Eurographics Conference on Visualization (EuroVis) 2015
H. Carr, K.-L. Ma, and G. Santucci
(Guest Editors)

Volume 34 (2015), Number 3

Efficient Local Histogram Searching via Bitmap Indexing

Tzu-Hsuan Wei, Chun-Ming Chen & Ayan Biswas †

The Ohio State University, Columbus OH, USA

Abstract

Representing features by local histograms is a proven technique in several volume analysis and visualization applications including feature tracking and transfer function design. The efficiency of these applications, however, is hampered by the high computational complexity of local histogram computation and matching. In this paper, we propose a novel algorithm to accelerate local histogram search by leveraging bitmap indexing. Our method avoids exhaustive searching of all voxels in the spatial domain by examining only the voxels whose values fall within the value range of user-defined local features and their neighborhood. Based on the idea that the value range of local features is in general much smaller than the dynamic range of the entire dataset, we propose a local voting scheme to construct the local histograms so that only a small number of voxels need to be examined. Experimental results show that our method can reduce much computational workload compared to the conventional approaches. To demonstrate the utility of our method, an interactive interface was developed to assist users in defining target features as local histograms and identify the locations of these features in the dataset.

1. Introduction

Representing features by local histograms has been utilized in many volume analysis and visualization applications. It has been shown that histograms in local regions can be used to identify material boundaries [TLB*11] or to predict material composition in a local area [LFB98]. Given a region of user's interest, the local histogram of that region is widely used to extract other regions of the(delete) similar characteristic. In data visualization, the technique of local histogram matching and searching can also benefit transfer function design [LLY06] and feature tracking in time-varying data [GW11]. The computational performance of local histogram searching, however, has not drawn much attention in these works. In general, to find the local features that match a user-defined local histogram requires first computation of a histogram from each local region in the data domain. This time-consuming exhaustive search prohibits the aforementioned applications from being practical in large data cases. Recently several algorithms have been proposed to support efficient local histogram generation. For example, the integral histogram proposed by Porikli [Por05] is able to retrieve the histogram from an arbitrary axis-aligned region in real time. However, the tremendous storage overhead to store the integral histograms still prohibits the applications for large or high dimensional dataset. With much less memory requirements, Sizintsev et al. [SDH08] proposed the distributive histogram, which reduces raw data accesses by only updating the histogram in non-overlapping regions when scanning through the data domain for histogram searching. Although significant performance speedup can be achieved compared to the conventional exhaustive search method, their method still requires scanning through the entire data domain for each query, which is a concern when the size of the data is large.

Recently bitmap indexing, designed to efficiently locate data that matches user-interested values, have been adopted in multiple fields. Bitmap indexing is a table-like data structure where the rows represent the data points and the columns represent distinct value ranges of the data. Using 0's and 1's to indicate whether the value at a data point falls into the corresponding value range and employing run-length encoding to compress each bit-vector [WOS02], bitmap indexing can achieve quick range query with small storage overhead.

In this paper, we present an efficient algorithm utilizing bitmap indexing to locate voxels whose local histograms in their neighborhood match the user defined target histogram.

† {wei.225, chen.1701, biswas.36}@osu.edu

From our observation, the value range defined in the user-specified target histogram is usually a subset of the entire data value range. We also observed that only those voxels whose neighboring points have values within the chosen value range can be the possible candidates that satisfy the search. Therefore, for local histogram searching we propose an algorithm that only scans through those voxels and their neighborhood regions so that the search space can be significantly reduced. To do this, we leverage bitmap indexing to quickly locate data points whose values fall into the target histogram bins. With these voxels identified, we propose a voting scheme to construct local frequency counts in each of the target histogram bins. By iteratively processing each bin, we gradually reduce the search space by excluding unqualified voxels whose local frequency counts do not match the target histogram, and thus much higher performance can be achieved for local histogram searching. Experiments show that our algorithm is faster than Sizintsev et al.'s method in most test cases.

The contributions of this paper are threefold:

1. We propose a novel local histogram searching algorithm which only investigates regions containing voxels in an user interested value range and their neighborhood regions without exhaustively scanning the entire data.
2. We utilize bitmap indexing to quickly locate voxels in the user interested value range, which are used to construct local histograms by our local voting scheme.
3. With our algorithm that generally generates the results within seconds, we provide a system that can quickly respond to local histogram queries for 3D data.

2. Related Work

2.1. Applications of Local Histograms

The usage of features represented by the histogram of local neighborhoods has been applied in diverse fields such as computer vision, medical image visualization, scientific data visualization and query-driven visualization. Applications in the field of computer vision include object detection [EM95] and tracking [KP04], face detection [RM06], and human detection [DT05] and tracking [Bir98]. For data visualization, Laidlaw et al. [LFB98] proposed the partial-volume classification to identify the material composition in a local region. Lundström et al. [LLY06] proposed the partial range histogram (PRH) to identify the value range of the interested material and then applied the PRH to perform voxel classification. The classification result is then integrated into a transfer function design to provide detailed information in user-interested local regions. Gu et al. utilize a block level histogram to track features in time-varying data [GW11]. Johnson and Huang [JH09] proposed a feature detection approach for multifield data by studying local frequency distribution. Thompson et al. [TLB*11] proposed the use of the hixel, which stores the histogram per voxel or per block and applies the local histogram to compute fuzzy isosurfaces to identify the locations of material boundaries.

2.2. Local Histogram Searching

Since the conventional local histogram searching algorithm is to compute and compare local histograms in the neighborhood of every voxel in the dataset, whose computational cost is prohibitive in either of the cases of large data, large neighborhood size or a histogram in high resolutions, several methods have been proposed to improve the performance of local histogram searching. Porikli [Por05] proposed integral histogram, an extension of the summed area table, which extracts a local histogram in constant time so that it has been applied in many works such as object tracking [ARS06]. Due to the requirement of expensive storage overhead for integral histograms, several techniques have been adopted. Lee et al. [LS13] proposed a wavelet based compression technique for integral histograms. Chaudhuri et al. [CWL*14] utilized the decomposition scheme and similarity-driven indexing to reduce the storage cost. However, their storage overhead can still be larger than the original data size. Perreault and Hebert [PH07] proposed only to scan through non-overlapping regions and update the histogram from the previous query region by maintaining column histograms. Based on their method, Sizintsev et al. [SDH08] proposed the distributive histogram [HYT79] for local histogram searching, which can achieve higher performance than the conventional algorithm in 2D cases. However, it still has to scan through all the voxels in the data, which becomes costly in large 3D volume datasets.

