpolating predicate signatures, since the interpolation may introduce incorrect intermediate values between segment boundaries. One solution [27] is to perform segment membership tests by performing interpolation for only one segment at a time. This has been implemented on the GPU [8] using a multipass scheme, with interpolation necessarily done without the aid of hardware acceleration.

Our solution to interpolating predicate signatures without sacrificing hardware-accelerated filtering requires loading the predicate signature (tag ID) volume twice, once with a nearest neighbor filter and once with a linear filter. To generate smooth color boundaries, we de-



(a) Scalar Volume



(b) Query

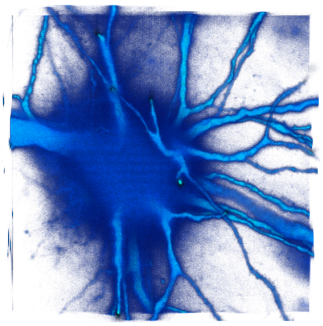
```
bin 0 {
   attribute 0
   range < 30
}
bin 1 {
   attribute 0
   range > 50
}
```

(c) Bins

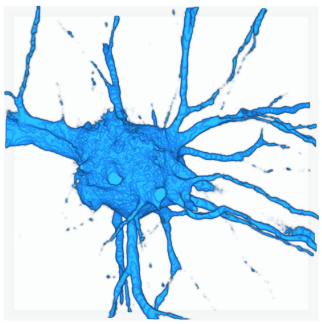
```
clause 0 {
   freq(bin 0) == 0%
   weight 10%
   dropoff 1.18
}
clause 1 {
   var(bin 1) > 1000
   weight 90%
   dropoff 5.27
}
```

(d) Clauses

Fig. 9. (a) Entropy field for a timestep of the TSI supernova simulation, rendered using a 1-D transfer function. (b) A query on the same timestep for heterogeneous regions with and without zero-valued background voxels. Blue and white regions have high variance, transparent red has a somewhat high variance, and blue and transparent red contain some amount of background.



(a) Scalar Volume



(b) Query Volume

```
bin 0 {
   attribute 0
   range 0 to 39
}
bin 1 {
   attribute 0
   range 40 to 255
}
```

(c) Bins

```
clause 0 {
   freq(bin 1) > freq(bin 0)
   weight 100%
   dropoff 0.37
}
```

(d) Clauses

Fig. 10. (a) Traditional volume rendering of a noisy nerve dataset. (b) A distribution query for neighborhoods containing more non-zero voxels than background. SeeDQ allows users to perform custom filtering using neighborhood distribution criteria.

termine the predicate signature for each sample at subvoxel precision by examining three predicate signature values: the nearest neighbor $t_n$, the linearly interpolated value $t_l$, and the nearest neighbor $t_g$ at the next voxel along the gradient. We select the sample's predicate signature $t$ as the closer of $t_n$ and $t_g$ to the sample's interpolated $t_l$, as in Equation 8.

$$t = \begin{cases} t_n & \text{if } |t_l - t_n| < |t_l - t_g| \\ t_g & \text{otherwise} \end{cases} \tag{8}$$

Essentially, the linearly interpolated tag is used to determine whether the sample is closer tag-wise to the adjacent segment than the nearest segment. In homogeneous regions, $t = t_l = t_g = t_n$, so the tag is constant. We've found this method to greatly reduce the harsh edges of nearest neighbor interpolation without adding much complexity.

The interpolated predicate signature is used to index into a colormap like the one shown in Figure 7. Each predicate signature is automatically assigned a distinct color, which can be modified interactively. Gradients and opacity are defined from the score. Users can further modulate opacity based on the predicate signature, allowing for entire segments to be toggled on and off. Given the color, opacity, and gradient, the fragment's output color is computed.

Typically, the blending of transparent surfaces creates colors that map ambiguously back to their data sources. In practice, however, we have found that few enough clauses are used in SeeDQ queries that the predicate signature colors are sufficiently low in number to prevent any ambiguity.

