Visual Informatics 3 (2019) 157–165

Contents lists available at ScienceDirect

Visual Informatics

journal homepage: www.elsevier.com/locate/visinf

A unified framework for exploring time-varying volumetric data based on block correspondence

Kecheng Lu [a,*], Chaoli Wang [b], Keqin Wu [c], Minglun Gong [d], Yunhai Wang [a,*]

[a] School of Computer Science and Technology, Shandong University, Shandong Province, China
[b] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA
[c] National Oceanic and Atmospheric Administration, Washington, D.C., HI, USA
[d] School of Computer Science, University of Guelph, Guelph, Ontario, Canada

ARTICLE INFO

Article history:
Received 31 August 2019
Accepted 6 October 2019
Available online 21 October 2019

Keywords:
Time-varying data visualization
Block correspondence
Feature extraction and tracking

ABSTRACT

Effective exploration of spatiotemporal volumetric data sets remains a key challenge in scientific visualization. Although great advances have been made over the years, existing solutions typically focus on only one or two aspects of data analysis and visualization. A streamlined workflow for analyzing time-varying data in a comprehensive and unified manner is still missing. Towards this goal, we present a novel approach for time-varying data visualization that encompasses keyframe identification, feature extraction and tracking under a single, unified framework. At the heart of our approach lies in the GPU-accelerated BlockMatch method, a dense block correspondence technique that extends the PatchMatch method from 2D pixels to 3D voxels. Based on the results of dense correspondence, we are able to identify keyframes from the time sequence using k-medoids clustering along with a bidirectional similarity measure. Furthermore, in conjunction with the graph cut algorithm, this framework enables us to perform fine-grained feature extraction and tracking. We tested our approach using several time-varying data sets to demonstrate its effectiveness and utility.

1. Introduction

The tremendous increase of supercomputing power enables the generation and study of increasingly large-scale time-varying volumetric data sets from direct numerical simulations of natural phenomena. Time-varying data are ubiquitous across almost all fields of science. They are dynamic in nature, containing hundreds to thousands of time frames. Therefore, efficient analysis and visualization of time-varying data plays a vital role for scientists to test their hypotheses and discover insights. But it is difficult to identify the regions of interest as the studied phenomena get increasingly complex.

To tackle the above challenge, researchers have proposed different solutions including finding keyframes, clustering data, tracking features, and designing proper transfer functions, etc. However, many of these solutions only focus on one or two aspects of data analysis, missing a streamlined workflow for analyzing the data in a comprehensive and unified manner. For instance, existing solutions often leverage the distributions of data values or feature quantities and apply information theory concepts or probabilistic mixture models to analyze the data (Wang et al., 2008; Dutta and Shen, 2016). These methods either overlook or deemphasize the important *spatial information* involved in the data and therefore are not able to perform accurate feature tracking or highlighting at the voxel level. Other works treat voxels' values over time as time-activity curves (TACs) for temporal clustering or sequencing (Woodring and Shen, 2009; Lee and Shen, 2009). These techniques are able to achieve fine-grained feature clustering and support temporally coherent transfer function design, but *neighborhood information* around a voxel is not considered for a more robust classification. To identify keyframes or representative time steps, previous works also resort to density- or distribution-based solutions (Wang et al., 2008; Tong et al., 2012; Frey and Ertl, 2017) without paying due attention to the higher-level *structural information* exhibited in the time-varying data sets.

Our work follows the recent trend of feature matching from sparse-level (e.g., scale-invariant feature transform, SIFT (Lowe, 2004)) to dense-level (e.g., PatchMatch (Barnes et al., 2009)) in computer vision and image processing. A recent work (Frey and Ertl, 2017) uses the mass transport theory to compute the transformation between two volumes. Spatial information is utilized there to analyze volumes but dense correspondence has not

* Corresponding authors.
E-mail addresses: lukecheng0407@gmail.com (K. Lu), chaoli.wang@nd.edu (C. Wang), keqin.wu@noaa.gov (K. Wu), minglun@uoguelph.ca (M. Gong), cloudseawang@gmail.com (Y. Wang).

Peer review under responsibility of Zhejiang University and Zhejiang University Press.

https://doi.org/10.1016/j.visinf.2019.10.001

been exploited yet. We extend 2D pixel-wise PatchMatch to 3D voxel-wise BlockMatch, enabling dense block correspondence for time-varying volumetric data sets. This step is critical for us to analyze the time-varying volumetric data in a holistic way since the detailed variational pattern of data can be obtained based on the correspondence information. Leveraging GPU, we are able to cost-effectively produce dense correspondence, which allows us to not only perform fine-grained feature extraction and tracking, but also identify keyframes from the entire time sequence in a more accurate fashion. As a result, we are able to present a complete framework that unifies keyframe selection, feature extraction and tracking into a single framework for time-varying data analysis and visualization.

2. Related work

2.1. Time-varying data visualization

Time-varying data visualization is a key topic in scientific visualization and has been a focused research direction since early 1990s. Compared with time-invariant data, the main challenge for time-varying data visualization lies in how to leverage the temporal coherence of data for efficient processing, organization and rendering, and how to capture the dynamic changes of time-varying data features over space and time for visual understanding. In the following, we discuss related work in temporal pattern matching and clustering, feature extraction and tracking, and representative time step selection.

