A Compact Multivariate Histogram Representation for Query-driven Visualization

Kewei Lu[*]
The Ohio State University

Han-Wei Shen[†]
The Ohio State University

Abstract

As the size of data continues to increase, distribution-based methods become increasingly more important for data summarization and queries. To represent the distribution from a dataset without relying on a particular parametric model, histograms are widely used in many applications as it is simple to create and efficient to query. For multivariate scientific datasets, however, storing multivariate histograms in the form of multi-dimensional arrays is very expensive as the size of the histogram grows exponentially with the number of variables. In this paper, we present a compact structure to store multivariate histograms to reduce its huge space cost while supporting different kinds of histogram queries efficiently. A data space transformation is employed first to transform the large multi-dimensional array to a much smaller array. Dictionaries are constructed to encode this transformation. Then, the multivariate histogram is represented as a sequence of index and frequency pairs where the indices are represented as bitstrings computed from a space filling curve traversal of the transformed array. With this compact representation, the storage cost for the histograms is reduced. Based on our representation, we also present several common types of queries such as histogram marginalization, bin-merging and computation of conditional probability. We parallelize both the histogram computation and queries to improve its efficiency. We present several query-driven visualization applications to explore and analyze multivariate scientific datasets. Experimental results to study the performance of our framework in terms of scalability and space cost are also discussed.

1 Introduction

Distributions play an important role in analyzing and visualizing data generated from various scientific and engineering simulations, as they are particularly effective for data aggregation, summarization, and query. Previously, locally and globally computed distributions have been used in transfer function design for volume rendering [11, 13]. Given the distributions computed from a multivariate dataset, different statistical metrics such as mean, variance and information entropy can be computed to facilitate the analysis and visualization of the data. Point-wise distributions have been used for flow visualization based on information theory [24]. Several query-driven visualization techniques also utilize distributions. By querying regions that contain a certain type of distribution, salient features can be identified from the volume data [10]. In [8], Gosink et al. showed distributions are useful for segmentation and extraction of salient features. Block level distributions are used to track features in time varying data in [9]. Distributions have also been used to predict isosurface statistics [2, 3]. In recent years, the rapid increase in the speed of processors and the number of processing cores empowers scientific simulations to run at unprecedented spatial and temporal resolutions, which in turn produce very large scale datasets. As a result, it is expected that distributions will play an increasingly more important role in supporting the need of large scale data analytics.

Data distributions can be represented using parametric or non-parametric models. For parametric models, it is assumed that data come from a particular type of distribution, which can be described by just a few parameters. Non-parametric models, on the other hand, make no assumptions about the data and hence can model a wider range of distributions. A popular non-parametric model for distributions is the histogram, which is commonly used because the distribution model for data generated from a simulation is often unknown. To support various analysis needs, there have been works on scalable computation of univariate histograms at different levels of detail from large scale data [5] and efficient range distribution query using integral histograms [6, 15, 12]. Most of the works focused on computing 1D histograms from univariate data, while relatively little research has been done for computation and storage of histograms for multivariate datasets.

Multivariate datasets are frequently encountered in scientific applications. Compared to univariate data, analyzing and visualizing multivariate datasets is much more challenging and hence remains an active topic of research. Computing and storing multivariate histograms is nontrivial because the memory requirement can grow exponentially with respect to the number of variables if stored as multi-dimensional arrays. Furthermore, different analysis tasks may require multivariate histograms that have different combinations of variables with different number of bins. Because of the computation and storage cost, computing multivariate histograms from the raw data at run time is often very expensive.

To solve aforementioned problems, in this paper, we present a novel technique to store multivariate histograms using their sparse property to reduce the storage cost. Naturally, a multivariate histogram can be represented as a multi-dimensional array. We first transform the large multi-dimensional array to an array of much smaller size based on the sparseness of multivariate histograms. Dictionaries are constructed to encode this transformation and can be used to map the transformed multi-dimensional array back to the original array. Then, we represent the multivariate histogram as a sequence of index and frequency pairs where the indices are represented as bitstrings computed from a space filling curve traversal of the multi-dimensional array. To support different types of applications, we demonstrate several histogram query operations such as marginalization, histogram bin-merging, conditional histogram and time histogram query using our representation. We also present several applications to analyze and visualize multivariate datasets. The contributions of this paper are:

- We present a novel technique to store multivariate histograms while reducing the storage requirement(Section 3);
- We present several query operations to extract different types of histograms from multivariate histograms stored in the proposed format(Section 4);
- We present several query-driven visualization applications using the proposed representation(Section 5);
- We demonstrate the effectiveness of our techniques for large datasets(Section 6).

[*] e-mail:luke@cse.ohio-state.edu
[†] e-mail:hwshen@cse.ohio-state.edu

IEEE Symposium on Large Data Analysis and Visualization 2015
October 25–30, Chicago, Il, USA
978-1-4673-8517-6/15/$31.00 ©2015 IEEE

2 Related Work

Distribution-based applications. As an effective way to summarize large scale scientific data, distributions have been used in various data analysis and visualization applications. HIXEL, introduced by Thompson et al. [20], stores one histogram per grid point or a data block. Based on this representation, the authors proposed an algorithm for fuzzy isosurface computation. Gu et al. [9] compute block-level histograms to track features presented in time-varying datasets. Lundstrom et al. [13] studied the design of transfer functions in direct volume rendering based on local histograms and demonstrated its effectiveness in a clinical evaluation. Given the distributions computed from data, many statistical metrics can be derived. Xu et al. [24] used Shannon's entropy computed from point-wise distributions to analyze the complexity of local data. Martin and Shen [14] introduced histogram spectra which are computed in each subvolume over a range of sampling frequency. The histogram spectra is used to achieve interactive level of detail selection for large scale time-varying multivariate data.

