DOI: 10.1111/cgf.13399
Eurographics Conference on Visualization (EuroVis) 2018
J. Heer, H. Leitte, and T. Ropinski
(Guest Editors)
*Volume 37 (2018), Number 3*

# Key Time Steps Selection for Large-Scale Time-Varying Volume Datasets Using an Information-Theoretic Storyboard

Bo Zhou[1] and Yi-Jen Chiang[1]

[1]Department of Computer Science and Engineering, Tandon School of Engineering, New York University, USA

**Abstract**

*Key time steps selection is essential for effective and efficient scientific visualization of large-scale time-varying datasets. We present a novel approach that can decide the number of most representative time steps while selecting them to minimize the difference in the amount of information from the original data. We use linear interpolation to reconstruct the data of intermediate time steps between selected time steps. We propose an evaluation of selected time steps by computing the difference in the amount of information (called* information difference*) using* variation of information (VI) *from information theory, which compares the interpolated time steps against the original data. In the one-time preprocessing phase, a* dynamic programming *is applied to extract the subset of time steps that minimize the information difference. In the run-time phase, a novel chart is used to present the dynamic programming results, which serves as a storyboard of the data to guide the user to select the best time steps very efficiently. We extend our preprocessing approach to a novel out-of-core approximate algorithm to achieve optimal I/O cost, which also greatly reduces the in-core computing time and exhibits a nice trade-off between computing speed and accuracy. As shown in the experiments, our approximate method outperforms the previous globally optimal DTW approach [TLS12] on out-of-core data by* ***significantly*** *improving the running time while keeping similar qualities, and is our major contribution.*

**Keywords:** Key Time Steps Selection, Time-Varying Volume Data, Scalar Field Data, Information Theory.

## 1. Introduction

Due to the exponential growth in data sizes, scientific visualization of time-varying datasets has constantly posted a big challenge. As huge numbers of time steps become common in time-varying simulations, where tiny changes between consecutive time steps are usually not very informative, selecting a subset of the most salient time steps to visualize becomes crucial in effective and efficient visual analysis of the evolution of the data. However, this crucial task is highly non-trivial, since the evolution of important events can have complex patterns and occur at unknown frequencies across the time steps. In this paper, we develop an *information-theoretic* approach for the evaluation and selection of key time steps from time-varying volume datasets.

Our approach is based on the following idea: given two selected time steps $i$ and $j$, if the intermediate time steps that are skipped can be reconstructed from $i$ and $j$ using linear interpolation with a small difference in the amount of information (called *information difference*) from the original data, then the selected steps $i$ and $j$ can well represent the skipped time steps and thus are "salient" or "representative". Specifically, we apply information theory to quantify the amount of information difference using *variation of information (VI)*, which compares the reconstructed/interpolated time steps against the original data. Then, what if the information difference is too large and $i$ and $j$ are not representative enough? A natural choice would be to add another selected time step between $i$ and $j$ that is most under-represented by this pair, to lower the information difference. In fact, this is the method used in [WLS13] for selecting representative isovalues from scalar fields (albeit under a different measure of representativeness). However, such approach is just a greedy method and can be sub-optimal, since we would never try "backtracked" solutions that do not select $i$ or $j$. To achieve *globally optimal solutions*, we devise a *dynamic programming* algorithm for extracting the subset of time steps that optimally minimize the information difference.

It is important to point out that our dynamic programming algorithm is *fully automatic* without the need of any user interaction. Moreover, it is performed in the *preprocessing* phase, which is only done *once* with the results stored in a file. For example, we can let it run once overnight without supervision, and the results are ready to be used for user interaction. In the run-time phase, the user can either provide a desired number of time steps $k$, or give a threshold $\varepsilon$ in percentage (%) for the information difference[†] and let our algorithm choose a minimum $k$ satisfying $\varepsilon$; in either case our algorithm will select the best $k$ time steps very efficiently. To guide

[†] This percentage is with respect to the maximum possible total information difference; see Sec. 3.2.3 for the details.

the user on what values of $k$ or $\varepsilon$ to choose, we use a novel chart to present the dynamic programming results to the user, which shows a selection table (for each value of $k$, what are the best $k$ time steps selected) and a curve of information difference (for each value of $k$, what is the corresponding amount (in %) of information difference for the selected $k$ time steps); see Fig. 1 for an example (detailed in Sec. 3.2.4). Together, they serve as a storyboard of the data to guide the user through the time-step selection process.

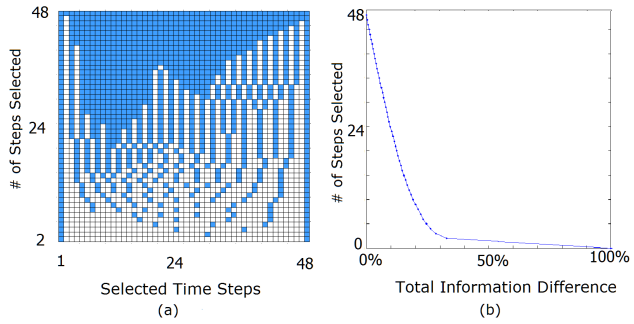


**Figure 1:** *Results of our in-core/accurate method (under VI) on Isabel-TC: (a) the Selection Table; (b) the curve of Total Information Difference.*

Our run-time algorithm works equally efficient in the out-of-core setting, where there are too many time steps to fit in-core. However, our basic preprocessing algorithm so far is in-core. When there are too many time steps to fit in-core, a direct adaptation would be prohibitively expensive due to the heavy I/O cost. We extend our basic approach to a novel out-of-core approximate algorithm to achieve *optimal I/O* (where the I/O cost is no more than reading the dataset *twice*), which also greatly reduces the in-core computing time and exhibits a nice trade-off between computing speed and accuracy.

We remark that previously most results for selecting representative time steps are based on local considerations and are suboptimal (see Sec. 2). In *dynamic time warping (DTW)* [TLS12], a dynamic programming is applied to achieve global optimality when selecting $k$ time steps for a user-specified $k$. However, the user is not provided with information on how to choose a suitable $k$. Also, the reconstruction of the skipped time steps is different. Moreover, the approach is in-core and does not consider the out-of-core setting. See Sec. 2 for more details.

**Contributions:** The contributions of this paper are as follows.
**(1)** We give a fully automatic preprocessing method using information theory and dynamic programming, to achieve globally optimal key time steps selection with minimum information difference.
**(2)** We provide an information-theoretic storyboard of the data to guide the user through the key-time-step selection process in the run-time phase, which can be performed extremely fast.
**(3)** We extend our preprocessing approach to a novel out-of-core approximate algorithm, using *sliding window* and *multi-pass dynamic programming*, to achieve optimal I/O and much faster in-core computation with a nice speed-accuracy trade-off. This method **significantly** improves the running time of the previous globally optimal DTW [TLS12] on out-of-core data with similar qualities, and is our major contribution.
**Remark:** Our algorithms actually work *independently* of the metric used, i.e., the VI can be replaced by any metric. If the user has a better domain knowledge, he/she can choose a metric that better identifies the desired features. Our metric of information difference (VI) is useful in general, since information theory could extract hidden/unknown salient data features [WS11]. In the experiments we evaluated our algorithms against other methods under both VI and root-mean-square error (RMSE).