## 5 Results and Discussion

We tested our method using a variety of medical and scientific datasets. The datasets used for the example queries are described in Table 1. For all queries shown in this work (excluding the temporarily-defined neighborhood in Section 5.4, we use spherical neighborhoods with a radius of 2 or 3 voxels.

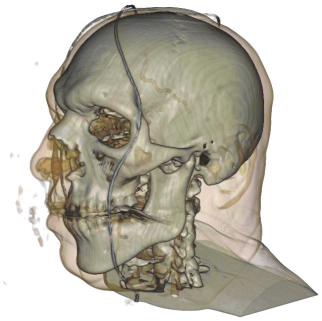
| Dataset | Modality | Resolution | | Time |
|---|---|---|---|---|
| | | Variables | Spatial | |
| TSI | Simulation | 5 | $432 \times 432 \times 432$ | 300 |
| Jet | Simulation | 1 | $256 \times 256 \times 256$ | 100 |
| Tornado | Simulation | 3 | $256 \times 256 \times 256$ | 1 |
| Head | CT | 1 | $128 \times 256 \times 256$ | 1 |
| Nerve | Confocal Microscopy | 1 | $512 \times 512 \times 76$ | 1 |
| Combustion | Simulation | 5 | $480 \times 720 \times 120$ | 122 |
| Chest | CT | 1 | $384 \times 384 \times 240$ | 1 |
| Vortex | Simulation | 1 | $128 \times 128 \times 128$ | 99 |
| Climate | Simulation | 71 | $256 \times 128$ | 1200 |

Table 1. Details of datasets used in figures.

### 5.1 Univariate Queries

In comparison to traditional feature detection using scalar values and 1-D transfer functions, our distribution query method can perform many similar tasks. In fact, SeeDQ can be considered a generalization of traditional preinterpolative 1-D transfer functions. To map a traditional transfer function to a SeeDQ query, we define a bin for each scalar value and add a clause that matches neighborhoods where the scalar value's frequency is non-zero. The predicate signature colormap is set so that the predicate signatures corresponding to each bin map to the color and opacity of the associated scalar value. Finally, we run the query using a neighborhood containing only one voxel.

Queries in SeeDQ also allow users to pick out scalar-based features that traditional transfer functions cannot. For example, Figure 8 shows where dense bone structures occur in the Chest dataset. Using a 1-D transfer function, as in (a), small and thin bones cannot be distinguished from thicker bones since both have the same scalar value. Our distribution-based method, on the other hand, easily pulls out neighborhoods that contain a larger percentage of bone values. In (b), the fully matching neighborhoods are an opaque yellow, and the partial matches (i.e., the thinner bones) are rendered with limited opacity in cyan. A spherical neighborhood of radius 3 (123 voxels) was used as the sample space.



(a) Scalar Volume

(b) Query

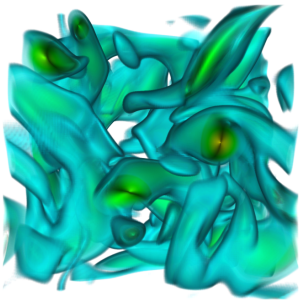
```
bin 0 {
   attribute 0
   range 0 to 24
}
bin 1 {
   attribute 0
   range > 100
}
```

(c) Bins

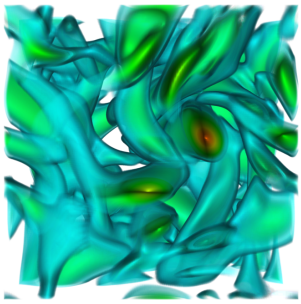
```
clause 0 {
   freq(bin 1) > 1%
   weight 0%
}
clause 1 {
   freq(bin 0) > 20%
   weight 100%
   dropoff 0.12
}
```

(d) Clauses

Fig. 11. (a) Volume rendering of the Visible Male dataset using a 2-D transfer function. (b) A query for neighborhoods that must contain bone and some amount of air. Those with a lot of air, like the sinuses and ear canal, are rendered in magenta. Those with less air are rendered mostly transparent for context. A weight of 0% forces the associated clause to be met for any non-zero score to be assigned.