Distribution computation. Due to the important role that distributions play in data analysis and visualization, various techniques have been proposed for efficient computation of distributions. Integral histogram was proposed by Porikli et al. [16] to compute the histogram of arbitrary region in constant time. However, because a histogram contains multiple bins, storing one integral histogram per grid point is very expensive and hence better solutions are needed for three-dimensional data. Martin and Shen [15] developed span distributions to reduce the I/O and memory costs for range queries. Two novel transformations, a decomposition and a similarity-driven indexing, of the integral histogram was proposed by Chaudhuri [6] to reduce the storage cost associated with integral histograms. Lee et al. [12] tackled the storage problem of integral histograms by using the discrete wavelet transform. When the data size is large, distributions could be computed on distributed machines in parallel [5]. Rubel et al. [17] used FastBit to achieve efficient parallel computation of 2D histograms and conditional histograms. In [18], the authors presented several efficient algorithms for computing 2D conditional histograms using bitmap indexing in parallel. All these methods focus on efficient computation of univariate or 2D histograms, while in this paper, we focus on the computation of multivariate histograms.

Query-driven visualization. Query-driven visualization(QDV) allows users to analyze and visualize large scale data by selecting data records that are defined to be interesting based on a specific criterion. Those specific criteria could be specified as Boolean range queries such as $(100 \leq \mathit{pressure} \leq 200)$ AND $(0.0 \leq \mathit{QRain} \leq 0.01)$ [19]. Effective QDV techniques rely on fast retrieval of those data records. Different indexing techniques such as FastBit [19, 22, 23] have been used to effectively identify those interesting data records. Distribution based query driven techniques have been proposed in [10, 20]. In this paper, our query criteria are specified with respect to histograms. Instead of identifying the data records that match the query criteria, we query for those stored block-wise histograms which match the specified query criteria and each block will get a value which represents how well the block matches the query criteria.

Multivariate histogram. Unlike univariate histogram, the computation of multivariate histogram is nontrivial. One challenge related to multivariate histograms is the curse of dimensionality which states that as the number of variables increases, the size of the multivariate histogram increases exponentially. To solve this problem, a compact representation was proposed by Chanussot [4]. In Chanussot's method, the data space is indexed by a space filling curve so that the N dimensional vector could be represented as a single scalar value which is its curvilinear abscissa along the space filling curve. Then, instead of storing all the entries of the multivariate histogram, they only store the non-empty entries and represent the multivariate histogram as a table with two columns: the first column contains the index of the non-empty entries. The second column stores the corresponding frequencies. A potential problem is that given a large number of variables and bins, the curvilinear abscissa could be an extremely large value that requires a large number of bits to represent. The authors also present another modified representation in which the first column records the number of empty entries between the current non-empty entry with the previous non-empty one. By using this modified representation, it will give relative smaller values in the first column. However, a problem related to this representation is that it does not explicitly contain the bin index for each variable which is required by several query operations. Summarization and factorization is needed to get the bin index of each variable. Goil et al. [7] presented a bit-encoded sparse structure(BESS) to store multi-dimensional sparse data. The multi-dimensional data is divided into chunks first, and then each chunk is indexed separately with bit strings. Since the size of a chunk is smaller than the original data, they require less bits to encode. The chunk offset for each chunk also needs to be stored in order to dereference to get the original bin indices. In this paper, we propose a new representation to store block-wise multivariate histograms with less storage. Instead of dividing the multidimensional histogram into chunks, a data space transformation is applied first to reduce the size of the original multivariate histogram and then the smaller multivariate histogram is indexed. Compared with chunking, this representation gives us an one to one mapping from the encoded bin index to the original bin index which allows us to perform certain query operations such as histogram marginalization without decoding.

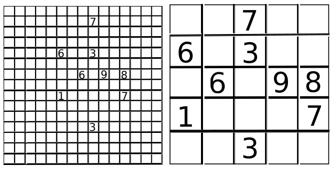
3 Multivariate Histogram Representation

A nontrivial problem related to multivariate histogram computation is how to represent and store the multivariate histogram. A straightforward way is to store the multivariate histogram as a multi-dimensional array. However, the memory requirement of using a multi-dimensional array would increase considerably as the number of variables and bins increase. Suppose that there are N variables, and we use the same number of bins for all dimensions and the number is B, if the frequencies of the multivariate histogram are stored as floating point numbers, we will need B^N floating point numbers to represent the multivariate histogram. The memory cost grows exponentially with respect to the number of variables.

To reduce the memory requirement for storing a multivariate histogram, we use the sparseness property of multivariate histograms. Given a dataset with P data points, at most P entries in the multivariate histogram have non-zero frequencies while all the other entries are zeros. For large scale datasets, if we divide the entire dataset into smaller blocks and compute a multivariate histogram for each block, the number of data points per data block is even smaller than the number of entries in the multivariate histogram. Considering a block with size of 8^3 and each data point has 3 attributes, if we construct a multivariate histogram with 256 bins for each dimension, at most $8^3/256^3 \approx 0.0031\%$ entries of the multivariate histogram are not zeros.

3.1 Data Space Transformation

Given this super-sparse block-wise multivariate histogram, a data space transformation is employed first to reduce the size of the multivariate histogram. The main motivation of this data space transformation comes from the following observation: for scientific multivariate datasets, given a block, the value for an arbitrary variable within that block changes smoothly. Also because all block-wise multivariate histograms are computed using the same global value range for the same variable, for an arbitrary dimension, those non-empty entries usually locate in a smaller number of bins along that



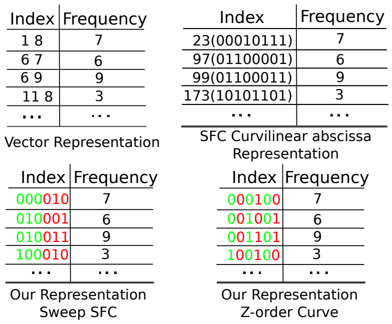
(a)

								7						
					6			3						
							6		9		8			
					1						7			
								3						

(b)

		7		
6		3		
	6		9	8
1				7
		3		

Figure 1: A multivariate histogram with two variables, each with 15 bins. (a) The original multidimensional array. (b) The transformed multidimensional array.



Index	Frequency
1 8	7
6 7	6
6 9	9
11 8	3
...	...