## 2. Previous Work

In video processing, a closely related work is key frame selection from videos. In [LK02], a dynamic programming method is used to select key frames by maximizing an energy function. For many other results we refer to an excellent survey [HXL*11, Section II.B]. Another closely related problem is viewpoint, lighting, and more generally parameter selection in visualization, for which the results include [BS05, Gum02, LHV06, MAB*97, CM10]. Work on isosurface-topology analysis for selecting critical isovalues and time steps includes [SB06, TFO09]. Also, there has been a rich literature on information-theoretic visualization techniques; see the recent book [CFV*16] for an excellent survey.

For our problem of selecting salient time steps from time-varying volume datasets, there has been an active research. One class of approaches is to provide an overview of the data for the user to visually decide which time steps to select [WS09, LS08, AM07, AFM06]. However, such visual-inspection process by the user lacks quantitative measures and is also labor intensive; for large-scale datasets automatic techniques are more desirable. In [AFM06], similar time steps are grouped and one time step is selected from each group. In [WYM08], *importance curves* are computed to select those time steps that are most dissimilar from their previous one and are thus regarded salient. Again, these methods are based on local considerations and thus the solutions are not globally optimal. Other related techniques include those based on the Time Activity Curve (TAC) (e.g., [WS09, LS09b, LS09a]) and the *TransGraph* [GW11]. Recently, an *in-situ*, incremental time-step selection approach is given in [MLF*16], which (like ours) also uses a linear model to predict and interpolate between time steps. The main advantage is that it does not require the entire dataset, but the selection decision is local and not globally optimal.

In the recent *flow-based* approach [FE17], a *flow-based metric* is proposed to compute the distance between volumes, which can capture spatial features. (Although we use linear interpolation as it is simple and intuitive, we can also use other interpolations, and thus we could open that possibility by using a more advanced interpolation that considers spatial components.) The flow-based approach initially selects time steps from random samples of regular partitions of the whole time series, and progressively considers additional time steps by random sampling with the proposed probability distributions. This method is non-deterministic; it tries to handle large data efficiently by random sampling while not considering the full dataset or the skipped time steps. On the contrary, our methods are deterministic and compute optimal/near-optimal solutions from the full dataset. We handle large data by an out-of-core

approximate algorithm, which achieves close-to-optimal solutions with fast run-time and optimal I/O cost.

The technique most closely related to ours is *dynamic time warping (DTW)* using dynamic programming [TLS12]. Its dynamic programming part is similar to ours, but it defines different error/cost functions. Moreover, if $i$ and $j$ are two consecutive selected time steps and $r$ is a skipped step in between, then $r$ is "reconstructed" as either $i$ or $j$, rather than linearly interpolated between $i$ and $j$ as in our method. As mentioned, we provide additional advantages of presenting a storyboard to guide the user on how to choose a suitable value of $k$ (number of time steps to select) or $\varepsilon$ (threshold on the amount of information difference) and handling the out-of-core setting I/O-optimally.

## 3. Our Approach

As mentioned, our algorithms work *independently* of the metric used. In the following, we stay with information difference (VI) as the metric, and point out where to modify (see Remark at the end of Sec. 3.1.2) when other metrics are used.

Given a time-varying dataset of $T$ time steps, where each time step has data size $N$, our goal is to be able to select a subset of $k$ time steps that minimize the total information difference, for any $k \leq T$. As mentioned, in the preprocessing phase we use a dynamic programming algorithm to compute and store the necessary results. In the run-time phase, such results are presented to the user as a storyboard to guide the user through the query process. In the following, we first discuss information difference — its definition and how to compute it, then present our dynamic programming algorithm and the resulting storyboard. Finally we discuss how to extend the preprocessing algorithm to handle the out-of-core setting efficiently.

### 3.1. Information Difference

#### 3.1.1. Background of Information Theory

Our notion of information difference uses information theory. *Entropy* was introduced by Shannon for measuring the amount of information and uncertainty. Formally, for a discrete random variable $X$ with possible values $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ and the probability mass function $p(X)$, Shannon's entropy [Sha48] of $X$ is defined as

$$H(X) = -\sum_{x_i \in \mathcal{X}} p(x_i) \log_2 p(x_i) \tag{1}$$

To compute entropy of a volumetric data, usually we start from computing a histogram for the data. Then we treat the values of the dataset as a variable, with the distribution computed from the histogram. Given two discrete random variables $X$ and $Y$, the *conditional entropy* $H(X|Y)$ of $X$ given $Y$ is defined [CT91] as follows:

$$H(X|Y) = -\sum_{x_i \in \mathcal{X}, y_j \in \mathcal{Y}} p(x_i, y_j) \log_2 \frac{p(x_i, y_j)}{p(y_j)}. \tag{2}$$

where $\mathcal{Y}$ is the set of possible values of $Y$ and $p(x_i, y_j)$ is the probability that $X = x_i, Y = y_j$. This quantity $H(X|Y)$ indicates the remaining amount of information of $X$ given the knowledge/information of $Y$ (see Fig. 2). Similarly we can define the conditional entropy $H(Y|X)$. The *variation of information (VI)* $VI(X;Y)$ between $X$ and $Y$ is defined [CT91] as follows:

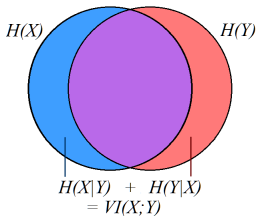


**Figure 2:** *Conditional entropy and variation of information (VI).*

$$VI(X;Y) = H(X|Y) + H(Y|X). \tag{3}$$

This quantity $VI(X;Y)$ indicates the amount of information in $X$ not presented by $Y$, plus the amount of information in $Y$ not presented by $X$ (see Fig. 2). In other words, it indicates the difference between the amount of information in $X$ and in $Y$, i.e., the information difference between $X$ and $Y$.

#### 3.1.2. Definition of Information Difference (InfoD)

To quantify the quality of a selected subset of time steps, we define a metric using variation of information (VI)[‡]. When some time steps are selected, the original data of the skipped time steps seems to be lost; however, we can reconstruct them by linear interpolation, and measure how much information is different after the reconstruction.

Formally, when two time steps $i$ and $j$ are selected, the *information difference (InfoD)* $c(i, j)$ for the pair $(i, j)$ is defined as follows: For each skipped time step $r$ between $i$ and $j$, we linearly interpolate from the data at time steps $i$ and $j$ to get an estimation, i.e. $X'_r = \text{Interpolate}(X_i, X_j, r)$. Then we compute the VI between the original data and the estimation at time step $r$.

$$Diff_r = VI(X_r; X'_r) \tag{4}$$

The intuition is as following: when time steps $i$ and $j$ are selected, the skipped time steps are believed to change smoothly between $i$ and $j$. We estimate the data $X'_r$ using the information of $X_i$ and $X_j$. The information difference (InfoD) between the original data $X_r$ and the estimated data $X'_r$ is $VI(X_r; X'_r)$. See Fig. 3.

Finally we define the InfoD $c(i, j)$ for the pair $(i, j)$ as the sum of such VI for all skipped time steps between $i$ and $j$. This stands for the InfoD if we select time steps $i$ and $j$ *while discarding all time steps in between*.

$$c(i, j) = \sum_{i<r<j} Diff_r. \tag{5}$$

‡ VI is indeed a metric, i.e., it satisfies the triangle inequality, non-negativity, identity of indiscernibles and symmetry [KSAG08].