Temporal Pattern Matching and Clustering. Another stream of work on time-varying data visualization focuses on identifying temporal features or trends for clustering and sequencing. The results not only provide a compact view of the time evolution of the data but also give suggestions for transfer function specification. Motivated by the regular expressions and globbing in text string searching, Glatter et al. (2008) developed a textual pattern matching approach for specifying and identifying temporal patterns. Since temporal patterns are sequential by nature, they can include different variables in the specification. Wang et al. (2008) took a blockwise approach to derive the importance values of data blocks using the conditional entropy from information theory. The resulting importance curves reveal the temporal characteristics of data blocks, which are used for block clustering and time-varying feature highlighting. Woodring and Shen (2009) assumed that voxels that behave similarly belong to the same feature and searched the data for classes that have similar behavior over time for temporal clustering and sequencing. The cluster sequences are used to derive transfer functions. In another work Woodring and Shen (2009), they proposed to discover and highlight data based on time-varying trends at different temporal resolutions. Wavelet transforms are used to transform voxel values over time into multi-scale time series curves, which are clustered and displayed in a spreadsheet view for user exploration. Lee and Shen (2009) investigated important multivariate temporal trends using SUBDTW and used them to describe the correlation and causal effects among different variables in a time-varying multivariate data set. Gu and Wang (2013) unified data compacting, indexing and classification into a single framework. Leveraging hierarchical symbolic representation, they built an indexable version of the underlying time-varying data in the form of time-activity curves from which they created a visual representation called iTree for time-varying data exploration. Wang et al. (2016) bundled information from multiple fields into the pattern description. They extracted a sparse set of invariant features for each scalar field using the 3D SIFT algorithm. Users are able to define a pattern as a set of SIFT features in multiple fields via brushing and to locate matching features over time.

Feature Extraction and Tracking. Extracting features and track them over time is an important task for time-varying data visualization. Samtaney et al. (1994) extracted features (i.e., regions that fulfill certain criteria) and made correspondence among features in neighboring time steps. They matched features through their attributes such as the centroid, volume and mass. To allow fast identification of feature overlapping in space, Silver and Wang (1997) utilized octrees to store all the features, used spatial overlap to determine matching feature candidates, and applied a normalized volume difference test to choose the best matching from the candidates. Woodring and Shen (2009) identified features or clusters based on the input time-activity curves within a time interval and created a directed graph that connects the clusters over time. To find the links between features, they estimated the probability of transferring one feature to another by computing the distance of their time histograms. Ozer et al. (2014) introduced the use of Petri nets to model and detect activities in scientific visualization. They utilized Petri nets to model the activity of interest and ran the algorithm for hypothesis validation based on the results of feature extraction and tracking. Dutta and Shen (2016) designed a distribution-driven approach that utilizes the motion and similarity properties of an object to the target feature and fuses the information gained from them to generate a classification field at every time step. Accurate and robust feature tracking is achieved based on the resulting feature-aware classification fields. In this paper, we allow users to mark features in keyframes and generate mask volumes for intermediate frames. Feature tracking is achieved by volumetric feature segmentation via the graph cut algorithm.

Representative Time Step Selection. Selecting important time steps becomes an issue when the number of time steps in a time-varying data set is large. This is analogous to keyframe selection in video analysis. To identify representative time steps, Wang et al. (2008) partitioned the time sequence into segments with nearly equal accumulated importance values for the time steps within in each segment. They selected a representative time step from each segment to maximize the joint entropy of the selected time steps. Tong et al. (2012) applied different metrics to compute the distance between data sets and employed the dynamic programming strategy to select the most interesting time steps accordingly. Frey and Ertl (2017) presented a time step selection method that works in streaming and in-situ settings. This method is based on progressive volume transformation that can interpolate and provide meaningful distances between data sets. Their solution selects a set of time steps from which the time series can be reconstructed without exceeding a given error bound. In another work Frey and Ertl (2016), they proposed a different time step selection solution that optimizes the coverage of the complete data on the basis of a minimum flow-based technique to determine meaningful distances between time steps. Inspired by how keyframes are detected from video processing, we identify representative time steps or keyframes using a combination of a dense block correspondence method and the k-medoids clustering method.

2.2. Correspondence in image and video

Estimating correspondence between two images is a fundamental problem in computer vision (Szeliski, 2006). It is often performed by either matching a sparse set of key points, or by matching all pixels in both images. For efficiency, many methods have been proposed to build the sparse correspondence, among which SIFT-based correspondence (Lowe, 2004) searching is commonly used. These methods work well for many tasks, such as 3D reconstruction and object detection, but they are less effective for applications like texture synthesis and image completion that

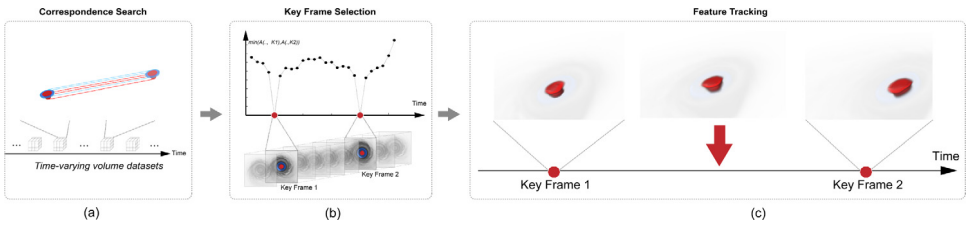

Fig. 1. Our approach for analysis and visualization of time-varying volumetric data includes correspondence search (a) and keyframe selection (b). The applications of our approach is feature tracking (c).

inherently require the dense correspondence. Our applications of feature tracking and transfer function melding both require the dense correspondence between voxels in 3D volume.

Instead, the dense correspondence for video sequences can be readily obtained by estimating optical flow. Specifically, it formulates the correspondence between two adjacent frames as an estimation of a 2D flow field (Brox and Malik, 2011) solved by global optimization. Since this method assumes the locality and smoothness of the flow, it can efficiently handle the images with small changes, but might fail to match objects under large displacements. On the other hand, its involved global optimization is extremely time-consuming for large images. In contrast, PatchMatch proposed by Barnes et al. (2009) can not only find dense correspondences an order of magnitude faster than previous approaches but also handle the objects with large displacements. Due to these advantages, it has been used for many different applications, such as video summarization (Barnes et al., 2010), image melding (Darabi et al., 2012) and synthesis of artistic brushes (Lu et al., 2013). In this work, we extend it to 3D volume data and demonstrate its usefulness in time-varying volume visualization.

3. Overview

Our goal is to find the dense correspondence in time-varying data, from which we can analyze the relationships among different time frames in a holistic way and better visualize the underlying data. Fig. 1 illustrates the framework of our block correspondence based method for time-varying data exploration. The major components are:

Correspondence Search. Our approach first extends the PatchMatch method from 2D image processing to a BlockMatch method that computes 3D block correspondences between each pair of volume frames (Section 4.1). Dense voxel-level correspondences are obtained using overlapping blocks centered at each voxels.