Vector Representation

Index	Frequency
23(00010111)	7
97(01100001)	6
99(01100011)	9
173(10101101)	3
...	...

SFC Curvilinear abscissa Representation

Index	Frequency
000010	7
010001	6
010011	9
100010	3
...	...

Our Representation Sweep SFC

Index	Frequency
000100	7
001001	6
001101	9
100100	3
...	...

Our Representation Z-order Curve

Figure 2: Different representations.

dimension instead of spreading out a large number of bins along that dimension.

The following table shows the average number of distinct index values for each variable collected from a number of block-wise multivariate histograms that are computed for each dataset:

Data	V_0	V_1	V_2	V_3	V_4
Isabel	10.4	10.0	5.8	3.2	4.1
Combustion	5.8	12.0	11.1	12.5	9.7
Ion Front	9.9	10.7	1.9	10.7	13.3

Three datasets and five variables for each dataset are used. The number of bins are set to 256 for each variable. As it shows, for all those variables, the number of distinct index values is much smaller than 256. Through data space transformation, the extremely large and sparse multi-dimensional array can be converted to a much smaller and denser multi-dimensional array which requires much smaller space compared with the original multi-dimensional array.

Let $V = V_0, V_1, ..., V_{N-1}$ denote the set of variables that the multivariate histogram is computed from and B_i represents the number of bins for the i_{th} variable. Then, the multivariate histogram can be represented as a N-dimensional array M_O with size $B_0 \times B_1 \times \cdots \times B_{N-1}$. The data space transformation operation transforms M_O to another N-dimensional array M_T with size $P_0 \times P_1 \times \cdots \times P_{N-1}$, where usually $P_i \ll B_i$ for $0 \leq i \leq N-1$.

A dictionary is constructed to record the transformation for each variable. The dictionary provides an one to one mapping for each variable from the bin index computed from M_T to the bin index computed from M_O. Let D_i denote the dictionary constructed for the i_{th} variable, given a bin index $j \in [0, P_i - 1]$, D_i maps j to a unique value $k \in [0, B_i - 1]$ which is the original bin index, we refer this step as decoding.

Given a multi-dimensional array, to perform transformations and generate dictionaries for each variable, for each variable V_i, let $bin(V_i)$ denote its bin index, for every $bin(V_i) \in [0, B_i]$, if all those histogram entries with the bin index $bin(V_i)$ have zero frequency, we remove all those entries, otherwise, we insert a new entry to the dictionary for the variable V_i. After the transformation, we have a much smaller array to be indexed. Compared with directly indexing the original multi-dimensional array, indexing the transformed multi-dimensional array requires less number of bits so that reduce the storage overhead. A data space transformation example is shown in Figure 1.

3.2 Multivariate Histogram Construction

After the data space transformation, we represent the resulting multi-dimensional array as a collection of index and frequency pairs, where each pair corresponds to a nonempty entry in the multi-dimensional array. Similar to the method presented in [4], we also use space filling curve to index the data space. However, before the indexing, we pad the resulting multi-dimensional array from data space transformation to be power of two along all its dimensions, so that we could directly use bit concatenation or bit interleaving to compute the index based on whether we use sweep or Z-order space filling curve to index the data space. When the sweep space filling curve is used, the computed indices is equivalent to using BESS [7] to index the transformed array without chunking.

The main benefit of this padding step is that after padding, different bits of the index correspond to the bin indices of different variables, so that we could directly manipulate individual bits to operate the bin index of different variables. Since many histogram queries require the bin indices of different variables, this feature allows us to perform those histogram queries with efficient bit-wise operations.

In our representation, a multivariate histogram is represented as dictionaries which are used to record the result of data space transformation, and the index and frequency pairs. In this representation, an index can be decomposed into different parts where each part represents the bin index of a variable after the data space transformation. Hereafter we refer to the bin index that we get from the transformed multi-dimensional array as *transformed bin index*. The dictionary can be used to map the transformed bin index to the bin index computed from the original data space. We refer to the bin index computed from the original multi-dimensional array as *original bin index*. An example of different representations for the multivariate histogram is shown in Figure 2. In our representation, the bits corresponding to the first variable are colored red and the bits corresponding to the second variable are colored green. We could manipulate these bits to operate the bin index for different variables. In Section 4 we presented several common query operations to obtain different types of histograms by manipulating those bits. There are several benefits of using our representation:

- Compared with the vector and space filling curve curvilinear abscissa representations, the storage cost of our representation is reduced, because we first do a data space transformation to reduce the size of the multi-dimensional array dramatically and then use a space filling curve to index the transformed multidimensional array. Fewer bits are used to represent the index. Even though we keep dictionaries to record the transformation, the overall storage cost is still reduced in general;
- Compared with Goil's representation, because of the one-to-one mapping from the transformed bin index to the original bin index of our representation, the decoding step is not necessary for all queries and also not needed for all variables. Based on the type of query, we can directly obtain the target histogram without decoding or only having to decode a subset of variables.

4 Histogram Query

In this section, we present several query operations using our histogram representation to derive novel histograms for different visualization applications. Here we assume the sweep space filling

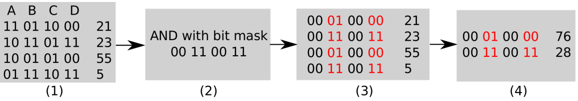


Figure 3: Query for histogram P(B,D). (1) A multivariate histogram which has four variables A, B, C, and D. (2) An AND operation is performed with a bit mask. (3) The result after the AND operation. (4) The entries with the same index are summed together to get the marginalization result.

curve is used. One query example is histogram marginalization which is useful for users to investigate the relationships among a subset of variables. Multiple query operations can also be combined together, for example the user can first perform a histogram marginalization, followed by a histogram bin merge to obtain a lower resolution histogram among the selected variables. Below we describe the algorithms to perform such queries using our representation in detail.