(a) Timestep 1



(b) Timestep 10

(c) Query Volume

```
bin 0 {
   attribute 0
   range 0 to 13
   attribute 1
   range 0 to 13
}
```

(d) Bins

```
clause 0 {
   covar(bin 0) > 0.3
   weight 100%
   dropoff 0.12
}
clause 1 {
   covar(bin 0) < -0.3
   weight 100%
   dropoff 0.12
}
```

(e) Clauses

Fig. 12. (a) and (b) Timesteps 1 and 10, respectively, of the Vortex dataset, rendered with a 1-D transfer function. (c) A query on a volume containing the two timesteps in (a) and (b). Here, the query is disjunctive, requesting neighborhoods with either positive or negative covariance between the two timesteps. Orange indicates that the two timesteps have a direct correlation, while purple indicates that they are indirectly correlated.

Alternatively, queries may consider the dispersion of values in neighborhoods. For instance, queries can examine a bin's variance metric to find regions of certain homogeneity. Figure 9 shows a query for extremely heterogeneous regions, as shown in white and blue in (b). Since the interface between the supernova and the zero-valued background surrounding it also has a high variance that normally occludes internal structures, we use a compound query to distinguish neighborhoods that do not contain background voxels. The resulting predicate signatures allow us to further discriminate the partially matched neighborhoods so that we can map the occluding surface to a minimal opacity for context.

SeeDQ queries enable users to perform custom filtering on volume data. Figure 10 shows one such filter on a nerve dataset containing a great deal of noise. The query targets neighborhoods that contain more foreground voxels than background. This is not unlike an erode operation in image processing, which could be used to locate weak structures. This query demonstrates a bin-to-bin comparison, allowing the target criteria to be defined in terms of the particular neighborhood being scored.

We've additionally found it useful to support predicates that must be mandatorily matched. Figure 11 shows an example of this, restricting querying to neighborhoods that contain at least 1% bone by specifying a weight of 0%. Another predicate is used to find neighborhoods that contain at least 20% air. In combination, the query finds bone in close proximity to air, revealing the sinuses and ear canals.

### 5.2 Multivariate Queries

Distribution queries easily extend to find features of arbitrary dimension. Any number of physical properties or timesteps can be considered by simply creating a bin to track the desired attribute for each neighborhood.

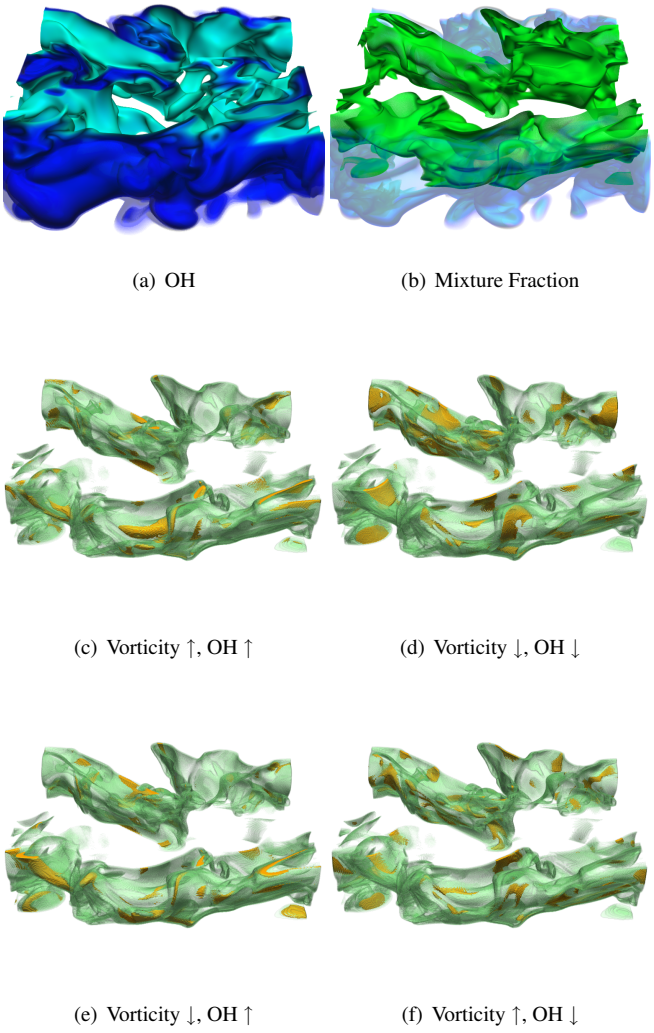
Figure 5 shows a query on two timesteps of the Jet dataset. Here, the user is interested in all scalar values greater than 200 and creates two bins, with bin 0 holding values from timestep 80 and bin 1 from timestep 99. The query states that for a full match, both bins must hold at least 80% of the neighborhood. In (a), the score volume is rendered using the score as a scalar value. Only along the central core of the shockwave do the scores reach 1.0. Outside of this core, the scores drop to 0 as neighborhoods contain fewer voxels having values greater than 200. In (b), we see how the two timesteps intertwine by coloring according to the predicate signatures. Cyan indicates that only timestep 80 was fully matched, magenta indicates timestep 99, and yellow indicates neither was fully matched. (The core where both timesteps match is occluded.) This type of query gives an indication of how much and where two timesteps overlap.