3. Bitmap Indexing

Bitmap indexing is an efficient indexing data structure which takes advantage of bitwise operations supported by the computer hardware. It is able to quickly respond to the value range query and has been applied frequently in query-based visualization applications [CHA*11] [RBPW12] [SAW12]. By using bitmap indexing, our work can quickly extract voxels whose values fall within the user interest value range and thus accelerate the searching process. Below we briefly introduce the concept of bitmap indexing.

We provide an example in Table 1 to explain bitmap indexing. In this simple example, we divide the whole data value range into smaller groups and assign each group to a corresponding bin, which is similar in concept to histograms. In table 1, the dataset has 8 elements and the data value range is from 0.0 to 4.0. Here the bin width was set to 1 so that 4 distinct value ranges were created. Each column of this bitmap represents one value range and is stored in a *bit vector* whose length is the same as the total number of data elements. In each bit vector, a bit is set to 1 if the value of the corresponding voxel falls into the value range; otherwise it

Table 1: *An example of the bitmap indexing.*

Dataset		Bitmap Indexing			
Voxel ID	Data Value	b_0 [0.0,1.0)	b_1 [1.0,2.0)	b_2 [2.0,3.0)	b_3 [3.0,4.0]
0	0.2	1	0	0	0
1	1.1	0	1	0	0
2	2.4	0	0	1	0
3	2.7	0	0	1	0
4	3.8	0	0	0	1
5	2.2	0	0	1	0
6	1.6	0	1	0	0
7	0.7	1	0	0	0

is set to 0. Equation 1 formulates the bit assignment for a bitmap which will be used in the method section.

$$bmap_{ik} = \begin{cases} 1 & v_i \in b_k, \\ 0 & v_i \notin b_k, \end{cases} \quad (1)$$

In this equation, $bmap_{ik}$ represents the bit at the position (i,k) in the bitmap, b_k represents the bin k, and the v_i denotes the value of the ith voxel.

As the size of scientific datasets continues to grow, compressed bitmap indexing is becoming a popular method to reduce the data size overhead. Multiple compressed bitmap indexes based on the run-length encoding scheme have been proposed, such as Byte-aligned Bitmap Code (BBC) [Ant95] and Word-Aligned Hybrid (WAH) [WOS04] [WOS02]. The WAH compression is used in our method since it achieves better performance than most of the other compressed bitmap indexing algorithms [WAB*09]. Since a series of 0s or 1s is frequently encountered in the bit vector, WAH only stores the counts of continuous 0s or 1s. Since the length of the vector describing this encoding count is much smaller than the length of the bit vector, the size of the bitmap and the time to access a bit can be reduced. In terms of query response time, our approach benefits from the compressed bitmap in two ways. First, it reduces the I/O time by loading shorter vectors. Second, the time to find the bit that is assigned to 1 in a bit vector becomes faster when decoding a shorter vector.

4. Problem Statement and Algorithm Overview

4.1. Problem Statement

The exploration of a volumetric dataset usually requires detailed inspection of local features, which can often be described by local histograms. A local histogram is the histogram collected from the values in a voxel's local neighborhood. The goal of this work is to efficiently locate the voxels whose local histograms match the user desired local histograms without exhaustively searching in the entire volume. To do this, the user defines the *target histogram* H_T, the neighborhood size *NBR*, and a tolerance factor δ of the difference between the target histogram and the local histograms. To compare two histograms, in this work we define two types of histogram comparison metrics to detect

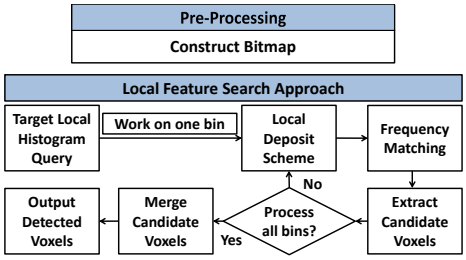


Figure 1: *The flow chart of the proposed local histogram search approach*

voxels whose local histogram matches H_T. The first type is a strict comparison in which we reject an input histogram if the difference of the frequency of any bin k within the user-interested value range is larger than δ_{b_k}, which we call *single-bin-error comparison*. We formulate this comparison in Equation 2, where the detected voxels are defined as *voxels of interest*, or *VOI*:

$$VOI \equiv \{v_i \mid abs(H_{v_i}(b_k) - H_T(b_k)) \leq \delta_{b_k}, \\ \forall i \in \{1, \cdots n\}, \forall b_k \in \text{user-interested value range}\} \quad (2)$$

Here, H_{v_i} represents the local histogram of voxel v_i. n is the total number of voxels in the volume, b_k represents one of the bins in the user interested value range, and $H(b_k)$ is the frequency of histogram H at bin b_k.