4.1 Histogram Marginalization

The first operation that we present here is histogram marginalization. The purpose of this operation is to compute a multivariate histogram that involves an arbitrary subset of variables. We assume that an initial histogram computed from all variables, stored in the representation described above, is already available, but the user wants a multivariate histogram related to only a subset of variables. To compute this new histogram, we marginalize the pre-stored multivariate histogram over the variables that have not been selected by the user. Suppose the pre-computed histogram involves four variables A, B, C and D and the user is interested in the histogram of B and D. To compute the new histogram, we marginalize the multivariate histogram over variables A and C, that is: $P(B=b, D=d) = \sum_a \sum_c P(A=a, B=b, C=c, D=d)$

As discussed in Section 3, the multivariate histogram is stored as dictionaries, along with index and frequency pairs where each index is represented as a bit string. Since in our representation, different bits in the index correspond to the bin indices of different variables, the histogram marginalization operation can be easily performed using a bitwise AND operation, followed by a frequency sum. For the histogram marginalization operation, we do not need to decode the index to map the transformed bin index to the original bin index. The marginalization operation can be directly applied on the transformed bin index. This is because that:

- For a histogram marginalization operation, whether two histogram entries need to be summed together or not only depends on whether these two entries have the same bin index for the marginal variables.
- Given two histogram entries, if the transformed bin indices for a variable are the same, the original bin indices for this variable must be the same.

When using our representation, the marginalization can be done by setting the bits corresponding to the variables that are not selected by the user to 0s and then summing up the frequencies that have the same index. A simple AND operation with a bit mask is used to set the necessary bits to 0s. An example for this marginalization operation is shown in Figure 3. This multivariate histogram has 4 variables A, B, C and D. After data space transformation and padding, each variable has 4 bins, so the length of the index is 8 and every 2 bits represent the bin index of a variable.

4.2 Histogram Bin Merge

Histogram bin merge is to combine the frequencies in adjacent bins along certain dimensions of the multivariate histogram to arrive at a new histogram with a different level of value discretization. This operation is used when certain analytical tasks require histograms of different resolutions. We assume that the precomputed multivariate histogram is defined at a higher resolution. The number of bins for each variable before data space transformation is chosen to be power of 2 for convenience reason. Suppose the number of bins before data space transformation for the high resolution multivariate histogram with 4 variables, for example, is $2^{K_1}, 2^{K_2}, 2^{K_3}, 2^{K_4}$, respectively for each variable, a lower resolution histogram with $2^{K_1-1}, 2^{K_2-2}, 2^{K_3-1}, 2^{K_N-0}$ bins for each variable can be accurately computed by summing the frequencies of the consecutive bins groups of size $2^1, 2^2, 2^1, 2^0$. For this operation, we need to decode first and then perform the operation.

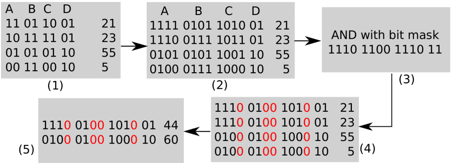


Figure 4: (1) A multivariate histogram which has 4 variables. (2) We query for a lower resolution multivariate histogram with $L = \{1,2,1,0\}$, and we decode the bin index for variable A,B and C to get the original bin index. (3) A logical AND operation is performed with a bit mask. (4) The result after the AND operation. (5) The entries with the same index are summed together to get the bin merge result.

In term of the query procedure, the user specifies how many levels he wants to coarsen from the original multivariate histogram along each dimension by providing a N dimensional vector where N is the number of variables. Histogram bin merge operation is performed by using the same bit-wise AND operation as the histogram marginalization operation except that the bit mask in use is different. Given a high resolution multivariate histogram with N variables with $2^{K_1}, 2^{K_2}, \ldots, 2^{K_{N-1}}, 2^{K_N}$ bins respectively for each variable, the user specifies a N dimensional vector $L = \{l_1, l_2, \ldots l_{N-1}, l_N\}$ where $0 \leq l_i \leq K_i$ for $i = 1, \ldots, N$ to obtain the lower resolution histogram of $2^{K_1-l_1}, 2^{K_2-l_2}, \ldots, 2^{K_{N-1}-l_{N-1}}, 2^{K_N-l_N}$ bins. When $l_i = 0$, it means we do not need to perform bin merge operation along the i_{th} variable. In our implementation, we only decode the bin index for the variables whose level to coarsen is not zero. Given $L = \{l_1, l_2, \ldots l_{N-1}, l_N\}$ where $l_2 = 0$ and $l_N = 0$, suppose the number of bits used for variable 2 and N after the data space transformation is m_2 and m_N, the corresponding bit mask is:

$$\underbrace{1,\ldots,1}_{K_1-l_1}\underbrace{0,\ldots,0}_{l_1}\underbrace{1,\ldots,1}_{m_2}\underbrace{1,\ldots,1}_{K_3-l_3}\underbrace{0,\ldots,0}_{l_3},\ldots,\ \underbrace{1,\ldots,1}_{K_{N-1}-l_{N-1}}\ \underbrace{0,\ldots,0}_{l_{N-1}},\underbrace{1,\ldots,1}_{m_N}$$

An histogram bin merge example is shown in Figure 4. The multivariate histogram has 4 variables. Before the data space transformation, each variable has 16 bins while after the data space transformation and padding each variable has 4 bins.