Keyframe Selection. In Section 4.2, we use the bidirectional similarity metric to measure the distance between volumes and derive the volume distance matrix, from which we employ the k-medoids clustering method to identify keyframes.

Applications. With keyframes and correspondences built between frames, we design a block voting approach for propagating information defined on keyframes to intermediate frames (Section 5.1). The information propagated can be used for tracking and extracting features from the time-varying volume sequence (Section 5.2).

4. Correspondence analysis

Given a sequence of time-varying volume data, we would like to analyze the content by first matching between different volumes and then selecting keyframes among the sequence.

4.1. Blockmatch

Our first task is to find the correspondences for all volume pairs. That is, for any given two volumes S and T, we solve the following optimization

$$M_{S\to T}(s) = \arg\min_{t\in T} D(s,t), \tag{1}$$

where $s \in S$ and $t \in T$ are voxels in the two volumes and $M_{S\to T}$ is the mapping between them. $D(\cdot,\cdot)$ measures the distance in the attribute space, where the attributes can be scalar values or multivariate value vectors. Although our goal is to build the dense correspondence between voxels, directly searching the best match for each individual voxel does not take the neighboring context into consideration and hence often leads to mismatches (Wexler et al., 2007). Hence, we define $D(s,t)$ as the sum of attribute distances between corresponding voxels within two blocks centered at s and t. That is,

$$D(s,t) = \sum_{k\in\Gamma} \|f(s+k) - f(t+k)\|, \tag{2}$$

where Γ is a set containing vectors that connect from the center voxel x to all voxels in the block domain (including x itself). $f(x)$ is the attribute defined at x. Since the best match for a given voxel x is determined using a block of voxels centered at x, the mismatching problem can be effectively addressed. On the other hand, the blocks for adjacent voxels overlap with each other, allowing dense correspondences being computed.

The naive brute-force search for the above optimization is expensive: $O(M \times N^2)$ for the volume with N voxels and block size of M. We therefore choose to adopt the PatchMatch algorithm (Barnes et al., 2009), which is known for its efficiency in building patch-based dense correspondence for image pairs. For every patch within the source image, it finds the approximate nearest neighbor (ANN) patch in the target image, which is an order of magnitude faster than previous approaches. The essence of PatchMatch lies in the observation that neighboring patches have probably neighboring matches.

Time-varying volume sequences also follow this observation (Woodring and Shen, 2009). However, directly extending PatchMatch from 2D to 3D incurs high computational cost. We therefore propose BlockMatch that customizes the original PatchMatch algorithm for efficient matching among time-varying volume sequences. As shown in Fig. 2, our BlockMatch has three main components. After initialization, it improves the correspondences by iteratively alternating propagation and random search.

Initialization. During the initialization, the original 2D PatchMatch assigns each patch in the source image to a random patch in the target image. Due to the extended search space, using the same random initialization strategy for 3D BlockMatch is highly inefficient. Considering that we are handling time-varying volume data, where changes among adjacent volumes are usually small, we simply assign a block in the source volume to the block centered at the same voxel in the target volume, as shown in 2(a).

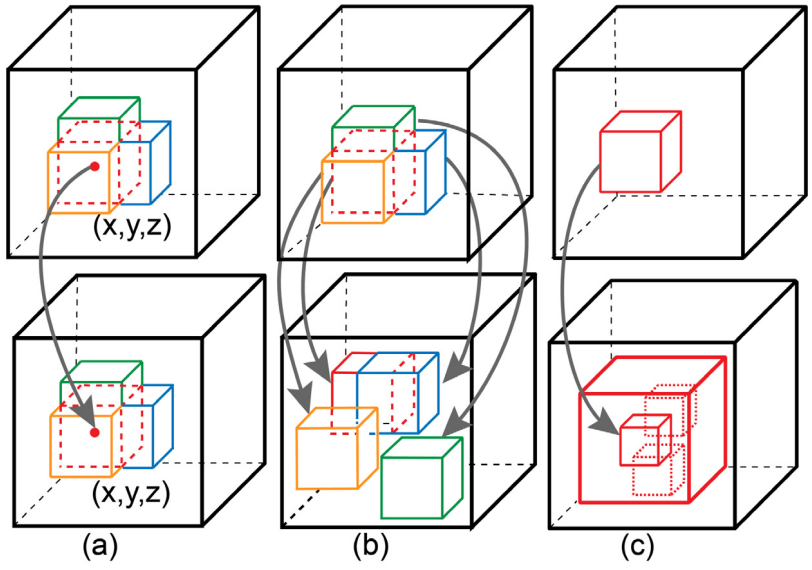

Fig. 2. Three components of the BlockMatch algorithm: (a) blocks in the source volume initially are assigned to the blocks centered at the same voxel in the target volume; (b) the red block checks neighbors to see if they will improve the correspondence, propagating good matches; (c) the block searches randomly for improvements in a restricted subvolume.

For two volumes that are far apart in the sequence, we still use the random initialization strategy.

Propagation. A key observation used in 2D PatchMatch is that, if a source patch matches well with a target patch, then the neighbors of this source patch would likely match well with those of the target patch. Hence, the original PatchMatch algorithm uses the propagation operation to spread good matches among the neighbors. We inherit this idea for 3D BlockMatch. That is, we use the following formula to update the matching block for s based on the matches of s's neighbors

$$M_{S\to T}(s) = \arg\min_{n\in\Omega} M_{S\to T}(s+n) - n, \tag{3}$$

where Ω is the set containing six offset vectors ($\pm$ in x, y, z).

Random Search. Optimization with propagation only will lead to premature convergence to locally optimal solutions. Hence, we apply a global random search step after the propagation operation. That is, assume a source block p matches a target block q_0, then a random new block q_i within the subvolume centered at q will be tested

$$q_i = q_0 + w\xi\mathbf{R}_i, \tag{4}$$

where $\mathbf{R}_i$ is a uniform random vector in $[-1, 1]\times[-1, 1]\times[-1, 1]$, w is the maximum search radius, and ξ is a fixed ratio. If the new block q_i is a better match, it will replace q_0. This step is repeated multiple times, where the subvolume centered at q for random block selection reduces w by half. For our time-varying volume data, a global random search is time-consuming and not always necessary due to the spatiotemporal coherence of the corresponding volumes in adjacent time steps. Accordingly, a smaller $\xi = 1/6$ value is used to improve the time efficiency, whereas the initial value of w is set to the maximum volume dimension.