An example of using covariance to compare two timesteps of the Vortex dataset is shown in Figure 12. Here, one predicate queries for neighborhoods where the two timesteps are positively correlated (orange), while another queries for a negative correlation (purple). Two interesting structures appear in the rendered volume where a region of positive correlation envelops a region of negative correlation.

### 5.3 Case Study: Combustion Dataset

Even if a feature like an isosurface is familiar to users, one might still have questions regarding how the feature interacts with other variables. Distribution queries can be used to enhance the feature by incorporating further clauses. Here, we show a relevant scientific example from a study of combustion.

Figure 13 shows several renderings of the Combustion dataset from the Visualization 2008 Meet the Scientists panel. Combustion scientists are curious how regions of flame extinction behave through time [1]. It is already known that flame extinction occurs in locations where the following criteria are observed: (a) a stochiometric mixture rate of 0.42, indicating the presence of flame, and (b) low oxygen content. However, how quickly these regions change in size and how they reignite in relation to other variables is not fully understood. To verify hypotheses made by combustion scientists, we apply SeeDQ in order to see how vorticity, the measure of rotational flow, varies with the amount of available oxygen through time. Four sample queries relating these values with different inequalities are shown in (c) through (e). The clauses used to generate these queries score neighborhoods according to how much of the correct mixture fraction and OH levels they contain and how mean vorticity and OH levels relate between timesteps 121 and 122, the final timesteps of the simulation. Neighborhoods surrounding the dissipating flame and having vorticity and oxygen increase or decrease in mean value in a certain combination are shown in gold, with partially matching neighborhoods rendered in green. In this way, scientists can use frequency queries to posit meaningful relationships between variables through time.



(a) OH (b) Mixture Fraction

(c) Vorticity ↑, OH ↑ (d) Vorticity ↓, OH ↓

(e) Vorticity ↓, OH ↑ (f) Vorticity ↑, OH ↓

Fig. 13. (a) Traditional volume rendering of low OH content from timestep 122 of a combustion simulation. A rainbox colormap is used, with value $7 \cdot 10^{-4}$ colored blue-green. (b) Traditional volume rendering of mixing rate less than 0.42. Value 0.42 maps to green. (c)-(e) Several queries on vorticity and oxygen content through time around locations of flame dissipation. Flame dissipates where $\mathrm{OH} < 7 \cdot 10^{-4}$ and the mixture fraction equals 0.42. Neighborhoods containing these isoranges and having mean vorticity and oxygen content increase or decrease in a certain way through time are shown in gold. For instance, (c) is a query for areas of dissipation where vorticity and oxygen are both increasing in value from timestep 121. Neighborhoods not completely meeting the dissipation ranges or not satisfying the vorticity-oxygen clauses are shown in green. (d) Vorticity and OH both decreasing. (e) Vorticity decreasing, OH increasing. (f) Vorticity increasing, OH decreasing.