4.3 Conditional Histogram

The next operation that we present is the conditional histogram query. Given N variables $V = v_1, v_2, \ldots, v_N$, a set of variables $U \subset V$ and another set of variables $W \in (V - U)$, this operation computes the distribution of U given the variables in W in particular bin ranges. The conditional histogram is useful when the user wants to investigate the relationship between a set of variables U and another set of variables W when the variables in W are in the specific value ranges. For example, we can compute the entropy of $P(U|W)$ which shows the uncertainty of U given the variables in W in the specific value ranges. Suppose U contains two variables B and C, W contains two variables A and D, and the specific value range for

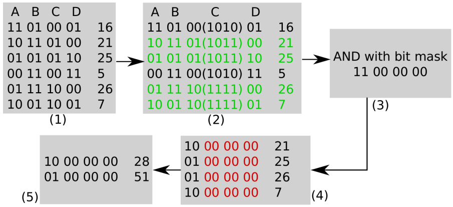


Figure 5: Query for $P(A|11 \leq bin(C) \leq 15)$. (1) A multivariate histogram which has 4 variables A, B, C and D. (2) We decode the bin index for variable C to get the original bin index and then the corresponding entries with $11 \leq bin(C) \leq 15$ are selected(green color). (3) A logical AND operation is performed with a bit mask for those selected entries. (4) The result after the AND operation. (5) The entries with the same index are summed together

variable A is (min_A, max_A) and for variable D is (min_D, max_D), the conditional histogram query operation is defined as:

$$P(bin(B) = b, bin(C) = c | min_A \leq bin(A) \leq max_A, min_D \leq bin(D) \leq max_D) = \sum_{a=min_A}^{max_A} \sum_{d=min_D}^{max_D} P(bin(A) = a, bin(B) = b, bin(C) = c, bin(D) = d)$$

The first step to compute the conditional histogram is to select all the entries that fall into the specific bin ranges for the variables in W. For this selection step, we need to decode for the variables in W to get their original bin index. Then, for each selected entries, a logical AND operation is performed with a bit mask to set all the bits not corresponding to the variables in U to 0s. The final results are computed by summing up the frequencies of the entries that have the same index.

An example is shown in Figure 5. This multivariate histogram has 4 variables A, B, C and D. Before the data space transformation, each variable has 16 bins while after the data space transformation and padding each variable has 4 bins.

4.4 Time Histogram

The last operation that we present is the time histogram query. Our method can be easily extended to time varying multivariate data. For time varying multivariate data, we treat the time step as another variable and set the number of bins to be the number of time steps. When a time varying data is used, we can query the time histograms [1] for an user selected data block. The queried time histogram is useful to show the features presented in this selected data block over time. Time histogram query is similar as histogram marginalization query except that the variable time step must be included in the marginal variables. In term of the query procedure, the user specifies the variables and the data block from which the time histogram is computed. The variable time step must be included in those selected variables, and then the rest of the query is the same as histogram marginalization query.

5 Visualization Applications and Analysis

5.1 Local Statistical Analysis

Several statistical metrics such as mean, variance and information entropy [24] can be computed from block-wise multivariate distributions to determine whether certain features exist in the blocks. Computing such block-wise distributions from large scale data, however, is expensive. Also, when the user changes the combination of variables or the resolution of the desired histograms, we need to repeat the computation and access the raw data, which can be very costly. With our technique, we do not need to access the raw data, instead, we can efficiently derive histograms from different combinations of variables at different resolutions. Figure 6 shows an example of local statistical analysis using our technique.

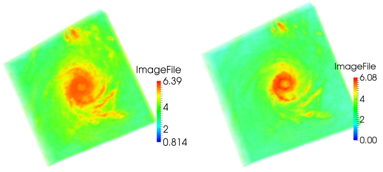


Figure 6: Local statistical analysis for Isabel.

The dataset used is Isabel, which contains a total of 13 variables. Initially, we compute and store the block-wise multivariate histograms from all the 13 variables. At analysis time, we read back these pre-stored multivariate histograms and then compute the required histograms for analysis. First, the user queries the multivariate histograms of selected combination of variables for each block by marginalization, and then the multivariate histograms are coarsened to a desired bin resolution by histogram bin merge.

We show a block-wise entropy field computed from the multivariate histograms of U, V and W velocity fields in Figure 6. The left image uses the multivariate histograms at the finest level of detail (256 bins for each variable). Lower bin resolution histograms (128 bins for each variable) are used to generate the right image.

5.2 Query Driven Visualization

Given the multivariate histograms computed from each block, query-driven techniques can be used to visualize and analyze the multivariate dataset. During analysis, the user can easily retrieve the required histograms to perform query-driven visualization with our techniques. In this section, we present four different query-driven visualization applications based on our histogram representation.

5.2.1 Joint Probability Query

Since the data is decomposed and represented as multiple block-wise multivariate histograms, an useful query would be to query for blocks where the co-occurrence of some variables satisfies a user defined condition. This kind of query requires us to efficiently derive the multivariate histogram for the selected variables. An example of such a query is: $P(200 \leq bin(A) \leq 255, 0 \leq bin(B) \leq 50) > 70\%$. In this query, we query for the blocks where the probability for variable A belonging to bin 200 to 255 and variable B belonging to bin 0 to 50 is bigger than 70%. This query gives us the blocks where the majority of points have high value for variable A and low value for variable B. We can also use OR to combine multiple queries. For example, $P(200 \leq bin(A) \leq 255, 0 \leq bin(B) \leq 50) > 70\%$ OR $P(100 \leq bin(C) \leq 150) > 50\%$. With our technique, we can quickly answer the query by computing the desired multivariate histogram of the selected variables to get the probability. A fuzzy similarity score proposed by Johnson and Huang [10] can be used to support fuzzy matching. The similarity score has a value range of 0 to 1 while larger value means they are more similar.

Figure 7 shows two query results using Isabel and Combustion. In the left image, we query the blocks that the probability of Pressure belongs to bin 0 to 50 or the accumulate probability of Cloud belongs to bin 30 to 255 and QRain belongs to bin 30 to 255 is bigger than 50%. The low Pressure query gives us the hurricane eye while the region corresponding to hurricane wall contained in the higher Cloud and QRain query. In the right image, we query the blocks that the probability of OH belongs to bin 0 to 96 and Mixture fraction belongs to bin 105 to 120, which correspond to where the flame dissipates.