GPU implementation. Since performing such an ANN search in 3D is time-consuming, we implement random search on GPU in parallel because of each voxel is independent of each other. Meanwhile, we adopt jump flooding (Rong and Tan, 2006) to implement propagation on GPU. Our experiment shows that with this new version, we can speed up the timing performance 20 times and a pair of frames can be handled within 8 s for two $128 \times 128 \times 128$ volumes with the block size of $7 \times 7 \times 7$.

4.2. Keyframe selection

In many cases, the volume data do not change at a uniform pace over time. Through identifying key time frames, the volume data sequence can be split into multiple smoothly-varying segments, each of which is bounded by two keyframes. We will show later that this treatment allows features defined on keyframes to be faithfully interpolated through the intermediate frames. Based on the dense correspondence, we employ a perceptual distance metric, bidirectional similarity (BDS) metric (Simakov et al., 2008) to identify keyframes. After calculating the BDS distance for all volume pairs, a $n \times n$ distance matrix (n is the number of time steps) is constructed for keyframe selection.

Bidirectional Similarity Metric (BDS). BDS offers a new approach to summarize volume data by measuring the sum of the average distance of all blocks in S to their most similar (nearest-neighbor) blocks in T and vice versa

$$BDS(S,T) = \frac{1}{N_S}\sum_{s\in S} D(s, M_{S\to T}(s)) + \frac{1}{N_T}\sum_{t\in T} D(t, M_{T\to S}(t)), \tag{5}$$

where s and t are two voxels in S and T. N_S and N_T are the number of blocks in S and T. $D(s, t)$ are defined in Eq. (2). We can see that the smaller the BDS is, the closer S and T are.

Keyframe Identification. Given a $n \times n$ distance matrix, our goal is to identify a set of keyframes $\mathbf{k} = \{k_1, \ldots, k_m\}$. Here, we treat keyframe selection as a clustering problem, where each keyframe k_i is a representative frame for a cluster (sub-sequence). This goal can be formulated as the objective function

$$F(\mathbf{k}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\tau_{i,k_j} BDS(i, k_j), \tag{6}$$

where $BDS(i, k_j)$ is the distance between the ith frame and keyframe k_j, and τ_{i,k_j} is a binary variable. If the ith frame is assigned to the jth cluster, τ_{i,k_j} is 1; otherwise τ_{i,k_j} is 0.

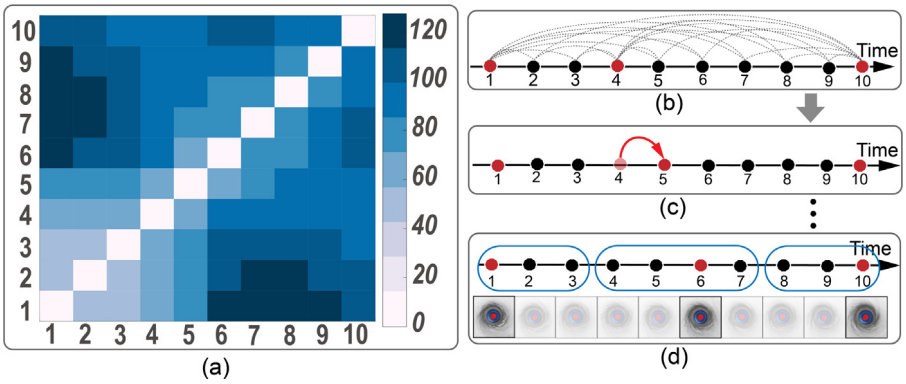

Fig. 3. (a) Using the distance matrix to identify three keyframes where the 1st and 10th frames must be keyframes. (b) The randomly selected 4th frame is taken as initialization of one additional keyframe. (c) During the first iteration the 5th frame is selected. (d) Finally the 6th frame is selected and 10 frames are assigned to three clusters.

We minimize Eq. (6) by adapting the k-medoid clustering algorithm (Kaufman and Rousseeuw, 1987), which attempts to find a non-overlapping set of clusters such that each cluster has the most representative point, referred as *medoid*. Specifically, the algorithm proceeds by alternating between two steps: (i) associate each point to the closest medoid; and (ii) select the medoids $\mathbf{k}$ with the smallest value of $F(\mathbf{k})$. After a certain number of iterations, we are able to identify stable keyframes. To guarantee each time step is bounded by two key frames, we treat the first and last frames as pre-determined keyframes, while enforcing the constraint that neighboring frames should be assigned to the same cluster. Fig. 3 shows an example of the subset of *Hurricane Isabel* data set, where the 6th frame is selected as the additional keyframe after two iterations.

5. Applications

Interactive visual exploration of time-varying volume data is still a challenge in many areas of science and engineering (Ma, 2003). One main reason is the lack of effective ways for feature extraction/tracking. In this section, we show that our block-based dense correspondence allows us to construct novel algorithms for semi-automatic feature extraction/tracking.

In our scenario, the user interactively specifies features of interest for extracted keyframes, from which we automatically derive the features for the intermediate frames. There are three ways to derive such features. The straightforward way is to directly apply the features for keyframes to the intermediate frames but ignores the dynamic nature of time-varying data. The second way is to sequentially propagate features of one keyframe to intermediate frames (Wang et al., 2011), where the incoherence could quickly accumulate and add up to a larger error after a few propagation steps. We choose the third way, which is to generate features for each intermediate frame based on the features for its two bounding keyframes.

In our framework, features specified by users at keyframes are treated as general augmented information. That is, volumetric features of interest are considered as binary attribute of the data. Here we first discuss how to propagate the augmented information to intermediate frames using dense voxel correspondences (Section 5.1), which is followed by discussions on recovering features (Section 5.2) at intermediate frames.