The second type of histogram comparison metric we use is called *sum-of-error comparison*, where we reject a local histogram if the sum of the errors of the bin frequencies is larger than the total error threshold δ_{sum}, as formulated in Equation 3:

$$VOI \equiv \{v_i \mid \sum_{b_k} f_{err}(H_{v_i}(b_k), H_T(b_k)) \leq \delta_{sum}, \\ \forall i \in \{1, \cdots n\}, b_k \in \text{user-interested value range}\} \quad (3)$$

Here $f_{err}(H_{v_i}(b_k), H_T(b_k))$ represents a bin-to-bin error between two distributions H_{v_i} and H_T. In this comparison type several known error metrics can be applied, such as $L_1 - norm$, $L_2 - norm$, χ^2 statistics and Kullback-Leibler divergence (or Jeffery divergence) [MGW10]. For example, if we apply the $L_1 - norm$, the error metric $f_{err}(\cdot)$ becomes

$$f_{err}(H_{v_i}(b_k), H_T(b_k)) \equiv abs(H_{v_i}(b_k) - H_T(b_k)) \quad (4)$$

4.2. Algorithm Overview

In this work, we propose an efficient algorithm that uses bitmap indexing to tackle the local feature searching problem by matching local distributions. The framework of our method is shown in Fig. 1. Rather than searching for the entire volume and detecting voxels that have similar local histogram to the target histogram, our method is to only search for the voxels within the value range of user-defined local

features and their neighborhood. Within this search space, we perform the histogram matching algorithm based on bin-wise comparison with the single-bin-error or the sum-of-error metric. For one single bin, a local deposit scheme is proposed to determine the bin frequency for each voxel. The local deposit is similar to a voting process which is applied to collect the corresponding value contribution from the local neighborhood of each voxel. We retrieved the voxels whose values fall into the bins of interest from the compressed bitmap and then perform the local deposit on their neighborhood voxels. After performing the local deposit, the bin frequency for each voxel, $H_{v_i}(b_k)$, can be determined according to its deposit count. The distance between $H_{v_i}(b_k)$ and $H_T(b_k)$ is computed by checking how they are matched. Finally, we collect the final detected voxels *VOI* based on the error metric we use.

5. Proposed Method

In this section, we describe our proposed algorithm to efficiently search for the target local histogram.

5.1. Local Histogram Searching Algorithm

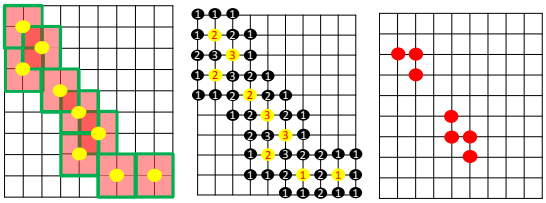
While searching for a user-specified histogram in a dataset, the conventional approach is to go through all the voxels in the dataset; create the local histogram around the voxels and finally compare each of these local histograms with the user specified target histogram. This exhaustive search quickly becomes infeasible when the dataset is too large. In this simple search method, we make two key observations:

1. When the user specifies a target histogram, generally this target histogram has a smaller dynamic range compared to the value range present in the whole dataset.
2. The construction of a local histogram can be thought of inversely; instead of collecting data from a voxel's neighborhood to construct a local neighborhood, each data element can contribute its value to the voxels in its neighborhood so that those voxels' local histograms can be constructed.

Based on these two observations, we formulate a novel algorithm to identify voxels whose local histogram matches that specified by the users. Below we elaborate our algorithm in detail.

5.2. Histogram Matching Based on Bin Comparison

When the user specifies a target histogram to be searched in the dataset, the interested bins and their corresponding frequencies are provided. The interested bins provided by the user can be a subset of the bins presented in the histogram created from the entire dataset. These interested bins, defined as *bins of interest*, or *BOI*, are only to be considered during the histogram matching. The voxels whose values fall into the BOI are the contributors to the local histograms of



(a) (b) (c)

Figure 2: *Example of local deposit scheme for one single bin: (a): Yellow points are active voxels of the bin. Each green square region is the neighborhood of one active voxel, and the neighborhood size in this example is* 3×3*. Red region is the search space. (b): The number on each voxel in the search space represents deposit count which refers to frequency of the bin on the voxel. Yellow points represent active voxels and Black points represent the rest of voxels in the search region. (c): In this example, a voxel whose frequency is equal to 3 is defined as a candidate voxel. All the candidate voxels for the bin are shown as red points.*

their neighborhood voxels. From our observation, only these voxels will affect the result of local histogram matching and we define them as *active voxels*. That is,

$$v_{active}^{b_k} \equiv \{v_i \mid bmap_{ik} = 1 \wedge b_k \in \{\mathrm{BOI}\}, \quad \forall i \in \{1, \cdots n\}\} \quad (5)$$

Since bin-wise comparison is performed during histogram matching, the knowledge of bins of user-interest and the corresponding active voxels can now be utilized to reduce the computation time of the histogram matching. Therefore, instead of exhaustively searching in the entire domain, we can reduce our search space to the active voxels and their neighborhood. Fig. 2 (b) shows an example of the search space. The yellow points represent active voxels of bin 10 of the target histogram in figure (a). Each green square region is the neighborhood of one active voxel and the red region is our reduced search space. This reduction in the number of voxels to be processed provides a significant decrease in computation time.

Given a user specified target histogram, quickly locating the active voxels is a challenging task. With the use of the compressed bitmap indices representing the original dataset, however, this can be achieved in a much shorter time along with moderately low storage overhead. When the dataset is represented through bitmap indexing, the bit vectors corresponding to the BOI can be extracted and used to quickly locate the physical locations of the active voxels. After locating the active voxels, the next stage is to construct the local histogram to perform matching which is discussed in the next section.

5.3. Local Deposit

To generate the local histogram for a voxel, an obvious strategy is to find all the neighborhood voxels around it and form a histogram. But creation of this local histogram can be approached differently. Given a data element and a neighborhood size, we can locate all its neighboring voxels and take a contribution from this data element. The data value of this data element increases the frequency count of the corresponding bin of all these neighborhood voxels' histograms. Intuitively, this is similar to a voting process described in the Hough Transform [Hou59]. Since every data element contributes to its neighbors' local histograms, after all the data elements are done voting, each voxel will be left with counts, and hence the local histogram, from its neighbors. We call this voting scheme as local deposit and apply it on the active voxels only which are extracted in the first stage as described in Section 5.2. This way we can first reduce the number of voxels to work with and then utilize these voxels to create the local histograms.