5.2.2 Conditional Probability Query

As described in Section 4, we can compute conditional histograms from those block-wise multivariate histograms. Based on the con-

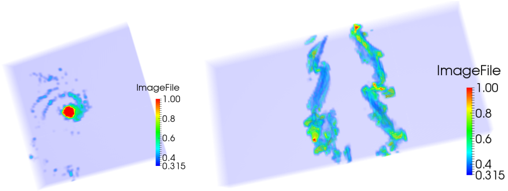


Figure 7: **Left:** $P(30 \leq bin(Cloud) \leq 255, 30 \leq bin(QRain) \leq 255) > 50\%$ OR $P(0 \leq bin(Pressure) \leq 50) > 50\%$ **Right:** $P(0 \leq bin(OH) \leq 96, 105 \leq bin(MixtureFraction) \leq 120) > 50\%$.

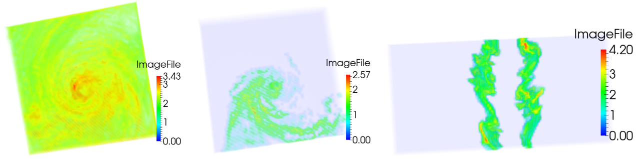


Figure 8: **Left:** The entropy field computed from $P(Temperature|0 \leq bin(QVapor) \leq 10)$ for Isabel. **Middle:** The entropy field computed from $P(Temperature|200 \leq bin(QVapor) \leq 210)$ for Isabel. **Right:** The entropy field computed from $P(OH|120 \leq bin(MixtureFraction) \leq 130)$ for Combustion.

ditional histograms, further statistical information such as information entropy and variance could be derived. For example, if Y is a set of variables and X is another variable, by computing some statistical metrics of the distribution $P(Y|min_x < X < max_x)$, we can infer the relationship between Y and X when X is in a value range $[min_x, max_x]$. For example, the entropy of $P(Y|min_x < X < max_x)$ tells us how uncertain we know about Y when X belongs to value range $[min_x, max_x]$.

Figure 8 shows the query results using Isabel and Combustion. The left image shows the conditional entropy of temperature given QVapor belongs to bin 0 to 10. Higher entropy means when QVapor belongs to bin 0 to 10, we are more uncertain about the value of temperature. Lower entropy means we are more certain about the value of temperature. Zero entropy has two meaning, one is that we are very certain about the temperature value and the other is that QVapoer does not have values within bin 0 to 10 in this region. In the middle image, we show the conditional entropy of temperature given QVapor belonging to bin 200 to 210. The combustion dataset was used in the right image. It shows the conditional entropy of OH given Mixture Fraction belonging to bin 120 to 130.

5.2.3 Conditional Fuzzy Isosurfacing

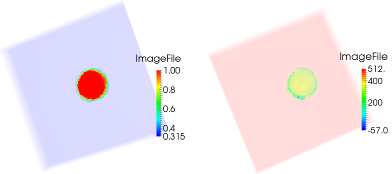


Figure 9: **Left:** $P(0 < bin(pressure) < 150) > 50\%$. **Right:** The conditional fuzzy isosurfaces computed from $P(QRain = 0.006|0 < bin(pressure) < 150)$

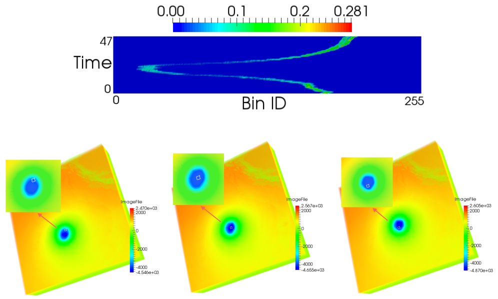


Figure 10: **Top**: Time histogram; **Bottom left to right**: Volume rendering of the pressure field for time step 17, 20 and 23. The data block from which the time histogram is computed is shown in white.

Thompson et. al. [20] proposed a technique to compute Fuzzy Isosurfaces from block-wise distributions. A likelihood value g is computed given a particular isovalue to indicate the likelihood of the existence of an isosurface for each block. Here, we extend it to multivariate data and compute the conditional fuzzy isosurfaces. Given a variable Y and another variable X, the conditional fuzzy isosurface is the fuzzy isosurface computed from distribution $P(Y|min_x < X < max_x)$. For blocks that do not have the conditional distribution $P(Y|min_x < X < max_x)$ because the variable X does not have values within $[min_x, max_x]$ in that block, we set the likelihood value g to be the total number of data points in the block which means the block is strongly above the isovalue.

The right image in Figure 9 shows the conditional fuzzy isosurface computed from Isabel. We computed the fuzzy isosurface of QRain with an isovalue 0.006 when the pressure belongs to bin 0 to 150. The left image in Figure 9 shows the query results of $P(0 < bin(pressure) < 150) > 50\%$ using the method presented in Section 5.2.1. When only the pressure is used, we get the hurricane center in the result. By using the conditional fuzzy isosurfaces to show the regions with high QRain value such as 0.006 given pressure belonging to bin 0 to 150, we get the outside of the hurricane.

5.2.4 Time Varying Data Analysis

If a time varying dataset was used, with our approach, we could easily query the time histogram [1] of different variables from arbitrary data blocks. A marginalization operation over the user selected variable V and the variable time step for the user selected block gives us the time histogram of the variable V for the selected block. The time histogram is concatenated by the 1D histograms for the variable V computed from different time steps. The time histogram computed from the block can be used to indicate the feature behavior presented in this block over time.

The dataset *Isabel* models a strong hurricane in the West Atlantic region. It has 48 times steps and captures the movement of the hurricane. In general, the low pressure area indicates the presence of hurricane eye, so the time histogram computed from pressure can be used to identify when the hurricane passes a local block. An example is shown in Figure 10. The top figure in Figure 10 shows a time histogram computed from the variable pressure for a data block from Isabel. Low pressure indicates the presence of hurricane eye. From the time histogram, we can see the hurricane passes through the data block from time step 17 to 23. The bottom three figures in Figure 10 show the volume rendering of pressure variable for time step 17, 20 and 23. The data block is shown in white color, and the hurricane passes through the data block gradually.

6 Performance

In our implementation, we divide the entire dataset into axis-aligned blocks of equal size at the finest level of detail specified by the user. Then these data blocks are assigned to parallel processes in a round-robin order and each process computes a multivariate histogram for each assigned data block. We use **boost::dynamic_bitset** to represent the bit indices. The index and frequency pairs are stored using **boost::unordered_map**.