### 5.4 Case Study: Climate Dataset

We have also applied neighborhood distribution queries to a climatology simulation. The dataset consists of 71 variables measured as monthly averages for every month in years 2000 through 2099. Variables are measured across a discrete sampling of the Earth's surface with a grid resolution of $256 \times 128$. The simulation uses a land-only model; variables at not measured for bodies of water.

Our collaborators are curious how temperature and precipitation change through time across the Earth's surface. In this case, the neighborhood is defined to be temporal and not spatial, so the distributions we examine are comprised of data values for a single latitude and longitude coordinate through time.

Since the climatologists are curious about changes in two variables, we create four bins for our query: one for precipitation in 2000-2009, one for surface temperature in 2000-2009, one for precipitation in 2090-2099, and one for surface temperature in 2090-2099.

To determine how temperature and precipitation relate, we formulate clauses to query for each possible change to the variables. Mean temperature may go up by some amount from the beginning of the century to the end, or it may go up by some smaller amount. (In this simulation, in no locations does mean temperature go down.) Mean precipitation may go up, stay the same, or go down.

Figure 14 shows the result of applying this query to the IPCC climate dataset. Our collaborators have verified the consistency of this result with their expectations. Extreme latitudes see more significant temperature increases, and relative precipitation levels generally are exaggerated—dry locations become drier and wet locations become wetter—as time progresses.

### 5.5 Timing Results

To distribute the computational burden of evaluating an arbitrary query on neighborhoods centered on all of a volume's voxels, SeeDQ operates in parallel on subvolumes of the original dataset. We tested our tools on a 16-node Linux cluster, equipped with dual-core 3.2 GHz Intel Xeon processors with 4 GB RAM. Given the natural parallelism of our approach, performance in our MPICH implementation scales linearly with the number of processors for a query containing three predicates and three bins, as seen in Table 2.

Processing times are not interactive due to the high computational demand, which is proportional to the neighborhood size, the number of bins and clauses, and the number of full neighborhoods contained in the volume. For this reason, we recommend that queries first be built using the query formulation tool, which operates at interactive speeds on a single preview slice of the volume. After testing and tweaking, the parallel 3-D query tool should be used to produce full volume renderings.

A considerable amount of overhead is introduced by optimizations, but the benefits are quickly seen as processing times grow by a much smaller factor than do the neighborhood sizes. Given that small neighborhoods have less statistical significance, we favor performance for larger neighborhood sizes.

The GPU-based volume renderer was built using the OpenGL Shading Language (GLSL) and operates much like traditional hardware-

| Neighborhood Size | | Mean Query Time Per CPU (seconds) | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 Node | 2 Nodes | | 4 Nodes | | 16 Nodes | |
| Radius | Number of Voxels | Time | Time | Speedup | Time | Speedup | Time | Speedup |
| 1 | 7 | 15.70 | 6.06 | 2.6x | 3.41 | 4.6x | 0.92 | 17.1x |
| 2 | 33 | 19.98 | 9.86 | 2.0x | 4.29 | 4.6x | 1.28 | 15.6x |
| 3 | 123 | 33.08 | 12.18 | 2.7x | 7.16 | 4.6x | 2.04 | 16.2x |
| 4 | 257 | 52.08 | 26.18 | 2.0x | 13.30 | 3.9x | 3.24 | 16.1x |

Table 2. Average processing time per node for a three-bin, three-predicate query on the $256^3$ Tornado dataset for spherical neighborhoods of increasing size and voxel count. In this example, the number of voxels in each neighborhood is a count of all voxels within a Euclidean distance of the radius value.