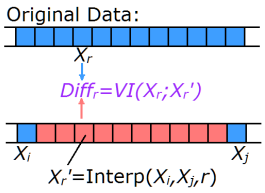


**Figure 3:** *Information difference (InfoD) at time step r when time steps i and j are selected.*

We further define $D^{(k)}(i)$ as the minimum InfoD for time steps 1 to $i$ when $k$ time steps are selected in the range $[1,i]$, *including* 1 and $i$. Note that $k \geq 2$. For the base case when $k = 2$, since we must only choose 1 and $i$, we have $D^{(2)}(i) = c(1,i)$.

**Remark:** When a different metric is used, we only need to modify Eq. (4). E.g., for root-mean-square error (RMSE), Eq. (4) becomes $Diff_r = RMSE(X_r;X_r') = \sqrt{\sum(d_p - d_p')^2/N}$, where $d_p, d_p'$ are the scalar values of a grid point in $X_r$ and $X_r'$ and there are $N$ grid points.

#### 3.1.3. Initial Computation for Information Difference

Recall that $N$ is the size of the data at each time step, and $T$ the total number of time steps. To facilitate dynamic programming, we need to first perform an *initial computation* to compute $c(i,j)$ for *every pair* $(i,j)$, $1 \leq i < j \leq T$, and store them into a 2D table. The total running time for this task is $O(T^3N)$, which turns out to be the dominating run-time bottleneck of our approach. In Sec. 3.3 we discuss how to reduce this bottleneck in the out-of-core setting.

### 3.2. Main Time-Step Selection Algorithm

We assume that the **first** and **last** time steps must always be chosen so that our linear interpolation can be done cleanly. Recall that $D^{(k)}(i)$ is defined as the minimum InfoD for time steps 1 to $i$ when $k$ time steps are selected in the range $[1,i]$, including 1 and $i$, where $k \geq 2$. Also, we have $D^{(2)}(i) = c(1,i)$. Then our task is to compute the minimum InfoD $D^{(k)}(T)$, while identifying which $k$ time steps to select to achieve such $D^{(k)}(T)$. To obtain solutions for all possible values of $k$, we run $D^{(k)}(T)$ for $k$ from 2 up to $T$.

#### 3.2.1. Dynamic Programming

To compute $D^{(k)}(i)$, we have the following recurrence:

$$D^{(k)}(i) = \min_{1<p<i} \{D^{(k-1)}(p) + c(p,i)\}. \tag{6}$$

We apply dynamic programming to solve the recurrence by memorizing all the valid $\{i,k\}$ tuples as states in a table. The correctness of dynamic programming can be easily proved since this problem exhibits the properties of overlapping subproblems and optimal substructure. As shown in Eq. (6), to select $k$ time steps in the range $[1,i]$, we try every time step $p$ $(1 < p < i)$ such that $p,i$ are the last two time steps among the selected time steps, and find the one that has the minimum total InfoD.

The dynamic programming memoization table stores the states for all $\{i,k\}$ tuples, where $1 \leq i \leq T$, $2 \leq k \leq T$. Thus the computation takes $O(T^3)$ time and uses $O(T^2)$ space. The total time for the preprocessing, including the Initial Computation in Sec. 3.1.3 ($O(T^3N)$ time) and dynamic programming, is $O(T^3N + T^3)$.

#### 3.2.2. Selection Table

With the memoization table, we can directly answer such query as: given a number $k$, which subset of $k$ time steps selected is the best. We just need to store the time steps that are selected when building the table. To answer the query, we construct the set of selected time steps by tracing back from $D^{(k)}(T)$.

For all $2 \leq k \leq T$, we obtain the sets of selected time steps; we put them in a *Selection Table* as shown in Fig. 1(a). Each row represents a set of $k$ selected time steps. From left to right are time steps 1 to $T$, with the selected time steps marked **blue**.

#### 3.2.3. Total Information Difference

The selected time steps do not yield any InfoD. Each unselected time step $r$ will produce an InfoD as computed by Eq. (4) and contribute to the *Total Information Difference (Total InfoD)*, $Diff_{\text{total}}$, given by

$$Diff_{\text{total}} = \sum_{r \text{ is skipped}} Diff_r = \sum_{1 \leq r \leq T} Diff_r. \tag{7}$$

The last equation is true because $Diff_r = 0$ for selected steps $r$.

The dynamic programming gives us, for each value of $k$, the best $k$ selected time steps and their optimal Total InfoD $Diff_{\text{total}}$. Now we want to express such total information difference in terms of a percentage $\eta$, so that given a query threshold $\varepsilon$ (in %) we can find the smallest $k$ whose $\eta$ satisfies $\varepsilon$. For this, we want to divide $Diff_{\text{total}}$ by a quantity $Q$ where $Diff_{\text{total}} \leq Q$ and $Q$ is *independent* of the selection results of any $k$.

By the definitions of entropy and conditional entropy (see Sec. 3.1.1), we have $H(X|Y) \leq H(X)$ and $H(Y|X) \leq H(Y)$ (see Fig. 2), and thus $VI(X;Y) = H(X|Y) + H(Y|X) \leq H(X) + H(Y)$. For a particular time step $r$, let $X = X_r$ denote the original data and $Y$ denote the data from our selection result (i.e., $Y = X_r$ if $r$ is selected and $Y = X_r'$ if $r$ is skipped), then $Diff_r \leq H(X_r) + H(Y)$. To make $H(Y)$ *independent* of any selection results, observe that $H(Y)$ has the maximum value when each value of $Y$ has the same probability $1/n_b$, where $n_b$ is the number of histogram bins. By Eq. (1), we have $H(Y) \leq \log_2 n_b$. Putting these together, we have $Diff_r \leq H(X_r) + \log_2 n_b$, and thus

$$Diff_{\text{total}} = \sum_{1 \leq r \leq T} Diff_r \leq \sum_{1 \leq r \leq T} (H(X_r) + \log_2 n_b) = H_{\text{total}} + T \log_2 n_b, \tag{8}$$

where

$$H_{\text{total}} = \sum_{1 \leq r \leq T} H(X_r). \tag{9}$$

We define the *maximum possible total InfoD* as

$$Max\_Diff_{\text{total}} = H_{\text{total}} + T \log_2 n_b. \tag{10}$$

Note that $Max\_Diff_{\text{total}}$ is independent of the selection results and is $\geq Diff_{\text{total}}$ of any $k$.

Finally, we define the percentage of total information difference:

$$\eta = \frac{Diff_{\text{total}}}{Max\_Diff_{\text{total}}} \times 100\%.$$

We are able to answer such query as: given a percentage $\varepsilon$ as a threshold, what is the minimum number $k$ of time steps needed to guarantee the information difference at most $\varepsilon$, and what are such optimal $k$ time steps. (The query takes time $O(\log T + k)$ by a binary search on a table storing for each $k$ its corresponding best $\eta$ value.)

We plot the curve of percentage of Total Information Difference (Total InfoD) for every $k$, shown in Fig. 1(b). The InfoD is 100% if nothing is selected. Note that selecting the first and last time steps already improves the InfoD to 32%. As more time steps are selected, the InfoD is reduced.