In our algorithm, we perform the local deposit scheme for each bin in the BOI separately. While performing the deposit scheme for a single bin, we allocate a *deposit buffer* where each element of the buffer corresponds to a voxel in the dataset. During the deposit procedure, for each active voxel we locate its neighborhood voxels and at each voxel location we increment the count of the corresponding element in the deposit buffer. After all active voxels for one single bin are processed, the accumulated count at each location in the deposit buffer will essentially be the frequency for this bin in the local neighborhood of the corresponding voxel. This procedure for one single bin is explained schematically in Figure 2. By the local deposit scheme, the frequency for each voxel can be retrieved and is defined as:

$$H_{v_i}(b_k) = \left\| v^{b_k}_{active} \in NBR(v_i) \right\|, i = 1...n. \quad (6)$$

Here, $NBR(v_i)$ represents the neighborhood of voxel v_i. The computed $H_{v_i}(b_k)$ is later used to match the target histogram H_T at bin b_k, as will be described in the next section.

5.4. Frequency Matching and Collection of VOI

In this section, we introduce how the deposit count is utilized to detect which voxels contain the target histogram H_T. During the local deposit stage for a single bin b_k, we obtain the deposit count of bin b_k for each voxel v_i, which is equal to the bin frequency $H_{v_i}(b_k)$. By comparing with the target frequency $H_T(b_k)$, we can compute the bin-wise error based on the error metric the user selects as described in Section 4.1. To determine the final result of the VOI, we accommodate our algorithm for two types of histogram comparison metrics as follows.

Single-bin-error comparison: For the single-bin-error comparison, all the voxels that match within the error tolerance are labeled as *candidate voxels*. We define the candidate voxels for bin k as CDV^{b_k} in 7:

$$CDV^{b_k} \equiv \{v_i \mid abs(H_{v_i}(b_k) - H_T(b_k)) \le \delta_{b_k}, \forall i \in \{1,\cdots n\}, b_k \in BOI\} \quad (7)$$

Here, δ_{b_k} is the user-defined error tolerance for bin k. In Fig. 2, suppose the user-assigned frequency in bin 10 is greater than 2, the candidate voxels for bin 10 are the red points shown in Fig. 2 (c). If all bins of interest have been processed, the intersection of the candidate voxels from all bins of interest are the final result VOI (Equation 8) which is given as:

$$VOI = \bigcap_{b_k \in BOI} CDV^{b_k} \quad (8)$$

The step-by-step search algorithm is shown in Algorithm 1.

Sum-of-error comparison: For the sum-of-error metric scheme, we use the same algorithm as single-bin-error comparison, but record the accumulated errors computed from each bin for all voxels that have been deposited by any active voxels. Then we use the final accumulated errors to compare with the total error threshold δ_{sum} after processing all bins in the BOI. If a voxel's deposit count is zero, meaning that the frequencies of the user-interested bins for this voxel are all zero, its sum of errors is simply computed from the tar-

Algorithm 1 Optimized local historam searching algorithm for single-bin-error

1: Let db_{v_i} be the deposit buffer at voxel v_i and initialize to 0, $db_{NBR(v_i)}$ represents the deposit buffer at neighborhood voxels of v_i including itself. Let E_{v_i} be the error accumulator at voxel v_i and initialize to 0
2: Let $\{b_{k1},\cdots,b_{kp}\}$ be the sequence of processing bin sorted by $\|v^{b_k}_{active}\|$ in increasing order. $b_k \in$ bins of interest, p is the number of bins of interest
3: **for** $i = 1$ to p **do**
4: **if** $i == 1$ **then**
5: $v_{temp} \leftarrow v^{b_{k1}}_{active}$
6: **else**
7: $v_{temp} \leftarrow v^{b_{ki}}_{active} \bigcap NBR(CDV^{b_{k(i-1)}})$
8: **end if**
9: **for** $j = 1$ to $\|v_{temp}\|$ **do**
10: $db_{NBR(v^i_{temp})} \leftarrow db_{NBR(v^i_{temp})} + 1$
11: **end for**
12: **for** each $v_x \in NBR(v_{temp})$ **do**
13: **if** $abs(H_{v_x}(b_{ki}) - H_T(b_{ki})) \le \delta$ **then**
14: $CDV^{b_{kj}} \leftarrow v_x$
15: $E_{v_x} = E_{v_x} + abs(H_{v_x}(b_{ki}) - H_T(b_{ki}))$
16: **end if**
17: **end for**
18: **end for**
19: **return** $CDV^{b_{kp}}$

get frequencies in the BOI. By doing so, we can save storage space and time for updating errors of non-deposited voxels. After processing all bins in the BOI, we scan through the voxels with accumulated errors, and output the *VOI* that are within the user-specified error threshold δ_{sum}.

5.5. Local Deposit Workload Reduction for Single-bin-error Comparison

If the single-bin-error comparison scheme is used, we can apply two strategies to reduce the local deposit workload in our algorithm. The first one is called outcome transition, in which after processing each bin, we pass the resulting candidate voxels to the next bin. If a voxel is rejected while processing any bin, it will never become a member of the VOI. As a result, we can skip those voxels and only deposit on the candidate voxel locations while performing the local deposit scheme. The second workload reduction strategy is called ordered-processing. By utilizing the first strategy, the number of candidate voxels in general will decrease after each bin is processed so that the workload of local deposit for the later bins will also be reduced. From another observation, if there is a smaller number of active voxels in one bin, the number of outcomes (candidate voxels) will also likely be smaller. Therefore, we first sort the bins in the BOI according to the number of active voxels associated to each bin in the increasing order, and then perform the local deposit for the bins according to the order. By applying ordered-processing, fewer candidate voxels are expected to be left after processing the first few bins. Consequently, less workload is needed for the later bins that have larger number of active voxels. The numbers of active voxels for all bins are pre-computed and stored along with the bitmaps.

6. Results

In this section we compare the performance of our method with an existing local histogram searching algorithm for both error metrics. Then we show case studies using our local histogram searching system.