5.1. Information propagation through block voting

Given an intermediate frame S and a nearby keyframe T, the dense correspondence indicates that voxel $p \in S$ corresponds to voxel $q = M_{S\to T}(p)$ in T. Hence the augmented information defined at q can be transferred to p. However, such direct information transfer approach corresponds to the nearest pixel sampling used in backward image warping, which often leads to aliasing artifacts. To address this problem, we introduce *block voting*, which considers the correspondences of all neighboring blocks containing p for information transfer.

More specificity, voxel p is contained in all blocks centered at $p + k$ ($k \in \Gamma$) when performing block matching in Section 4.1. Hence, the correspondences for these blocks, $q^k = M_{S\to T}(p + k)$, are used to compute the information transferred to p based on the following weighted average function:

$$g(p) = \frac{\sum_{k\in\Gamma} \omega(p+k, q^k) g(q^k - k)}{\sum_{k\in\Gamma} \omega(p+k, q^k)}, \tag{7}$$

where $g(x)$ denotes the augmented information defined at x. $\omega(s, t)$ is a weight function defined based on block similarity between two blocks of voxels centered at $s \in S$ and $t \in T$. That is:

$$\omega(s,t) = \exp\left(-\frac{D(s,t)}{2\sigma^2}\right) \eta^{-dist(s,t)}, \tag{8}$$

where the first term uses a Gaussian kernel to measure attribute similarity between the two blocks, whereas the second term measures spatial similarity as in Wexler et al. (2007). $dist(s, t)$ computes the Euclidean distance between voxels s and t, whereas η is a constant. We empirically set $\eta = 1.8$ in our implementation (see Fig. 4).

5.2. Feature tracking

When studying time-varying data, researchers are often interested in certain features or structures and their evolution along time. For example, physicists and meteorologists are interested in some physically meaningful features like vortices and hurricane eyes. However, they often do not have a precise definition of feature. Similar to the work of Dutta and Shen (2016), we allow the user to define features by interactively thresholding attributes or masking regions of interest with a box. Users only need to specify features on keyframes, where a voxel is labeled as either a feature (value 1) or a non-feature (value 0). This augmented information (label volume) is propagated to all other intermediate frames.

Mask Fusion. With features masks defined at keyframes k_i and k_{i+1} on both ends, we now try to synthesize the mask volume for the intermediate frame at the jth time step via block voting. Specifically, for each voxel p in the jth frame, we use

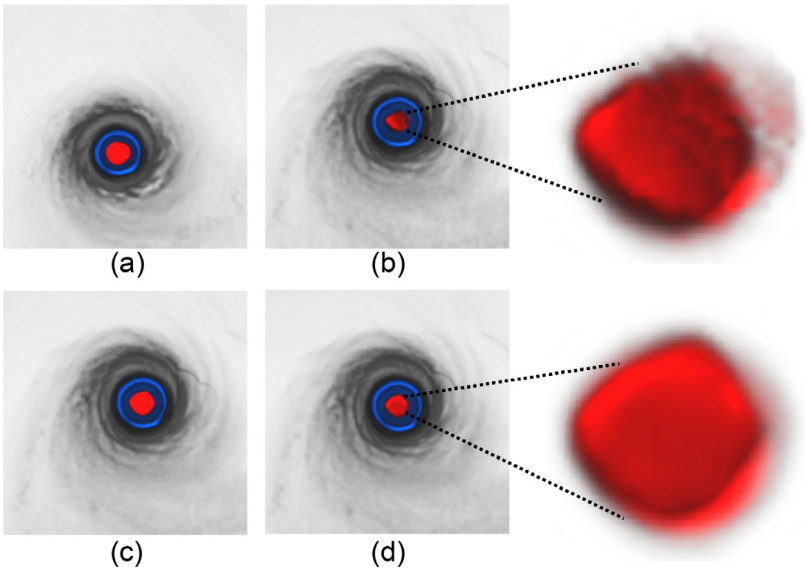

Fig. 4. (a) and (c): the volumes of the 15th and 29th time steps, respectively. (b) and (d): the results generated by replacing the color and opacity volume of the 29th time step with that of the 15th time step via direct replacement and block voting, respectively. We can see that the hurricane's eye in (d) is closer to that in (c) than the one in (b).

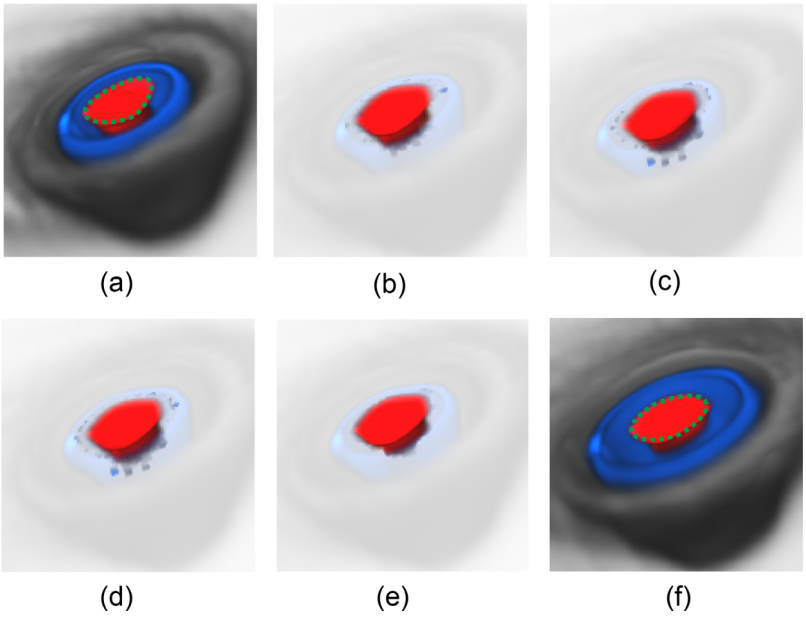

Fig. 5. Extraction of the hurricane's eye at the 20th time step based on the masks specified for the keyframes at the 15th and 30th time steps. (a) and (f): the eyes at the 15th and 30th time steps are selected by green lassos, respectively. (b) and (c): the mask volumes are synthesized using the mask volumes at the 15th and 30th time steps, respectively. (d): the mask volume is generated by fusing the ones in (b) and (c). (e): the mask volume is generated by applying graph-cut. For perceiving the motion of the eye, all mask volumes are shown with the rest of volume as the background.