**Remark:** For a different metric, $Max\_Diff_{\text{total}}$ must be derived to modify Eq. (10) for using $\eta$ in %. For RMSE, we skip the derivation and just use the total RMSE (Eq. (7), *not* in %) for the curve.

#### 3.2.4. Understanding the Plots

The Selection Table and the Total Information Difference Curve are part of our results. The vertical axes of both plots are the number of selected time steps, aligned to each other (see Fig. 1). We intentionally align the plots in this way, so the user can read a row horizontally to obtain the selection of time steps as well as its corresponding InfoD in percentage. Moreover, the distributions of selections in the table and the shapes of the curves are also informative features. More examples are shown in Sec. 4.

### 3.3. Approximate Approach

Our preprocessing algorithm so far takes $O(T^3N + T^3)$ time (see Sec. 3.2.1 at the end). Note that the major bottleneck is the Initial Computation step in Sec. 3.1.3, i.e., to compute the table of InfoD $c(i, j)$ for every pair $(i, j)$ of time steps. Moreover, when dataset cannot fit in-core, a direct adaptation would be to read from disk to main memory every time step needed for computing the InfoD between the interpolated and original data. This would require $O(T^3(N/B))$ disk reads where $B$ is the number of items fitting in one disk block. Such high I/O cost will make the algorithm extremely slow. To overcome this difficulty, we devise a novel approximate method that runs much faster in terms of *both* I/O and in-core computing, with acceptable loss of accuracy.

#### 3.3.1. Intuition

When computing InfoD $c(i, j)$ for all pairs $(i, j)$, if $i$ and $j$ are too far away, there are many time steps in between and it is expensive to compute their InfoD (see Eq. (5)). However such $c(i, j)$ is rarely used, as it does not make much sense to select two far-away time steps while skipping everything in between. Thus we try to reduce the cost of computing $c(i, j)$ for far-away pairs $(i, j)$, by estimating the value of such $c(i, j)$ from what we already computed.

#### 3.3.2. Key Ideas

**Overview.** Let $t < T$ be the number of time steps to fit in-core ($t$ is a parameter for how much main memory to use). Our solution is a multi-pass method; in each pass we use a *sliding window W* of size $t$ time steps to slide through the current set $S$ of time steps to be processed. Initially, $S$ has all time steps and $|S| = T$. We want to reduce $|S|$ in each pass so that eventually $|S| \le t$ and the data of time steps in $S$ can fit in-core. For efficiency, we like the $|S|$ values over the passes to form a *geometric series* (e.g., $T, T/2, T/4, \cdots$) so that they sum up to $O(T)$, i.e., the total work over *all passes* is still asymptotically the same as the work of *only the first pass*. So in each pass we reduce $|S|$ by *half*. To do so, in each pass we will run dynamic programming for $k$ up to $|S|$, and select the best $|S|/2$ time steps as *representatives* to be put to $S$ for the next pass. For ease of exposition, we first discuss the key ideas by going through the first two passes.

**First Pass.** In the first pass, $S$ is $\{1, 2, \cdots, T\}$. We slide the window $W$ through $S$ as below. To start, we read the data of the first $t$ time steps of $S$ $(1, \cdots, t)$ in core. We then compute $c(1, j)$ as before (Eq. (5)) for each $j \le t$ (such steps $j$ are all in core), and set $c(1, j) = +\infty$ for the remaining $j$. Next we slide $W$ one position, keeping time steps $2, \cdots, t+1$ in core. Similarly, we compute $c(2, j)$ using Eq. (5) for each $j \le t+1$ and set $c(2, j) = +\infty$ for the remaining $j$.

Repeat this process to the end; for the *last* window $W$, we compute $c(i, j)$ for *all pairs* $i, j \in W$. Then $c(i, j)$ has an "actual value" (i.e., non-infinity) for $j - i + 1 \le t$ (the window size), and $c(i, j) = +\infty$ otherwise. Finally, we perform dynamic programming *using the table with the current* $c(i, j)$ *values* for $k$ up to $|S| = T$. We use the dynamic programming results to select the best $T/2$ time steps, say $i_1, i_2, \cdots, i_{T/2}$, and let $S = \{i_1, i_2, \cdots, i_{T/2}\}$. (Key idea: For far-away pairs $(i, j)$ that $W$ cannot accommodate, $c(i, j) = +\infty$ and such values would not be picked/used during dynamic programming to minimize the cost.) Now we proceed to the second pass. Note that the time steps in $S$ are *no longer consecutive*.

**Second Pass.** In the second pass, again we slide $W$ through $S$. To start, we read the data of the first $t$ time steps of $S$, $i_1, i_2, \cdots, i_t$, in core. Now the task is to compute $c(i_1, j)$ for $j = i_2, \cdots, i_t$ (such steps $j$ are in core). In the process, if $c(i_1, j)$ is already non-infinity, then we leave the value intact without trying to update it. This is because it already has an actual value computed in an earlier pass where $W$ accommodated a smaller range of time steps (i.e., with smaller gaps) and hence the already computed value is more accurate. In summary, we only compute $c(i_1, j)$ (for $j = i_2, \cdots, i_t$) if it is currently $+\infty$.

**Property 1:** $c(i_1, i_2)$ *already has a non-infinity value.*

**Proof:** Since among the $T/2$ selected time steps from dynamic programming, $i_1$ and $i_2$ are consecutive and thus $c(i_1, i_2) \ne \infty$ (see Eq. (6): if $p$ is selected, then $c(p, i)$ cannot be $\infty$). □

By Property 1, for $c(i_1, j)$ that we need to compute, there must be some other time step(s) $i_r$ in $W$ in between $i_1$ and $j$. (The above proof is valid for any consecutive time steps $i_m, i_{m+1}$ in $W$, not just for $i_1, i_2$.)

To simplify the notation, let $i_1 = i$ and $i_r = r$ and consider Fig. 4, where the blue squares in the second row denote the time steps currently inside $W$ (and thus in core) while the white squares denote the time steps skipped and not in $W$, and we want to compute $c(i, j)$. We do *not* want to read these white-square time steps; instead, we use the closest blue-square time step (other than $i$ and $j$) to represent them. For example, in Fig. 4, the 5 time steps in the second row

that are underlined by a purple line (including the blue square $r$) are all represented by time step $r$, i.e., we pretend that they all have $r$ as the original data. In addition, to speed up the computation, we do *not* interpolate for each such individual white-square time step, but instead use the interpolated time step $r$, $X_r' = \text{Interpolate}(X_i, X_j, r)$ (see Fig. 4), as the representative interpolated result for all such 5 underlined time steps. Thus the contribution of these 5 time steps to $c(i,j)$ is $5 \cdot Diff_r = 5 \cdot VI(X_r; X_r')$. Note that time step $r$ is in core and we can compute $Diff_r$ easily.

More generally, if $n_r$ time steps (including time step $r$) are represented by time step $r$ ($n_r = 5$ in the example), then $n_r \cdot Diff_r$ is contributed to $c(i,j)$. In case some white-square (i.e., missing) time step is of the equal closest distance to two blue-square time steps $r$ and $s$, then it is *represented by r and s half and half*, contributing 0.5 to both $n_r$ and $n_s$ when computing $n_r \cdot Diff_r$ and $n_s \cdot Diff_s$. So $n_r$ ($n_s$) can be a multiple of 0.5. In this way, we compute $c(i,j)$ when there are gaps/missing time steps between $i$ and $j$.