6.1. Performance

The tests were conducted on a machine with Intel Core 2 Duo E6750 CPU, 8GB system memory, and an nVidia GeForce GTX 460 GPU with 1GB of texture memory. Three datasets of different resolutions were used: *Isabel* is the pressure field of Hurricane Isabel from theVis'04 Contest; *Combustion* is the mixture fraction field of the combustion phenomenon provided by the Sandia National Laboratories; and *Plume* is the Solar Plume simulation for thermal downflow plumes on the surface layer of the Sun, where the scalar field in use was the velocity magnitude, provided by the National Center for Atmospheric Research. The detailed data sizes and the storage sizes for the compressed bitmap indices are listed in Table 2. The number of bins used for bitmap indexing was 256 for all datasets.

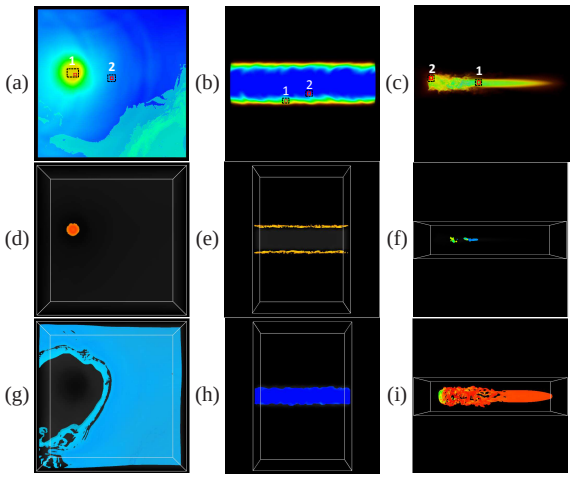


Figure 3: *The search result images for the Table 2. Black boxes in (a)-(c) are the two regions that we used to define our target histogram. Red box are used to decide the neighborhood size. (d)-(f) are three cases with smaller features. (g)-(i) are three cases with relative larger features.*

6.1.1. Performance Evaluation and Comparison

To evaluate our algorithm, we compare the performance between our method and the work by Sizintsev et al. [SDH08]. Sizintsev et al.'s method utilizes spatial coherence to provide more efficient computation times compared to most of the existing local histogram searching algorithms. We extended their method of processing 2D images to 3D volume data to suit our needs in the experiments. Besides, we also only consider bins in the BOI while comparing a local histogram with the target histogram. Therefore, the timing for their method is also affected by the number of bins in the BOI.

Two test cases to find different types of features were used for each dataset. In the first case, we tried to match some smaller features, which include the hurricane eye in the *Isabel* dataset, the mixture mass of fuel and oxidizer in the *combustion* dataset, and the turbulent region in the *Plume* dataset, as shown in region 1 in Fig. 3 (a)-(c). The resulting *VOI* of each dataset are shown in Fig. 3 (d)-(f). In addition, we also tested our algorithm in one more case for each dataset where the resulting *VOI* contains more voxels compared to the previous cases. The cases include the ocean in the *Isabel* dataset, the pure fuel mass in the *combustion* dataset, and the small velocity region in the *Plume* dataset and the resulting *VOI* are shown in Fig. 3 (g)-(i). To measure the performance of each case, we gathered several target histograms in the region near the features (shown as the black box in Fig. 3 (a)-(c)), and averaged the collected timings.

The neighborhood size used in the experiments shown in the Table 2 is 11^3. For the smaller feature cases, the number of bins in the BOI are 56, 56 and 64 for *Isabel*, *Combustion* and *Plume*, respectively. For the larger feature cases,

Table 2: *Three different test datasets in this paper. The data size and the bitmap size for each test dataset are also shown.*

Dataset	Resolution	Datasize	Bitmapsize	Test Cases	Computation times (s)				Memory Used for Bitmap Indexing	% voxels Contributed to *VOI*
					Single-Bin-Error		Sum-of-Error			
					Ours	Sizintsev	Ours	Sizintsev		
Isabel	$500 \times 500 \times 100$	95.3 MB	35.8 MB	Hurricane Eye	0.273	46.52	0.445	55.45	0.254 MB	0.039%
				Ocean	17.99	51.85	24.40	56.99	26.09 MB	2.2%
Combustion	$480 \times 720 \times 120$	158 MB	35.1 MB	mixture mass	9.481	75.47	23.39	96.98	7.671 MB	0.18%
				Pure fuel	18.38	92.53	30.25	96.76	33.72 MB	5.45%
Plume	$504 \times 504 \times 2048$	1930 MB	807 MB	turbulent region	8.051	993.1	29.31	1119.8	11.21 MB	0.016%
				small velocity region	72.42	1257.5	140.91	1240.1	86.41 MB	0.61%

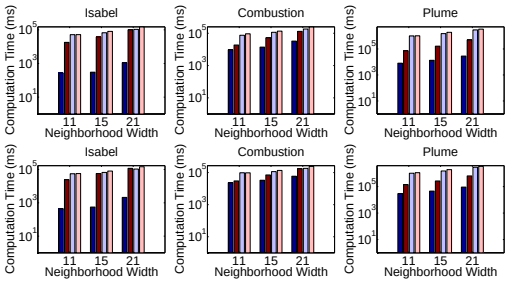


Figure 4: *The performance comparison of using different neighborhood sizes for different search cases in each dataset with the timing of Sizintsev's method. The performance in the upper row is with the single-bin-error and that in the bottom row is with the sum-of-error metric. Blue colors: Cases of Fig. 3 (d)-(f). Red colors: Cases of Fig. 3 (g)-(i). Dark color: our experiment. Light color: Sizintsev's method. Note the logarithmic scale was used for the vertical axes*

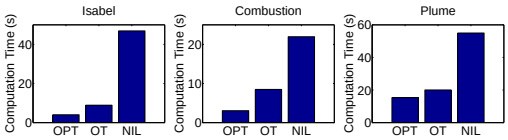


Figure 5: *Comparison of the computation times of different workload reduction schemes for the single-bin-error comparison. The first bar (OPT) shows the test with both workload reduction scheme introduced in section 5.5. The second bar (OT) is the test with outcome transition scheme but without reordering the processing bins according to the number of active voxels. The third bar (NIL) is without both workload reduction schemes .*

the number of bins in the BOI are 24, 13 and 8, respectively. In these two cases, we only counted the number of bins that have nonzero frequencies in the BOI. The features in the second test cases are within small value ranges since they are more likely to be the background, which has few value changes but locates in many regions. As a result, the second test cases work with smaller number of bins, but the larger total number of active voxels makes the search process slower than the first test cases. Since different target histograms are selected for each case, the error threshold should also be changed accordingly in order to find meaningful regions.