Eq. (7) to compute $g^{k_i}(p)$ and $g^{k_{i+1}}(p)$ using keyframes k_i and k_{i+1}, respectively. Then, the final voting value is computed as

$$\upsilon_p = \rho g^{k_i}(p) + (1-\rho) g^{k_{i+1}}(p), \tag{9}$$

where $\rho = (k_{i+1} - j)/(k_{i+1} - k_i)$. In this way, two volumes synthesized from each keyframe are fused together. As an example, Fig. 5(d) shows the result by fusing the volumes shown in Fig. 5(b) and (c).

Volume Segmentation. The synthesized mask volume describes the probability of each voxel being a feature. To clearly extract features regions, we use the graph cut algorithm (Boykov et al., 2001) to arrive at the final binary segmentation. Given a volume, we define the graph $G = \{V; E\}$ where the nodes V are given by the voxels and adjacent voxels are connected by edges. Then, we find the label of each voxel through optimizing the following objective function

$$E(L) = \sum_p D_p(l_p) + \lambda \sum_{q \in N_p} V_{pq}(l_p, l_q), \tag{10}$$

where N_q is the neighborhood of p, λ is the weight that balances the influence of the terms $D_p(l_p)$ and $V_{pq}(l_p, l_q)$. $D_p(l_p)$ is the data cost for assigning label l to voxel p and $V_{pq}(l_p, l_q)$ is the smoothness cost for assigning l_p and l_q to p and q, respectively. Since our feature mask is a binary volume, we set $D_p(l_p)$ as υ_p for being feature, otherwise as $1 - \upsilon_p$. The smooth cost is defined

as the magnitude of the gradient vector given by p and q. In our experiment, we empirically set $\gamma = 1.3$ for all data sets.

For the *Hurricane Isabel* data set, the optimization can be completed under 2 min for each mask. Fig. 5(e) shows the result by applying graph-cut to the fused result (Fig. 5(d)), where we can see that the eye's shape gets smoother and the noise is removed.

6. Results and discussion

We evaluate the effectiveness of our approach in the exploration of time-varying volume data using several scientific data sets. Compared to previous work (Wang et al., 2016; Dutta and Shen, 2016; Woodring and Shen, 2009; Wang et al., 2011), our approach enables the user to perform feature extraction/tracking in the same framework. Accordingly, we evaluate our approach in two different ways:

1. illustrating its efficacy for each individual task by comparing our method with previous work; and
2. demonstrating the utility of the entire framework through both tasks along with case studies.

All experiments were performed on a workstation with an NVIDIA GTX980 graphics card, Intel Core i7-6700K, and 16GB RAM.

6.1. Parameter choice and performance analysis

Besides the selection of regions of interest for keyframes, our framework has two other parameters: block size and the number of keyframes. A large block size will introduce large computational overhead but a small block might not have enough neighboring context. Thus, we empirically used $7 \times 7 \times 7$ block size in the correspondence computation for all data sets. Regarding the number of keyframes, we suggest the user to start from a small one. If some features cannot be characterized, then the user could manually add keyframes.

In Table 1 the timings are reported for the tested cases. We can see that our GPU implementation of BlockMatch is about 20–70 times faster than the CPU implementation. For the data sets with smaller size, the speedup is larger. The timing results for feature tracking include block voting, whose running time can be negligible compared to the optimization cost. Since the time complexity for graph cut optimization is proportional to the number of voxels, it takes more time for larger volume.

6.2. Evaluation of feature tracking

To demonstrate the effectiveness of our approach in feature extraction and tracking, we first compare it with the state-of-the-art method (Dutta and Shen, 2016) and then conduct a case study to further verify its accuracy.

6.2.1. Comparative evaluation: Hurricane Isabel data set

Dutta and Shen (2016) present a distribution-driven approach for feature extraction and tracking (DDFT). By modeling the feature of interest with probability distribution, they measured the possibility of each block being a part of feature with classification fields and then extracted the feature by thresholding the possibility value. In our method, the fused mask volume can be regarded as a classification field but we can automatically extract features using graph cut without a hard threshold. Moreover, they proceeded in an incremental way that could result in an accumulation of errors from multiple steps, while our block voting based on two keyframes within each time segment can alleviate this problem.

For comparison, we use the *pressure* variable of the *Hurricane Isabel* data set, courtesy of NCAR and NSF generated using the WRF model. This data set consists of 48 time steps, where each step has the dimension of $500 \times 500 \times 100$. To extract and track the temporal evolution of the hurricane's eye, DDFT first computes the λ_2 field using the velocity field for initial selection of the vortex regions, then tracks the feature in the classification field and finally extracts the feature with a carefully selected threshold. Instead, our method extracts four keyframes (the 1st, 15th, 32nd, and 48th frames) and asks the user to select the features of interest.

Fig. 6 shows the results at three time steps generated by DDFT and our method. We can see that our method can extract more the vortex feature more accurately, whereas the eye's shape extracted by DDFT varies a lot over time. To investigate the overall temporal evolution, we compare the trace volumes generated by two methods as shown in Fig. 6(d). It is clear that our method preserves the eye's shape during the motion while DDFT extracts a larger shape at initial time steps and gradually decreases the size. We assume the reason for the size change of DDFT is due to the fact that they used the same threshold of possibility values for the entire time sequence.

6.2.2. Case study: Tornado data set

To verify the accuracy of our extracted feature, we use the *Tornado* data set ($128 \times 128 \times 128$) generated by an analytical function (Crawfis and Max, 1993), containing the velocity vector at each grid point. This data simulates the temporal evolution of the tornado such as the vortex core structure with 50 time steps. Here, we increase the motion speed by modifying the equation to see whether our method can still accurately track the feature when the time dimension is sampled sparsely. After extracting four keyframes (the 1st, 16th, 30th, and 50th frames), we compute the *vortexness* at each point using the λ_2 vortex criterion, then manually select the vortex core structure, and finally apply our method to extract the structure for the rest of frames.