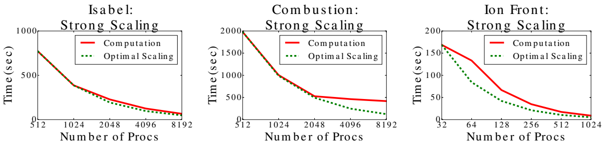


Figure 11: Strong scaling results of histogram construction.

This section presents the performance results of our technique, including the scalability of the distributed multivariate histogram computation, histogram size, and the scalability of different query operations. Three datasets were used. The dataset *Isabel* with a resolution of $500 \times 500 \times 100$ models a strong hurricane in the West Atlantic region in September 2003. This dataset contains thirteen variables and 32 time steps were used. The total size is about 38.74GB. *Combustion* is a dataset which simulates the turbulent combustion. This dataset contains five variables. The resolution of this dataset is $480 \times 720 \times 120$ and 64 time steps were used. The total size is about 49.44GB. The dataset *Ion Front* from the IEEE 2008 Visualization Design Contest [21] is a simulation of an ionization front instability. This dataset has a resolution of $600 \times 248 \times 248$. There are ten variables in this dataset and we use 1 time step for our experiment. The total size is 1407.71 MB.

We compared the sizes of multivariate histograms using our representation with two other representations. The first representation is denoted as vector representation, which stores the non-empty entries as index and frequency pairs while the index is represented as a K-tuple if there are K variables. The i_{th} element in the K-tuple represents the bin id of the i_{th} variable. The elements in the tuple are stored as unsigned short integers. The second representation denoted as *Curv.absci*. also represents the non-empty entries as index and frequency pairs but the index is represented as a single scalar value [4]. We store the scalar values as bit strings. The number of bits used depends on the maximum possible value of the index and each index has the same number of bits. For those two representation, we store the indices and frequencies separately in two files. For our representation, we stored three files, and they store the indices, dictionaries and frequencies respectively. Since the size of the frequency file is the same for all these three representations, we compared the sizes of the index files generated by the other two representations with the size of the index file plus the size of the dictionary file generated by our representation.

6.1 Construction Scalability

The multivariate histogram computation is done in parallel. We tested the scalability of this stage on the Blue Gene/Q *Vesta* system at the Argonne Leadership Computing Facility. *Vesta* contains 2,048 nodes each holding 16 PowerPc A2 1600MHz cores sharing 16GB of RAM and utilizes the General Parallel File System. The total memory is 32 TB.

We used all the three datasets in our test. The dimensions of the multivariate histogram generated from Isabel, Combustion and Ion Front are $256^{13} \times 32, 256^{5} \times 64$ and 256^{10} respectively. We set the number of partitions along x, y, and z dimensions to $64 \times 64 \times 16$ for Isabel, $64 \times 128 \times 16$ for Combustion and $64 \times 32 \times 32$ for Ion Front. The strong scaling results of the computation time including data space transformation and multivariate histogram representation computation are shown in Figure 11. As shown in the Figure, our multivariate histogram computation shows good scalability up to thousands of processes for all the three datasets.

6.2 Multivariate Histogram Size

In this section, we compared the storage cost by using our representation with the vector and *Curv.absci*. representations. The number of bins along each dimension were the same and in our experiments we tested for 16, 32, 64, 128, 256 and 512 bins for Isabel and Combustion, and 128, 256, 512, 1024, 2048 and 4096 bins for Ion Front.

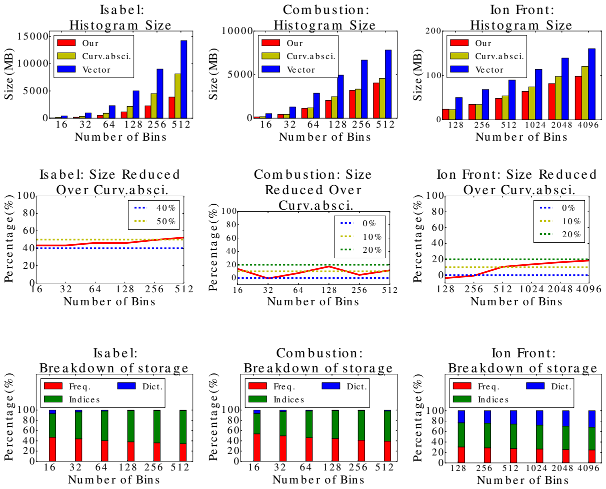


Figure 12: **Top**: Storage cost versus number of bins under different representations. **Middle**: Percentage of storage reduced with our representation over *Curv.absci*. representation. **Bottom**: The breakdown of the storage among frequencies, indices, and dictionaries.

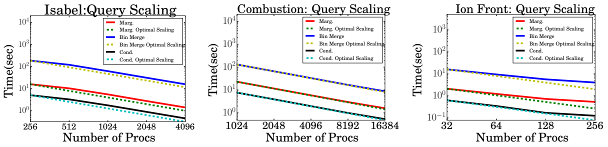


Figure 13: Strong scaling results of query.

The comparison of the storage cost for different representations is shown in the top three figures in Figure 12. The storage cost by using our representation is consistently smaller than the other two representations. This is reasonable because our representation uses fewer bits than the other two representations to represent the index. The middle three figures in Figure 12 show the percentage of storage reduced compared with the *Curv.absci*. representation. These figures indicate generally the percentage of storage cost reduced compared with the *Curv.absci*. representation increases as the number of bins increases. This is because when larger number of bins is used, the size of multivariate histogram is considerably large which make our data space transformation useful. The bottom three figures in Figure 12 show the breakdown of the storage costs among frequencies, indices, and dictionaries of our representation.