The computation times of single-bin-error and sum-of-error comparison, and the associated total memory usage due to bitmap indexing are listed in Table 2. As shown in Table 2, for both histogram comparison methods, the computation times in the cases with larger target features are generally higher than that in the cases with smaller target features. This is mainly because the local deposit for each bin needs to process more active voxels. We also compare the performance of histogram searching using different neighborhood sizes. Fig. 4 shows the results with neighborhood sizes 11^3, 15^3 and 21^3 in different search cases as shown in Fig. 3.

6.1.2. Evaluation of Local Deposit Workload Reduction

To show the impact of the workload reduction scheme introduced in the section 5.5 for the single-bin-error comparison, we tested three different implementations of local deposit computation including our proposed optimized algorithm, the algorithm without bin re-ordering, and the algorithm without outcome transition. In the experiment, we fixed the input target histogram and parameter settings such as neighborhood size (11^3) and error tolerance (0.001) for all bins in the BOI. As shown in Fig. 5, the implementation with the optimization with both outcome transition and bin reordering has a much smaller computation time. This shows that the workload reduction scheme is very useful in our local histogram searching algorithm when single-bin-error is used.

6.1.3. Algorithm Parameters Study

Besides the neighborhood size, the performance of our algorithm is affected by two other factors: the number of user-interested bins and the error tolerance. The number of user-interested bins, or *BOI*, will affect the number of active voxels to be processed during the computation; the error tolerance will affect the number of local histograms that match the target histogram, which is equal to the size of the *VOI*. While these factors may not affect the performance of other approaches that scan through the entire dataset, our method can obtain different speed-up depending on these two factors. Therefore, we conducted two experiments to examine these algorithm parameters.

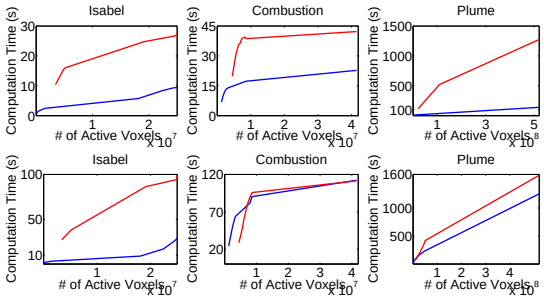


Figure 6: *Performance test using different numbers of active voxels in each dataset. Upper row: Single-bin-error comparison metric. Bottom row: Sum-of-error comparison metric. Blue line corresponds to the cases of Fig. 3 (d)-(f) and red line corresponds to the cases of Fig. 3 (g)-(i).*

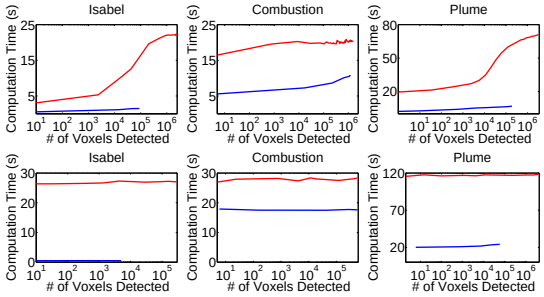


Figure 7: *Performance test using different numbers of detected voxels which match the target histogram in each dataset. Upper row: Single-bin-error comparison metric. Bottom row: Sum-of-error comparison metric. Blue line: cases for Fig. 3 (d)-(f); Red line: cases for Fig. 3 (g)-(i).*

To examine how the total number of active voxels affects our algorithm performance, we tested different cases with different number of bins in the BOI. In general, a large number of bins in the target histogram infers a large number of active voxels to process. Therefore, we plot the relations between the computation time and the number of active voxels due to the change of the BOI, as shown in Fig. 6. The test result shows that the computation time increases when the total number of active voxels becomes larger. This is because, if the user-defined target histogram contains the value range covering more bins, our algorithm generally has to process more active voxels in the process of local deposit.

For the single-bin-error scenario, the other important factor of performance in our algorithm is the size of the *VOI*, which is related to the number of candidate voxels generated in each bin. More specifically, if the user-defined feature is found in many locations, more voxels have to be processed in the local deposit stage. Consequently, the search space cannot be reduced much by our workload reduction scheme. Images in the upper row of Fig. 7 show the increasing trend of the computation time as the number of voxels found increases. For the sum-of-error comparison, we keep all the voxels deposited by the active voxels until the end and check the final results in the last stage, so the computation time is not affected by the size of the *VOI*, as shown in the bottom row of Fig. 7.

6.2. Rendering of Fuzzy Local Histogram Search Results

We have developed a graphic user interface to assist user in determining target histograms and also provided the volume rendering for the final result returned by our system. The regions obtained from the query result (or *VOI*) are colored by a 1D transfer function that maps scalar values to RGB values. We utilize the similarity value calculated on each voxel of the *VOI* to determine the opacity of each voxel. Higher opacity depicts higher certainty or belief in the search result and lower opacity represents higher uncertainty. Under this setting, users can visualize how well the target histogram matches a local region. The voxels which fall outside the users' query regions, are rendered in gray scale to preserve the context of the dataset as the user is focused on the particular feature. To display the rendering result from our fuzzy local histogram search and visually validate our search algorithm, we present two case studies from two datasets: *Isabel* and *Combustion* dataset.