Fig. 7 depicts the tracked features of interest at eight selected time steps. We can see that almost all interesting vortex structures are clearly extracted. Note that some neighboring frames have different structures because of the high-speed motion, but their structures are still faithfully extracted. Compared with the visualizations generated by previous work (Crawfis and Max, 1993), we believe that our results are reasonable. We did not compare with DDFT (Dutta and Shen, 2016), because they have also modified the analytical equation and involved different thresholds of the λ_2 value.

In summary, we can see that our method is capable of extraction and tracing of features and produces the results similar with the state-of-the-art methods. Since our method considers both correspondence between data attributes and spatial locations, it produces more accurate results for some data sets as shown in Fig. 6.

7. Conclusions and future work

We have presented a unified approach for time-varying data exploration based on block correspondence. The main contributions of our work are the following. First, we introduce the concept of dense correspondence to time-varying data analysis and demonstrate its practical feasibility with the use of GPU implementation to speed up the computation. Second, we extract keyframes based on dense correspondence using the bidirectional similarity metric and k-medoids clustering method. Third, we present new methods for feature tracking via the graph cut segmentation technique via the block voting scheme.

In the future, we would like to further investigate the extension of our approach to handle time-varying multivariate data sets. The BlockMatch method can readily take the multivariate

Table 1
The descriptions of data set and transfer function, BlockMatch time for a pair of volumes on CPU and GPU, and time for tracking feature and transfer function melding of one time step. Times are in seconds.

Data		TF		Time		
Name	Size	#Gaussian	#Dim	BlockMatch(CPU)	BlockMatch(GPU)	Tracking
Tornado	128 × 128 × 128 × 50	2	1D	335.28	7.95	36.38
Hurricane Isabel	500 × 500 × 100 × 48	3	2D	4626.50	80.10	112.87
Earthquake	256 × 256 × 96 × 599	4	1D	445.17	20.33	51.10
Vortex	128 × 128 × 128 × 100	3	1D	293.57	7.95	32.15
Turbulent Flow	192 × 64 × 48 × 113	3	2D	66.20	1.46	10.12

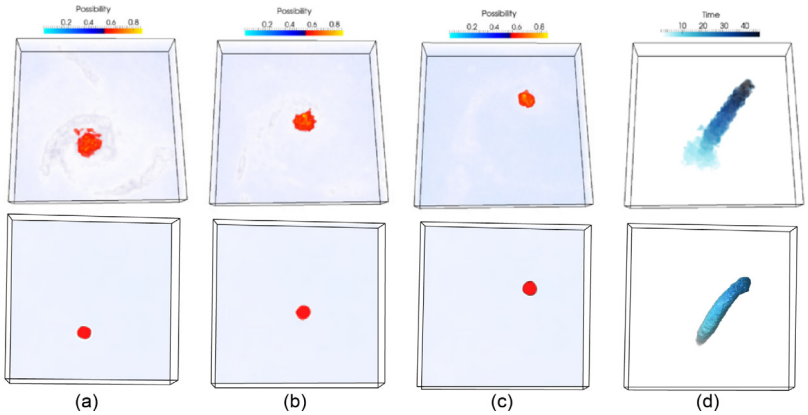

Fig. 6. Comparison the method of Dutta and Shen (2016) (the top row) and our method (the bottom row) for extraction and tracking of the vortex, which corresponds to the eye of the *Hurricane Isabel* data set. (a)–(c) show the 11th, 28th, and 41st time steps, respectively. (d) shows the temporal trace volume of the tracked feature.

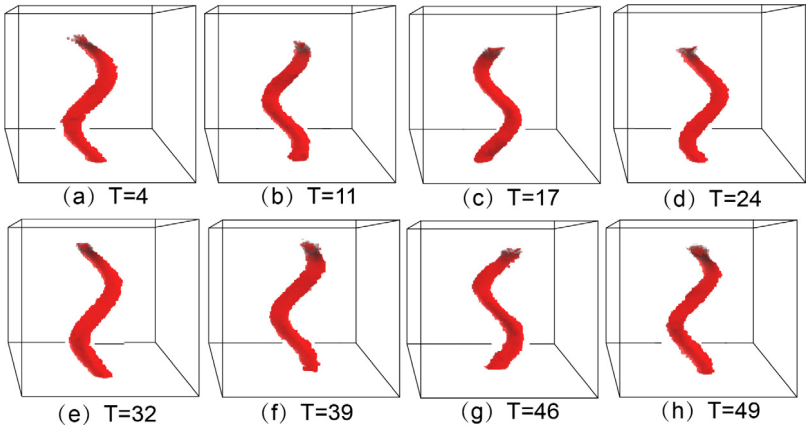

Fig. 7. Extraction and tracking of the vortex core of the *Tornado* data set. Eight selected time steps are shown here.

data as input as the distance measure in the attribute space can take multivariate value vectors. For feature tracking, the attribute volume could be generalized to consider multivariate attributes instead of single scalar values. We would investigate how the computational performance would be affected by this extension and seek the acceleration solution as needed.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported by the grants of BKBD-2017KF02.

References

Barnes, C., Goldman, D.B., Shechtman, E., Finkelstein, A., 2010. Video tapestries with continuous temporal zoom. ACM Trans. Graph. 29 (4), 89:1–89:9. http://dx.doi.org/10.1145/1778765.1778826.

Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B., 2009. Patchmatch: a randomized correspondence algorithm for structural image editing. ACM Trans. Graph. 28 (3), 24:1–24:11. http://dx.doi.org/10.1145/1531326.1531330.

Boykov, Y., Veksler, O., Zabih, R., 2001. Fast approximate energy minimization via graph cuts. IEEE Trans. Pattern Anal. Mach. Intell. 23 (11), 1222–1239. http://dx.doi.org/10.1109/34.969114.

Brox, T., Malik, J., 2011. Large displacement optical flow: descriptor matching in variational motion estimation. IEEE Trans. Pattern Anal. Mach. Intell. 33 (3), 500–513. http://dx.doi.org/10.1109/TPAMI.2010.143.