We proceed to slide $W$ through $S$ and repeat the same process. For the *last* $W$, we compute $c(i,j)$ for *all pairs* $i, j \in W$. Finally we perform a dynamic programming *using the table with the current* $c(i,j)$ *values*, now for $k$ up to $|S| = T/2$. Again we select from the dynamic programming results the best $|S|/2 = T/4$ time steps, set them as the elements of $S$, and go to the next pass.

**Remaining Passes.** We reduce $|S|$ by half after each pass. Finally $|S| \le t$ and the time steps in $S$ can all fit in $W$ in core. We compute $c(i,j)$ (for $i, j \in S$), run dynamic programming, and stop.

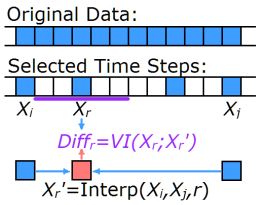


**Figure 4:** *Approximation of information difference.*

### 3.3.3. Multi-Pass Algorithm

Let $S$ be the set of time steps to be processed in the current pass. Initially $S$ is $\{1, \cdots, T\}$ and $|S| = T$. We have a global memoization table $D$ for dynamic programming, keeping the values $D^{(k)}(i)$ for all tuples $\{i,k\}$ in Eq. (6). We also have a global table $C$ storing the current values of $c(i,j)$. Below is a pseudo-code of the algorithm.

In each pass, the algorithm will only work on the time steps in $S$. There are three steps in each pass:

**Step 1:** Use a sliding window $W$ of $t$ time steps to slide through $S$ and compute $c(i,j)$ as discussed in Sec. 3.3.2.

**Step 2:** Perform dynamic programming as described in Sec. 3.2.1, *using the table C with the current* $c(i,j)$ *values*. The memoization table $D$ is built/updated as before using Eq. (6); however, the $D^{(k)}(i)$ value (and the corresponding time steps chosen to realize the optimal) is updated only when the new value is better. The variable $k$ (in Eq. (6)) loops from 2 to $|S|$.

**Algorithm:** Multi-Pass Algorithm

```
S ← {1 ··· T};                    ▷ |S| = T; S[i] means the i-th item in S.
loop
    w ← min{|S|, t};              ▷ w is the sliding window size |W|.
    for i ← 1 to |S| − 1 do
        for j ← i + 1 to min{i + w − 1, |S|} do
            update c(S[i], S[j]);
        end for
    end for ▷ W = {S[i] ··· S[i + w − 1]} for i = 1 ··· |S| − w + 1.
    Run dynamic programming using the current c() table (C);
    if |S| ≤ t break;
    S ← best selection of |S|/2 time steps in S;    ▷ |S| is halved.
end loop
```

**Step 3:** If $|S| \le t$, we are done. Else, select the $|S|/2$ most representative time steps using the dynamic programming result of Step 2, let $S$ be the set of these time steps, and start the next pass from Step 1, where $|S|$ has been halved.

In Step 1 of the next pass, we will update table $C$ of $c(i,j)$, since we can now compute for the $(i,j)$ pairs which were far away in the previous pass but now closer in the new $S$. Also, each pass improves the $D^{(k)}(i)$ values and the corresponding time-step selections, so that at the end the best solutions are maintained in the table $D$.

In each pass, we spend $O(t^2|S|N)$ time to update the $c(i,j)$ values, then use $O(T^2|S|)$ time to run the dynamic programming. In each pass we reduce $|S|$ in half, until $|S| \le t$. So the number of passes is bounded by $O(\log_2(T/t))$. Also, in each pass we only run $k$ up to the current $|S|$ (rather than to $T$), so the total time for dynamic programming from all passes is $O(T^2(T + T/2 + T/4 + \cdots)) = O(T^3)$. The total time to update the $c(i,j)$ values from all passes is $O(t^2N(T + T/2 + T/4 + \cdots)) = O(t^2TN)$. The overall in-core running time of the approximate algorithm is $O(t^2TN + T^3)$.

### 3.3.4. The I/O Issues

As said before, for large datasets that cannot fit in-core, the accurate approach in Secs. 3.1 and 3.2 will be extremely slow due to the heavy I/O cost, where a straightforward method would require $O(T^3(N/B))$ disk reads. In our approximate method, in each pass we use a sliding window to slide through the set $S$ and read the volumes, with I/O cost $O(|S| \cdot N/B)$. Since $|S|$ is halved every pass, the total number of I/O reads is bounded by $(N/B)(T + T/2 + T/4 + \cdots) \le 2T(N/B) = O(T(N/B))$. It is equivalent to reading the whole dataset *at most twice*.

### 3.3.5. Approximation of Information Difference

The InfoD $c(i,j)$ computed by our approximate approach is not exact. Recalling from Sec. 3.3.2 and Fig. 4, we compute the contribution to $c(i,j)$ from a time step $r \in W$ (where $W$ is the sliding window), $r$ in between $i$ and $j$, as $n_r \cdot Diff_r$. Since the resulting $c(i,j)$ is just an approximation, we denote it by $\hat{c}(i,j)$ instead. Formally, $\hat{c}(i,j)$ is computed using Eq. (11), where $n_r$ is the number of time steps that are closest to $r$ than any other time steps in $S$ (or equiv-

**Table 1:** *Test Datasets.*

| Dataset | Size | Dimensions | TimeSteps | DataType |
|---|---|---|---|---|
| *Synthetic-1* | 96 KB | 10x10x10 | 24 | Float |
| *Synthetic-2* | 96 KB | 10x10x10 | 24 | Float |
| *Isabel-TC* | 4.46 GB | 500x500x100 | 48 | Float |
| *Vortex* | 784 MB | 128x128x128 | 100 | Float |
| *TeraShake* | 23.7 GB | 750x375x100 | 227 | Float |
| *Radiation* | 27.4 GB | 600x248x248 | 200 | Float |
| *Radiation2* | 54.8 GB | 600x248x248 | 400 | Float |
| *Radiation4* | 109.6 GB | 600x248x248 | 800 | Float |
| *Radiation8* | 219.2 GB | 600x248x248 | 1600 | Float |

alently, in $W$). Recall from Sec. 3.3.2 that $n_r$ can be a multiple of 0.5, and that $n_r = 5$ in Fig. 4.

$$\hat{c}(i,j) = \sum_{i<r<j, r\in S} n_r \cdot Diff_r. \tag{11}$$

We argue that, using Eq. (11), $\hat{c}(i,j)$ is an upper bound of the InfoD computed from the original data using Eq. (5). Since the time steps in $S$ are those selected from the previous pass, they should be the most important ones, i.e., we have chosen them to minimize the InfoD. This means that the InfoD/VI at those time steps are higher than their neighboring time steps, and thus using them to represent their neighbors in Eq. (11) gives an upper bound. The $Diff_{\text{total}}$ (see Eq. (7)) from our dynamic programming results also becomes an upper bound. Our experiments supported these arguments (see Sec. 4.3). Note that $Max\_Diff_{\text{total}}$ (in Eq. (10)) remains *exact*, so that the resulting percentage $\eta$ is *conservative*.