6.2.1. Hurricane Dataset

Fig. 8 (a) - (e) show examples of our rendering using the pressure field of the hurricane Isabel dataset. This dataset contains 48 time steps and the grid size for each time step is $500 \times 500 \times 100$. Fig.8 (a) shows a volume rendering image of the first time step and we select the region around the hurricane eye as our target using the user interface. The red box represents our selected region (9^3 voxels) and the corresponding local histogram is shown beside that figure. In this experiment, we assume the pressure value near the hurricane eye does not vary too much over time and thus, we intend to capture the hurricane eye in the future time steps. The search results for time step 10, 20, 30 and 40 are shown in Fig. 8 (b) - (e) and the detected voxels are colored by yellow. The gray regions are shown as the context so that users can visualize the relative position of the hurricane eye and the land. From the rendering result in other time steps, the region around the hurricane eye is detected in each time step and the movement of the hurricane eye also can be seen. Based on the assumption that data values of features do not vary too much in different time steps, our algorithm can be applied for feature tracking in time-varying dataset by utilizing the local histogram search algorithm.

6.2.2. Combustion Dataset

Next, we apply our algorithm to the turbulent combustion simulation dataset. Among the five variables present in the dataset, Mixture Fraction is used in our experiment. Mixture fraction is an important variable in the dataset that describes the proportion of fuel and oxidizer mass [AMCH07].

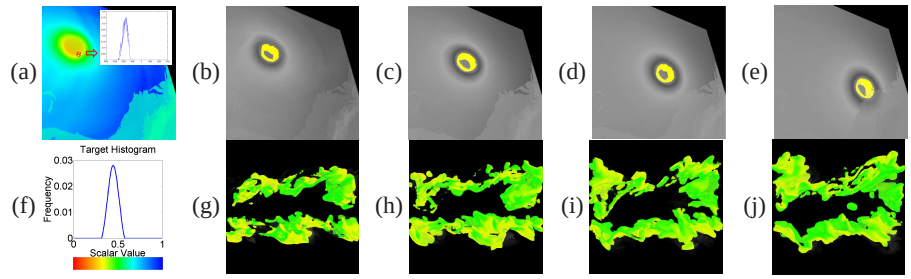


Figure 8: *Two case studies are shown for validation of our local histogram searching algorithm. (a)-(e) are for the Hurricane Isabel dataset. (a): The volume rendering of the dataset at first time step. (b)-(e): the rendering of searching results in time steps 10, 20, 30 and 40, respectively. (f)-(j) are for the combustion dataset. (f): the target histogram with value range* $[0.3, 0.55]$. *(g)-(j): the rendering of search results in time step 60, 70, 80 and 90 respectively.*

Generally, this variable provides the location of the flame in regions where its value is 0.42 although due to complex interactions in the fluid, the flame can deviate from this theoretical value. Now, we describe our target histogram to be a Gaussian like distribution with mean at 0.42 and the value range of this target histogram varying between 0.3 and 0.55 on both sides of the mean. The neighborhood size is set up as 15^3 for the experiment. Image in the second row of Fig. 8 shows the target histograms (f) and the detected results in time step 60, 70, 80, 90 are shown in Figures 8 (g)-(j), respectively. From the rendering results, the user can visualize the positional variation of the flame in different time steps.

7. Discussion

In this section, we discuss about the scope and limitations of our proposed algorithm. In this work, we apply two types of distribution comparison methods: single-bin-error and sum-of-error. In general, the performances for cases with single-bin-error are better than sum-of-error metric in our work. For the single-bin-error comparison, after processing one bin, some voxels will be discarded if their frequencies do not match the bin frequencies in the target histogram. Therefore, search region based on the candidate voxels shrinks after processing of each bin, resulting in higher speed-up than traditional methods. For the sum-of-error comparison, in order to compute the total error, we need to store voxels that have been deposited from the active voxels and their accumulated errors until the last bin in the BOI is processed. Therefore we do not achieve as much speed-up in this case. However, as shown in Table 2, our algorithm still performs better than Sizintsev *et al.*'s method in these test cases. Another factor that affects our performance is the number of active voxels in the user-interested bins. From Fig. 6, we see that the speed-up reduces when a larger number of active voxels is processed. Therefore, if the target histogram includes some bins whose number of active voxels are large, the computation time will increase and can be slower than Sizintsev *et al.*'s method. From the observation, this occurs if the number of active voxels is larger than 20% of total number of data points in the case of the ocean target of Isabel and the pure fuel target of Combustion. This is shown by the red lines in the images in the bottom row of Fig. 6 and the timings of Sizintsev *et al.*'s method in Table 2. From this observation, our work is more suitable for searching local features whose value ranges do not cover large regions in the volume. To summarize, the limitation of our work is the reduced speed-up while searching for features with larger spatial spread. In addition, our work is suitable for the distribution comparison with bin-to-bin error metrics but currently not applicable for cross-bin metrics such as earth movers distance and Kolmogorov-Smirnov distance. In the future, we would like to tackle these limitations to develop a more general algorithm.

8. Conclusion and Future Work

This paper presents an efficient algorithm to search voxels whose local histograms match the user-defined target histogram. Based on the idea that the value range of the user-defined local feature is generally much smaller than that of the entire dataset, the search space in our approach can be reduced. We utilize bitmap indexing to quickly locate voxels in the selected value range, and propose a local deposit scheme to determine the frequency of each bin. Compared to existing local histogram search algorithms, our method is faster in all the cases applying single-bin-error comparison and in the cases with small number of active voxels applying sum-of-error metric. In the future, we want to extend our work to multivariate volume datasets and facilitate the efficient search of joint histograms.

9. Acknowledgments

The authors would like to thank the anonymous reviewers for their comments. 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.

References

[AMCH07] AKIBA H., MA K.-L., CHEN J. H., HAWKES E. R.: Visualizing multivariate volume data from turbulent combustion simulations. *Computing in Science and Engineering 9*, 2 (2007), 76–83. 8

[Ant95] ANTOSHENKOV G.: Byte-aligned bitmap compression. In *Proceedings of the Conference on Data Compression* (1995), DCC '95, pp. 476–. 3

[ARS06] ADAM A., RIVLIN E., SHIMSHONI I.: Robust fragments-based tracking using the integral histogram. In *CVPR (1)* (2006), IEEE Computer Society, pp. 798–805. 2

[BCD*06] BETHEL E., CAMPBELL S., DART E., STOCKINGER K., WU K.: Accelerating network traffic analytics using query-driven visualization. *Visual Analytics Science And Technology, 2006 IEEE Symposium On* (31 2006-Nov. 2 2006), 115–122.