Crawfis, R. A., Max, N., 1993. Texture splats for 3D scalar and vector field visualization. In: Proceedings of IEEE Visualization Conference, pp. 261–266. http://dx.doi.org/10.1109/VISUAL.1993.398877.

Darabi, S., Shechtman, E., Barnes, C., Goldman, D.B., Sen, P., 2012. Image melding: combining inconsistent images using patch-based synthesis. ACM Trans. Graph. 31 (4), 82:1–82:10. http://dx.doi.org/10.1145/2185520.2185578.

Dutta, S., Shen, H.-W., 2016. Distribution driven extraction and tracking of features for time-varying data analysis. IEEE Trans. Vis. Comput. Graphics 22 (1), 837–846. http://dx.doi.org/10.1109/TVCG.2015.2467436.

Frey, S., Ertl, T., 2016. Flow-based temporal selection for interactive volume visualization. Comput. Graph. Forum 36 (8), 153–165. http://dx.doi.org/10.1111/cgf.13070.

Frey, S., Ertl, T., 2017. Progressive direct volume-to-volume transformation. IEEE Trans. Vis. Comput. Graphics 23 (1), 921–930. http://dx.doi.org/10.1109/TVCG.2016.2599042.

Glatter, M., Huang, J., Ahern, S., Daniel, J., Lu, A., 2008. Visualizing temporal patterns in large multivariate data using textual pattern matching. IEEE Trans. Vis. Comput. Graphics 14 (6), 1467–1474. http://dx.doi.org/10.1109/TVCG.2008.184.

Gu, Y., Wang, C., 2013. iTree: exploring time-varying data using indexable tree. In: Proceedings of IEEE Pacific Visualization Symposium, pp. 137–144. http://dx.doi.org/10.1109/PacificVis.2013.6596138.

Kaufman, L., Rousseeuw, P.J., 1987. Clustering by means of medoids. In: Dodge, Y. (Ed.), Statistical Data Analysis Based on the L_1–Norm and Related Methods. North-Holland, pp. 405–426.

Lee, T.-Y., Shen, H.-W., 2009. Visualization and exploration of temporal trend relationships in multivariate time-varying data. IEEE Trans. Vis. Comput. Graphics 15 (6), 1359–1366. http://dx.doi.org/10.1109/TVCG.2009.200.

Lowe, D.G., 2004. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. 60 (2), 91–110. http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

Lu, J., Barnes, C., DiVerdi, S., Finkelstein, A., 2013. Realbrush: painting with examples of physical media. ACM Trans. Graph. 32 (4), 117:1–117:12. http://dx.doi.org/10.1145/2461912.2461998.

Ma, K.-L., 2003. Visualizing time-varying volume data. IEEE Comput. Sci. Eng. 5 (2), 34–42. http://dx.doi.org/10.1109/MCISE.2003.1182960.

Ozer, S., Silver, D., Bemis, K., Martin, P., 2014. Activity detection in scientific visualization. IEEE Trans. Vis. Comput. Graphics 20 (3), 337–390. http://dx.doi.org/10.1109/TVCG.2013.117.

Rong, G., Tan, T.-S., 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In: Proceedings of ACM Symposium on Interactive 3D Graphics and Games, pp. 109–116. http://dx.doi.org/10.1145/1111411.1111431.

Samtaney, R., Silver, D., Zabusky, N., Cao, J., 1994. Visualizing features and tacking their evolution. IEEE Comput. 27 (7), 20–27. http://dx.doi.org/10.1109/2.299407.

Silver, D., Wang, X., 1997. Tracking and visualizing turbulent 3D features. IEEE Trans. Vis. Comput. Graphics 3 (2), 129–141. http://dx.doi.org/10.1109/2945.597796.

Simakov, D., Caspi, Y., Shechtman, E., Irani, M., 2008. Summarizing visual data using bidirectional similarity. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8. http://dx.doi.org/10.1109/CVPR.2008.4587842.

Szeliski, R., 2006. Image alignment and stitching: a tutorial, foundations and trends®. Comput. Graph. Vis. 2 (1), 1–104. http://dx.doi.org/10.1561/0600000009.

Tong, X., Lee, T.-Y., Shen, H.-W., 2012. Salient time steps selection from large scale time-varying data sets with dynamic time warping. In: Proceedings of IEEE Symposium on Large Data Analysis and Visualization, pp. 49–56. http://dx.doi.org/10.1109/LDAV.2012.6378975.

Wang, Y., Chen, W., Zhang, J., Dong, T., Shan, G., Chi, X., 2011. Efficient volume exploration using the gaussian mixture model. IEEE Trans. Vis. Comput. Graphics 17 (11), 1560–1573. http://dx.doi.org/10.1109/TVCG.2011.97.

Wang, Z., Seidel, H.-P., Weinkauf, T., 2016. Multi-field pattern matching based on sparse feature sampling. IEEE Trans. Vis. Comput. Graphics 22 (1), 807–816. http://dx.doi.org/10.1109/TVCG.2015.2467292.

Wang, C., Yu, H., Ma, K.-L., 2008. Importance-driven time-varying data visualization. IEEE Trans. Vis. Comput. Graphics 14 (6), 1547–1554. http://dx.doi.org/10.1109/TVCG.2008.140.

Wexler, Y., Shechtman, E., Irani, M., 2007. Space-time completion of video. IEEE Trans. Pattern Anal. Mach. Intell. 29 (3), 463–476. http://dx.doi.org/10.1109/TPAMI.2007.60.

Woodring, J., Shen, H.-W., 2009. Multiscale time activity data exploration via temporal clustering visualization spreadsheet. IEEE Trans. Vis. Comput. Graphics 15 (1), 123–137. http://dx.doi.org/10.1109/TVCG.2008.69.

Woodring, J., Shen, H.-W., 2009. Semi-automatic time-series transfer functions via temporal clustering and sequencing. Comput. Graph. Forum 28 (3), 791–798. http://dx.doi.org/10.1111/j.1467-8659.2009.01472.x.