## 4. Results

We have implemented our approaches in C++ and run our experiments on a PC with one 2.7-3.7GHz Intel i7-3740QM Quad-Core CPU, 16GB RAM, nVidia GeForce GTX 680M graphics card, and Linux OS. The images were rendered using the VTK [Kit03] library. The statistics of the test datasets are shown in Table 1. For each dataset, we need to scan it once to compute the value range if that is unknown in advance. Then the value range is subdivided evenly among the histogram bins. In Appendix I (Supplementary Materials), we show that our results are *not sensitive* to various bin numbers like 64, 128, 256 and 512, and we fixed the bin number to 128 throughout. Also, the run-time queries to select the desired time steps took *almost no time*. The actual time for the complete query includes only reading each selected time steps from disk and performing volume rendering.[§] Thus we do not report such timing results in run-time phase.

### 4.1. In-Core/Accurate Results

For small datasets (Isabel-TC and Vortex), we can afford to run our accurate method. We also ran the approximate method and compared their performance; see Table 2. The analysis of approximation errors will be discussed in Sec. 4.3.

[§] We used a generic transfer function to map scalar values linearly to colors and opacity. As said in the Remark at the end of Sec. 1, our InfoD metric could extract unknown features. If the user has a better domain knowledge, he/she can use a metric that better incorporates the transfer function.

**Table 2:** *Results for small datasets (under VI). N: # grid points at each time step; T: # time steps in the dataset; t: # time steps kept in core; Runtime: total runtime in* ***seconds****; I/O: I/O time in* ***seconds****; DP: dynamic programming time in* ***milliseconds****; $\hat{c}$-time: time to compute $\hat{c}()$ in* ***seconds****; Mem: memory footprint; NRMSE: normalized root-mean-square error; Est-Err: estimated error. (For NRMSE and Est-Err see Sec. 4.3).* ***The accurate method is when*** *$t = T$, where $c()$ is computed instead of $\hat{c}()$.*

| Dataset | $T$ | $t$ | Runtime | I/O | DP | $\hat{c}$-time | Mem | NRMSE | Est-Err |
|---|---|---|---|---|---|---|---|---|---|
| *Vortex* | 100 | 100 | 3713 | 8.9 | 1.6 | 3704 | 835MB | 0% | 0% |
| (784 MB) | 100 | 35 | 1254 | 9.5 | 3.3 | 1244 | 315MB | 0.19% | 0.38% |
| $N = 2.1$M | 100 | 10 | 138 | 8.4 | 4.4 | 129.6 | 115MB | 0.43% | 0.92% |
| *Isabel-TC* | 48 | 48 | 4908 | 45 | 0.3 | 4863 | 4.69GB | 0% | 0% |
| (4.46GB) | 48 | 20 | 2061 | 46 | 0.8 | 2015 | 2.02GB | 0.32% | 2.46% |
| $N = 25$M | 48 | 8 | 430 | 46 | 1.0 | 384 | 0.87GB | 0.62% | 4.07% |

**Dataset: Hurricane Isabel** The Hurricane Isabel data [WBK04] simulates an actual hurricane in September 2003. We only used the temperature scalar field from the original multivariate data.

Fig. 1 shows the results of our accurate method under the VI metric. We see in (a) that the data has more interesting features in the early part around time steps 5 to 20. Fig. 1(b) plots the curve of percentage of total InfoD. As seen, the InfoD drops quickly when selecting only a few time steps, but slows down as $k$ increases. This can be because there is a lot of redundant or implicit information between consecutive time steps. Selecting only a few time steps will provide a good representation of the dataset. Using the two plots, users would be able to decide what number of time steps to choose. One can decide the upper bound percentage of information difference, say $\varepsilon = 20\%$. By looking up the curve, our approach can tell that 8 time steps are enough to have an InfoD of at most 20%, along with the best selection of the time steps. The results exhibit a nice storyboard of the dataset. We show the rendering of selecting 5 and 8 time steps in Fig. 20 in Appendix III (Supplementary Materials). Similar storyboard results but under RMSE are given in Fig. 21 in Appendix III.

**Dataset: Vortex** The Vortex dataset was generated by pseudo-spectral simulations of continuous turbulent vortex structures. The scalar values denote the vorticity. Fig. 5 shows the results of our accurate method under VI. Unlike Isabel, Vortex contains information that is more uniformly distributed over all time steps. From Fig. 5(a) we see that the selection of time steps are likely to spread evenly, i.e., each row of selections is likely to have equally spaced blue blocks. The curve in (b) is closer to linear than any other datasets. These characteristics coincide with the results of other researches [MSSS98, WYM08]. We show the rendering of our selected 10 time steps in Fig. 22 in Appendix III. Similar storyboard results but under RMSE are given in Fig. 23 in Appendix III.

### 4.2. Out-of-Core/Approximate Results

For large datasets we can only afford to run our approximate method. Its performance results are shown in Tables 2 and 3. (The results with more $t$ values are also shown in Fig. 12.) The memory footprint is $O(tN)$. The I/O times were about the same for different $t$, while the total runtime grew very fast with $t$. (In Table 2 the inputs were small enough to be cached in-core so the I/O times were

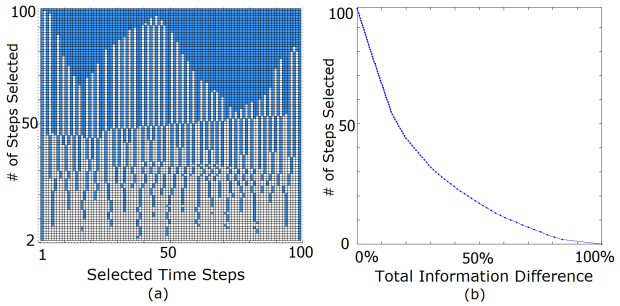


**Figure 5:** *Results of our in-core/accurate method (under VI) on Vortex.*

**Table 3:** *Results for large datasets (under VI). N: # grid points at each time step; T: # time steps in the dataset; t: # time steps kept in core; Runtime: total runtime in* ***seconds****; I/O: I/O time in* ***seconds****; DP: dynamic programming time in* ***milliseconds****; $\hat{c}$-time: time to compute $\hat{c}()$ in* ***seconds****; Mem: memory footprint; Est-Err: estimated error (see Sec. 4.3).*

| Dataset | $T$ | $t$ | Runtime | I/O | DP | $\hat{c}$-time | Mem | Est-Err |
|---|---|---|---|---|---|---|---|---|
| *TeraShake* | 227 | 12 | 5482 | 376 | 38 | 5106 | 1.45GB | 22.3% |
| (23.7GB) | 227 | 8 | 2425 | 365 | 38 | 2060 | 1.02GB | 32.9% |
| $N = 28$M | 227 | 4 | 752 | 360 | 39 | 392 | 0.60GB | 57.2% |
| *Radiation* | 200 | 12 | 7563 | 487 | 26 | 7076 | 1.91GB | 3.15% |
| (27.4GB) | 200 | 8 | 3336 | 470 | 28 | 2866 | 1.35GB | 7.66% |
| $N = 37$M | 200 | 4 | 923 | 438 | 29 | 485 | 0.79GB | 25.1% |

the same as reading the input once; in Table 3 the I/O times were about twice of reading the inputs.) Observe that the in-core computing time ($O(t^2TN + T^3)$, Sec. 3.3.3) was much higher than the I/O time. For the former, the dynamic programming (DP) time is $O(T^3)$, which is *negligible* compared to the $O(t^2TN)$ term. The latter is the time to compute $\hat{c}()$ ($\hat{c}$-time), and is *linear* in $TN$ (dataset size) for a fixed $t$. For the large datasets we fixed $t$ to 12.