[Bir98] BIRCHFIELD S.: Elliptical head tracking using intensity gradients and color histograms. In *CVPR* (1998), IEEE Computer Society, pp. 232–237. 2

[CHA*11] CHOU J., HOWISON M., AUSTIN B., WU K., QIANG J., BETHEL E. W., SHOSHANI A., RÜBEL O., PRABHAT, RYNE R. D.: Parallel index and query for large scale data analysis. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 30:1–30:11. 2

[CWL*14] CHAUDHURI A., WEI T.-H., LEE T.-Y., SHEN H.-W., PETERKA T.: Efficient range distribution query for visualizing scientific data. In *Proceedings of the 2014 IEEE Pacific Visualization Symposium (PacificVis)* (2014). 2

[DT05] DALAL N., TRIGGS B.: Histograms of oriented gradients for human detection. In *International Conference on Computer Vision & Pattern Recognition* (INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005), Schmid C., Soatto S., Tomasi C., (Eds.), vol. 2, pp. 886–893. 2

[EM95] ENNESSER F., MEDIONI G.: Finding waldo, or focus of attention using local color information. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 8 (August 1995). 2

[GW11] GU Y., WANG C.: Transgraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (Dec. 2011), 2015–2024. 1, 2

[Hou59] HOUGH P. A.: Machine Analysis Of Bubble Chamber Pictures. *Conf.Proc. C590914* (1959), 554–558. 5

[HYT79] HUANG T., YANG G., TANG G.: A fast two-dimensional median filtering algorithm. *Acoustics, Speech and Signal Processing, IEEE Transactions on 27*, 1 (Feb. 1979), 13–18. 2

[JH09] JOHNSON C. R., HUANG J.: Distribution-driven visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics 15*, 5 (2009), 734–746. 2

[KP04] KIM B.-G., PARK D.-J.: Unsupervised video object segmentation and tracking based on new edge features. *Pattern Recognition Letters 25*, 15 (2004), 1731–1742. 2

[LFB98] LAIDLAW D. H., FLEISCHER K. W., BARR A. H.: Partial-volume bayesian classification of material mixtures in mr volume data using voxel histograms. *IEEE Trans. Med. Imaging 17*, 1 (1998), 74–86. 1, 2

[LLY06] LUNDSTRÖM C., LJUNG P., YNNERMAN A.: Local histograms for design of transfer functions in direct volume rendering. *IEEE Trans. Vis. Comput. Graph. 12*, 6 (2006), 1570–1579. 1, 2

[LS13] LEE T.-Y., SHEN H.-W.: Efficient local statistical analysis via integral histograms with discrete wavelet transform. *IEEE Trans. Vis. Comput. Graph. 19*, 12 (2013), 2693–2702. 2

[MGW10] MA Y., GU X., WANG Y.: Histogram similarity measure using variable bin size distance. *Computer Vision and Image Understanding 114*, 8 (2010), 981–989. 3

[PH07] PERREAULT S., HÉBERT P.: Median filtering in constant time. *IEEE Transactions on Image Processing 16*, 9 (2007), 2389–2394. 2

[Por05] PORIKLI F. M.: Integral histogram: A fast way to extract histograms in cartesian spaces. In *CVPR (1)* (2005), IEEE Computer Society, pp. 829–836. 1, 2

[RBPW12] RÜBEL O., BETHEL E. W., PRABHAT, WU K.: Query-Driven Visualization and Analysis. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, Bethel E. W., Childs H., Hansen C., (Eds.). CRC Press/Francis–Taylor Group, Nov. 2012, pp. 117–144. 2

[RM06] RODRIGUEZ Y., MARCEL S.: Face authentication using adapted local binary pattern histograms. In *9th European Conference on Computer Vision (ECCV)* (0 2006). IDIAP-RR 06-06. 2

[SAW12] SU Y., AGRAWAL G., WOODRING J.: Indexing and parallel query processing support for visualizing climate datasets. In *ICPP* (2012), IEEE Computer Society, pp. 249–258. 2

[SDH08] SIZINTSEV M., DERPANIS K. G., HOGUE A.: Histogram-based search: A comparative study. In *CVPR* (2008), IEEE Computer Society. 1, 2, 6

[SSWB05] STOCKINGER K., SHALF J., WU K., BETHEL E. W.: Query-driven visualization of large data sets. In *IEEE Visualization* (2005), IEEE Computer Society, p. 22.

[TLB*11] THOMPSON D. C., LEVINE J. A., BENNETT J., BREMER P., GYULASSY A., PASCUCCI V., PÉBAY P. P.: Analysis of large-scale scalar data using hixels. In *IEEE Symposium on Large Data Analysis and Visualization, LDAV 2011, Providence, Rhode Island, USA, 23-24 October, 2011* (2011), pp. 23–30. 1, 2

[WAB*09] WU K., AHERN S., BETHEL E. W., CHEN J., CHILDS H., CORMIER-MICHEL E., GEDDES C. G. R., GU J., HAGEN H., HAMANN B., KOEGLER W., LAURENT J., MEREDITH J., MESSMER P., OTOO E., PEREVOZTCHIKOV V., POSKANZER A., PRABHAT, RÜBEL O., SHOSHANI A., SIM A., STOCKINGER K., WEBER G., ZHANG W.-M.: Fastbit: Interactively searching massive data. *Journal of Physics Conference Series, Proceedings of SciDAC 2009 180* (June 2009), 012053. 3

[WOS02] WU K., OTOO E. J., SHOSHANI A.: Compressing bitmap indexes for faster search operations. In *SSDBM* (2002), pp. 99–108. 1, 3

[WOS04] WU K., OTOO E., SHOSHANI A.: On the performance of bitmap indices for high cardinality attributes. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30* (2004), VLDB '04, pp. 24–35. 3