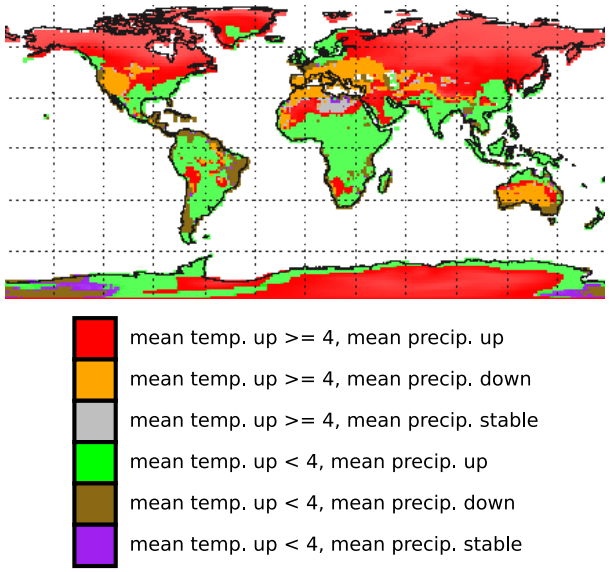



Fig. 14. A query on the interrelation between precipitation and temperature through time on a climate simulation. The color legend indicates how the means of the two variables move in tandem from the first decade of the millenium to the last decade. Clauses query for locations where temperature increases by more or less than 4 degrees Celsius and where the amount of precipitation increases, decreases, or holds constant.

accelerated slice-based volume renderers. Framerates on an NVIDIA GeForce 8800GT are approximately 25 fps for a $256^3$ volume, 512 by 512-pixel viewport, and a sampling rate of 1 sample per voxel. The added complexity of interpolating predicate signatures at subvoxel precision is minor compared to a multipass solution to this problem, incurring a cost of 5.5 ms per frame compared to a reference volume renderer.

## 6 Conclusion

Local frequency distributions offer users a new handle on gaining insight from volumetric datasets. By querying for features in distribution space with SeeDQ, we can see new results that agree with and enhance traditional feature detection methods based on scalars and gradients, in addition to locating a whole new class of features that were not previously accessible to or easily specified by the user. Furthermore, features may be defined across any number of variables or timesteps. The resulting query volume can be rendered for real-time feature analysis. Features can be quantitatively described in two ways from the generated imagery: 1) by associating color and the corresponding set of matched clauses, users can describe what combination of criteria a neighborhood meets; and 2) by using the preview tool to examine exactly how and to what extent a location matches both individual clauses and the entire query. Such meaningful interpretation and feedback is essential to the process of investigating our data.

For future work, we are interested in supporting more distribution primitives for detecting other kinds of features. We also want to investigate building into our query formulation tool mechanisms so that users can better investigate neighborhood distributions and how they correlate to features. Distributions are generally observed on a volume-wide scale, and the patterns seen in smaller regions may not be widely known. The creation of custom neighborhood shapes and selection of neighborhood size could be enhanced by allowing users to select a prototype neighborhood containing a known feature. Additionally, it would be interesting to adjust neighborhood shapes and sizes according to their context. This flexibility would accommodate features that vary in size. Finally, we believe a GPU-implementation of SeeDQ would realize significant reductions in computation time.

## Acknowledgements

The authors wish to thank Rob Sisneros, Markus Glatter, and Forrest Hoffman for valuable discussion and feedback. The Combustion dataset is made available by Dr. Jacqueline Chen at Sandia Laboratories through US Department of Energy's SciDAC Institute for Ultrascale Visuaization. This work was supported in part by DOE Early Career PI grant DE-FG02-04ER25610, NSF grants ACI-0329323 and CNS-0437508, and in part by the Institute for Ultra-Scale Visualization, under DOE SciDAC program (grant No. DE-FG02-06ER25778).

## References

[1] H. Akiba, K.-L. Ma, J. H. Chen, and E. R. Hawkes. Visualizing multivariate volume data from turbulent combustion simulations. *Computing in Science and Engineering*, 9(2):76–83, 2007.

[2] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *Proceedings of IEEE Visualization*, pages 167–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[3] W. Cai and G. Sakas. Data intermixing and multi-volume rendering. *Computer Graphics Forum*, 18(3):359–368, 1999.

[4] R. Crawfis and N. Max. Texture splats for 3d vector and scalar field visualization. In *Proceedings of IEEE Visualization*, pages 261–266, October 1993.