**Dataset: TeraShake** This data represents a physics-based simulation of a magnitude 7.7 earthquake on the Southern San Andreas Fault [ODM*04]. The original data is a vector field of velocity; we computed the magnitude of the velocity as a scalar field. In the simulation, the rupture begins at time step 1 and continues up to 55. Then the rupture stops but the earthquake wave continues to propagate. Around time step 70, the wave starts hitting Los Angeles Basin and makes the Basin to be another wave source. After time step 140 there is almost no activity; in the dataset the time steps after 140 are basically empty and contain no features.

In Fig. 6(a)(b) we show the results of our approximate method under VI. In the Selection Table (a), the top part indicates selecting many time steps. Some time steps are outstanding as they were always selected, but they are not favorable when choosing fewer time steps. This is due to our multi-pass process (Sec. 3.3.3). In the first pass, since $t = 12$, it could not select time steps with a gap more than $t - 2 = 10$ steps. In later passes, it found that time step 140 was much more critical, and that those after 140 were less important and could be discarded. This matches the description of the dataset. Also, the Selection Table suggests to pay attention to data around time step 70, which matches the simulation setting.

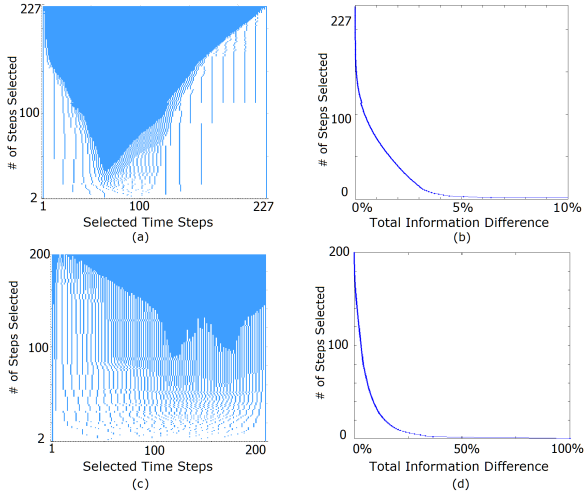


**Figure 6:** *Results of our approximate method under VI ($t = 12$) on (a)(b) TeraShake and (c)(d) Radiation.*

We show in Fig. 7 the rendering results of our selection of 10 time steps, where time steps close to the important events (up to and around 55, around 70, and up to 140) were selected, and those after 140 were skipped (until the last one). (Ignore Figs. 8–10 for now.)

We also show similar storyboard results of our approximate method but under RMSE in Fig. 24 in Appendix III.

**Dataset: Radiation** The Radiation [WN08] data was produced by numerical simulations of ionization front instability to understand the formation of galaxies. We performed our experiments on the temperature field. In the rendering results (Fig. 11) of our selection of 10 (out of 200) time steps under VI, we can observe how the fingers of radiation with high temperatures grow out, evolve and finally disappear. Scientists are interested in the ionized gas whose temperature is around 20,000 Kelvin. Our selected time step 29 presents the early stage of the phenomena, where fingers of radiation break through cracks in the shock. Our selected time steps 58 and 80 depict the process when the ionization front pushes forward. Time step 102 is selected just before the fingers hit the volume boundary (front face). The selected time steps 117 and 135 show the cross sections of the fingers at the front face where the internal structures can be observed. With the timeline indicating the rendered time steps shown on the top, the user can easily obtain an overview of the movements over time.

The selection table (under VI) in Fig. 6(c) suggests to pay more attention to the time steps 120 to 160 — this is the period when the fingers cross over the volume boundaries and disappear. From the curve in Fig. 6(d) we know that selecting about 10 to 20 time steps will be sufficient to represent the data; selecting more time steps will not reduce the Total InfoD significantly.

In the Supplementary Materials we include a video clip showing an animation of all 200 time steps (12 frames/time steps per second) of Radiation while at the bottom our 10 selected time steps are incrementally added when reached in the animation. (The (GPU-based) volume rendering of these 200 time steps took 21 minutes,

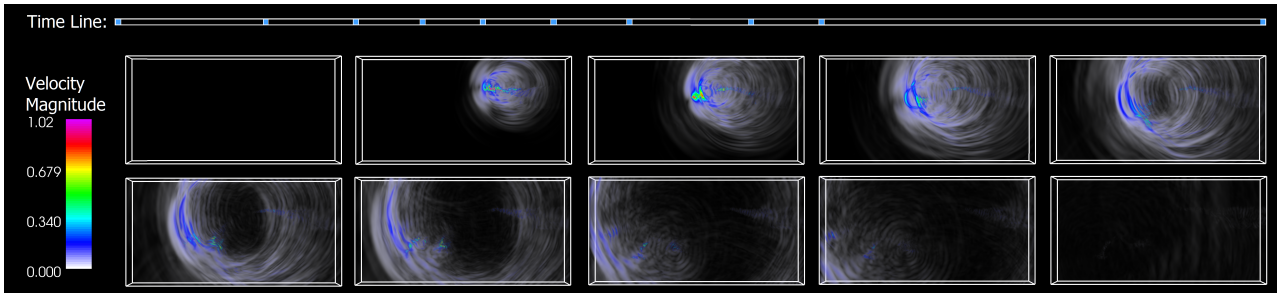


**Figure 7:** *Selecting 10 time steps from TeraShake using our approximate method under VI ($t = 12$). The selected time steps {1, 31, 48, 61, 73, 87, 102, 125, 140, 227} are shown in row-first order.*

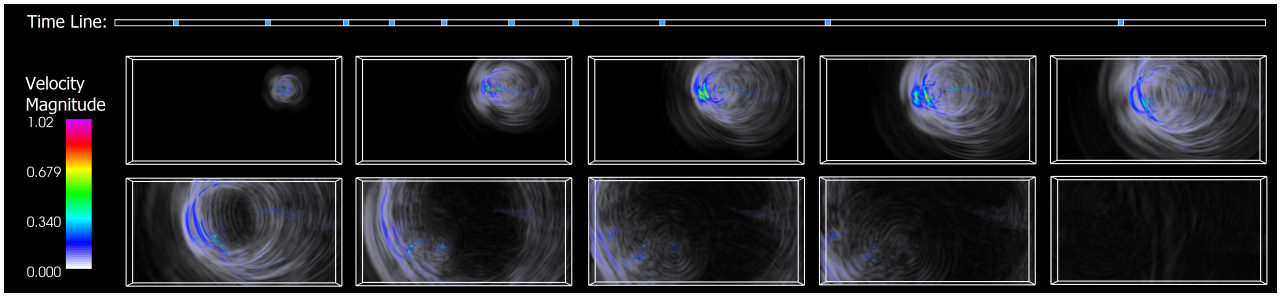


**Figure 8:** *Selecting 10 time steps from TeraShake using DTW under VI. The selected time steps {13, 32, 45, 54, 65, 79, 92, 107, 138, 199} are shown in row-first order. The selection quality and ours (Fig. 7) are similar but DTW took much longer time (see Figs. 15(b), 16(b)).*