6.3 Query Performance

We implemented all the query operations presented in Section 4 in parallel. In our implementation, those histograms are distributed to the compute processes, and then each process performs the query operations on those assigned histogram in parallel. In this section, we study the scalability of the marginalization, bin merge and conditional histogram query operations. We tested the performance on *Vesta*. For the time histogram query, since it is performed only on the user specified data block, no matter how many processes are used, only the process which has the user specified data block performs this query operation. Instead of studying the scalability, we show the performance of time histogram query given a smaller number of processes. For all tests, the dimensions of the multivariate histogram used for Isabel, Combustion and Ion Front are $256^{13} \times 32, 256^5 \times 64$ and 256^{10} respectively. We measured the response time for all the four query operations described in Section 4. For each query operation, we randomly generated 10 test cases, and the reported timings are the average response time.

The scalability results of the marginalization, bin merge and conditional histogram query operations are shown in Figure 13. As the figures demonstrated, strong scalability was observed up to a large number of processes for those three query operations and different datasets. The bin merge operation requires more time than the other operations because we need decoding for the bin merge operation while the marginalization operation does not require decoding. Although the conditional histogram operation requires to decode a subset of variables, since it only performs bit-wise operations on the entries which fall into the query bin index ranges, it generally performs better than the other query operations.

For time histogram query, we use 256, 1024 and 32 processes for Isabel, Combustion and Ion Front respectively, and the time required for each dataset is 0.0046 sec, 0.0028 sec and 0.00049 sec.

7 Conclusion and Future Works

In this paper we present a novel compact structure to represent a multivariate histogram. We first use a data space transformation to transform the original large data space to a much smaller data space, and then we index the small data space to compute the representation of the multivariate histogram. By using this representation, the space cost of storing a multivariate histogram is reduced significantly. Given this new representation, we present several query operations to compute different types of histograms efficiently. We also present several visualization applications using our technique.

In the future, we plan to extend our technique along the following directions. First, we will extend our technique to handle curvilinear and unstructured grid data which are generated from many simulations. Domain decomposition and local histogram computation are more challenging due to their irregular geometric and topological structures. Second, we plan to explore other query operations.

Acknowledgements

This work was supported in part by NSF grants IIS- 1250752, IIS-1065025, and US Department of Energy grants DE- SC0007444, DE- DC0012495, program manager Lucy Nowell. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract DE-AC02-06CH11357.

References

[1] H. Akiba, N. Fout, and K.-L. Ma. Simultaneous classification of time-varying volume data based on the time histogram. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'06, pages 171–178, 2006.

[2] C. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *Visualization '97., Proceedings*, pages 167–173, Oct 1997.

[3] H. Carr, B. Duffy, and B. Denby. On histograms and isosurface statistics. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1259–1266, Sept 2006.

[4] J. Chanussot, A. Clement, B. Vigouroux, and J. Chabod. Lossless compact histogram representation for multi-component images: application to histogram equalization. In *Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International*, volume 6, pages 3940–3942, July 2003.

[5] A. Chaudhuri, T.-Y. Lee, B. Zhou, C. Wang, T. Xu, H.-W. Shen, T. Peterka, and Y.-J. Chiang. Scalable computation of distributions from large scale data sets. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 113–120, Oct 2012.

[6] A. Chaudhuri, T. H. Wei, T. Y. Lee, H. W. Shen, and T. Peterka. Efficient range distribution query for visualizing scientific data. In *Pacific Visualization Symposium (PacificVis), 2014 IEEE*, pages 201–208, March 2014.

[7] S. Goil and A. Choudhary. Sparse data storage schemes for multidimensional data for olap and data mining. *Technical Report CPDC-9801-005, Northwestern University*, 1997.

[8] L. Gosink, C. Garth, J. Anderson, E. Bethel, and K. Joy. An application of multivariate statistical analysis for query-driven visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 17(3):264–275, March 2011.

[9] Y. Gu and C. Wang. Transgraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2015–2024, Dec 2011.

[10] C. R. Johnson and J. Huang. Distribution-driven visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):734–746, Sept. 2009.

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

[12] T.-Y. Lee and H.-W. Shen. Efficient local statistical analysis via integral histograms with discrete wavelet transform. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2693–2702, Dec 2013.

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

[14] S. Martin and H.-W. Shen. Histogram spectra for multivariate time-varying volume lod selection. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 39–46, Oct 2011.

[15] S. Martin and H.-W. Shen. Transformations for volumetric range distribution queries. In *Visualization Symposium (PacificVis), 2013 IEEE Pacific*, pages 89–96, Feb 2013.

[16] F. Porikli. Integral histogram: a fast way to extract histograms in cartesian spaces. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 829–836 vol. 1, June 2005.

[17] O. Rübel, Prabhat, K. Wu, H. Childs, J. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. Weber, P. Messmer, H. Hagen, B. Hamann, and E. W. Bethel. High performance multivariate visual data exploration for extremely large data. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 51:1–51:12, Piscataway, NJ, USA, 2008. IEEE Press.

[18] K. Stockinger, E. Bethel, S. Campbell, E. Dart, and K. Wu. Detecting distributed scans using high-performance query-driven visualization. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 39–39, Nov 2006.

[19] K. Stockinger, J. Shalf, K. Wu, and E. Bethel. Query-driven visualization of large data sets. In *Visualization, 2005. VIS 05. IEEE*, pages 167–174, Oct 2005.

[20] D. Thompson, J. Levine, J. Bennett, P.-T. Bremer, A. Gyulassy, V. Pascucci, and P. Pebay. Analysis of large-scale scalar data using hixels. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 23–30, Oct 2011.

[21] D. Whalen and M. L. Norman. Competition data set and description. In *2008 IEEE Visualization Design Contest*, 2008.

[22] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. G. R. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Laurent, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rübel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang. FastBit: Interactively Searching Massive Data. *Journal of Physics Conference Series, Proceedings of SciDAC 2009*, 180:012053, June 2009. LBNL-2164E.

[23] K. Wu, W. Koegler, J. Chen, and A. Shoshani. Using bitmap index for interactive exploration of large datasets. In *Scientific and Statistical Database Management, 2003. 15th International Conference on*, pages 65–74, July 2003.

[24] L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1216–1224, Nov 2010.