[5] H. Doleisch, M. Gasser, and H. Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *VISSYM '03: Proceedings of the Symposium on Data Visualisation 2003*, pages 239–248, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[6] S. Fang, T. Biddlecome, and M. Tuceryan. Image-based transfer function design for data exploration in volume visualization. In *Proceedings of IEEE Visualization*, pages 319–326, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[7] L. Gosink, J. Anderson, W. Bethel, and K. Joy. Variable interactions in query-driven visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1400–1407, 2007.

[8] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of IEEE Visualization*, page 40, Washington, DC, USA, 2003. IEEE Computer Society.

[9] J. Hladůvka, A. König, and E. Gröller. Salient representation of volume data. In D. Ebert, J. M. Favre, and R. Peikert, editors, *Data Visualization 2001, Proceedings of the Joint Eurographics – IEEE TVCG Symposium on Visualization*, pages 203–211,351, 2001.

[10] H. Jänicke, A. Wiebel, G. Scheuermann, and W. Kollmann. Multifield visualization using local statistical complexity. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1384–1391, 2007.

[11] D. A. Keim and H. P. Kriegel. VisDB: database exploration using multidimensional visualization. *Computer Graphics and Applications, IEEE*, 14(5):40–49, 1994.

[12] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium on Volume Visualization*, pages 79–86, 1998.

[13] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization*, pages 513–520, 2003.

[14] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, July-September 2002.

[15] J. M. Kniss, R. V. Uitert, A. Stephens, G.-S. Li, T. Tasdizen, and C. Hansen. Statistically quantitative volume visualization. In *Proceedings of IEEE Visualization*, volume 00, page 37, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[16] D. H. Laidlaw, K. W. Fleischer, and A. H. Barr. Partial-volume bayesian classification of material mixtures in mr volume data using voxel histograms. *IEEE Trans. Med. Imaging*, 17(1):74–86, 1998.

[17] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[18] C. Lundström, P. Ljung, and A. Ynnerman. Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1570–1579, 2006.

[19] C. Lundström, P. Ljung, and A. Ynnerman. Multi-Dimensional Transfer Function Design Using Sorted Histograms . In R. Machiraju and T. Möller, editors, *Volume Graphics*, pages 1–8, Boston, Massachusetts, USA, 2006. Eurographics Association.

[20] P. McCormick, J. Inman, J. Ahrens, C. Hansen, and G. Roth. Scout: A hardware-accelerated system for quantitatively driven visualization and analysis. In *Proceedings of IEEE Visualization*, pages 171–178, 2004.

[21] A. Pang, C. M. Wittenbrink, and S. K. Lodha. Approaches to uncertainty visualization. *The Visual Computer*, 13(8):370–390, 1997.

[22] V. Pekar, R. Wiemker, and D. Hempel. Fast detection of meaningful isosurfaces for volume data visualization. In *Proceedings of IEEE Visualization*, pages 223–230, Washington, DC, USA, 2001. IEEE Computer Society.

[23] Y. Sato, C.-F. Westin, A. Bhalerao, S. Nakajima, N. Shiraga, S. Tamura, and R. Kikinis. Tissue classification based on 3d local intensity structure for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):160–180, 2000.

[24] N. Sauber, H. Theisel, and H.-P. Seidel. Multifield-graphs: An approach to visualizing correlations in multifield scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):917–924, 2006.

[25] A. Stompel, E. B. Lum, and K.-L. Ma. Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques. In *Proceedings of 10th Pacific Graphics Conference on Computer Graphics and Applications (PG '02)*, volume 00, page 394, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[26] S. Tenginakai, J. Lee, and R. Machiraju. Salient iso-surface detection with model-independent statistical signatures. In *Proceedings of IEEE Visualization*, pages 231–238, Washington, DC, USA, 2001. IEEE Computer Society.

[27] U. Tiede, T. Schiemann, and K. H. Höhne. High quality rendering of attributed volume data. In *Proceedings of IEEE Visualization*, pages 255–262, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[28] F.-Y. Tzeng, E. Lum, and K.-L. Ma. A novel interface for higher-dimensional classification of volume data. In *Proceedings of IEEE Visualization*, pages 505–512, 2003.

[29] J. Woodring and H.-W. Shen. Multi-variate, time varying, and comparative visualization with contextual cues. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):909–916, 2006.