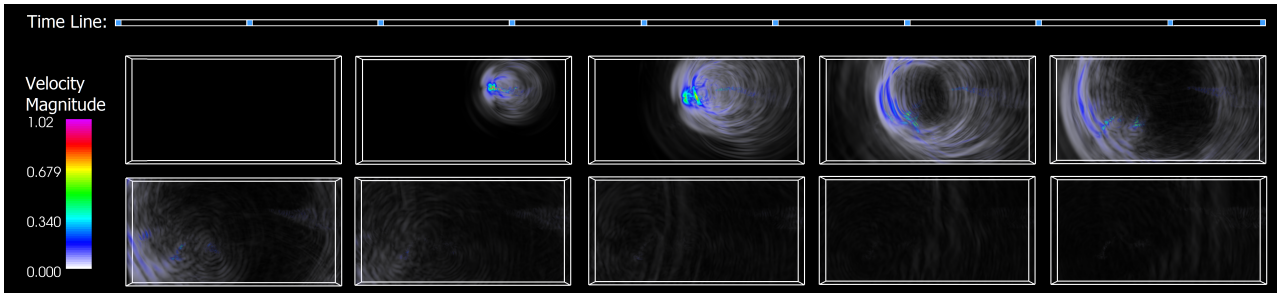


**Figure 9:** *Selecting 10 time steps from TeraShake using uniform sampling under VI. The selected time steps {1, 27, 53, 79, 105, 131, 157, 183, 209, 227} are shown in row-first order. The selection quality is much worse than ours in Fig. 7 (see Fig. 15(b)).*

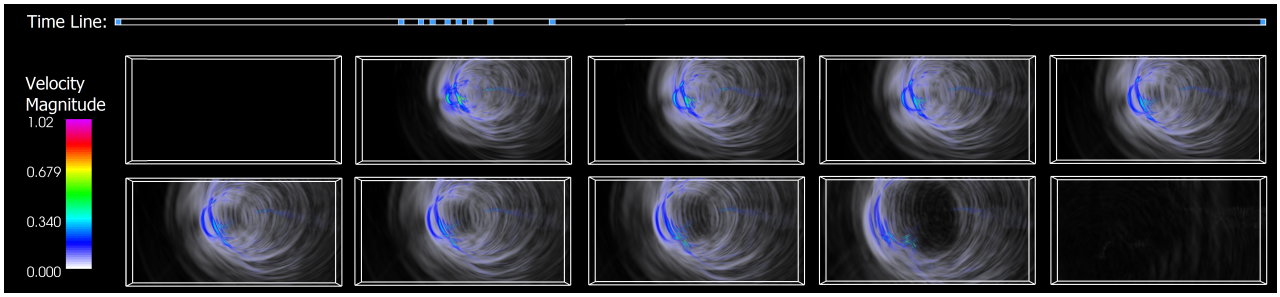


**Figure 10:** *Selecting 10 time steps from TeraShake using* Greedy *under VI. The selected time steps {1, 57, 61, 63, 66, 68, 70, 74, 86, 227} are shown in row-first order. The selection quality is much worse than ours (Fig. 7) with much slower running time (see Fig. 15(b)).*

including 7 minutes of I/O to read each time-step data from disk and write the resulting images to disk. The 10 selected time steps took total time $21 \cdot (10/200) = 1.05$ minutes.)

Similar storyboard results using our approximate method but under RMSE are given in Fig. 25 in Appendix III.

### 4.3. Errors of Approximation in the Out-of-Core Method

In the approximate method, the errors were introduced when computing $\hat{c}(i,j)$ using Eq. (11) (Sec. 3.3.5). The error is affected by the parameter $t$ (see Sec. 3.3).

#### 4.3.1. Comparing with Accurate Results

For small datasets we can afford to run the accurate method to obtain the ground truth, to be compared with the approximate results. We measured the error by computing the normalized root-mean-square error (NRMSE) for all the approximate values $\hat{c}(i,j)$:

$$NRMSE = \frac{\sqrt{\frac{\sum(\hat{c}(i,j)-c(i,j))^2}{n_c}}}{c_{max} - c_{min}}, \tag{12}$$

where $[c_{min}, c_{max}]$ is the value range of $c(i,j)$, and $n_c$ is the number of approximations. The experimental results are shown in Fig. 12.

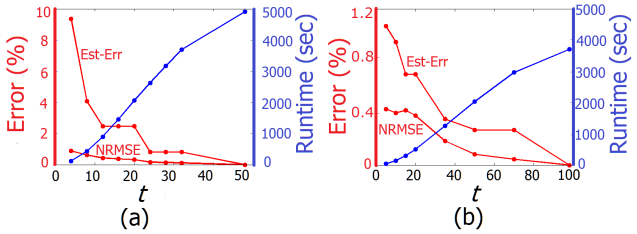


**Figure 12:** *The NRMSE, estimated error (Est-Err) and runtime (under VI) for small datasets: (a) Isabel-TC; (b) Vortex.*

#### 4.3.2. Estimating the Error

For large datasets, it is infeasible to run the accurate method. We still want to estimate the errors of the approximate method quickly and decide a suitable $t$. We compute $c(1,T)$ in $O(TN)$ time and $O(TN/B)$ I/O cost (Eq. (5)), and estimate the error by

$$\text{Est-Err} = \frac{|\hat{c}(1,T) - c(1,T)|}{c_{max} - c_{min}}. \tag{13}$$

Recall that we compute $\hat{c}(i,j)$ as an estimation of $c(i,j)$ using a subset of data in core (Eq. (11)). Each missing time step (white square in Fig. 4) contributes to the error $|\hat{c}() - c()|$; intuitively, the more missing time steps, the more error. As the range $[1,T]$ has the most missing time steps, we use Eq. (13) as a reasonable, easy-to-compute heuristic replacement of NRMSE to estimate the error.

The runtime and Est-Err are shown in Fig. 12 and Tables 2 and 3. The runtime grew *quadratically* in $t$ as expected. From Fig. 12, Est-Err is much larger and thus a *very conservative* upper bound on NRMSE. Also, Est-Err decreases as $t$ increases; we suggest using a small constant around 10-15 for $t$ for both speed and accuracy. In all the out-of-core experiments we fixed $t$ to 12; as will be seen in Figs. 15 and 16, our quality results were great and *close to optimal*.

#### 4.3.3. Estimated Total Information Difference

In our approximate method the dynamic programming results $D^{(k)}(T)$ are computed based on the approximation $\hat{c}(i,j)$; the resulting $D^{(k)}(T)$ is also an approximation, representing the estimated total InfoD. For each solution of $k$ selected time steps we can also compute its actual total InfoD. As seen in Fig. 13 (ignoring DTW for now), the estimated total InfoD is always an *upper bound* of the actual total InfoD, meaning that the estimation is always *conservative*. We also see that these two curves are actually *very close* to the curve of the globally optimal accurate method.

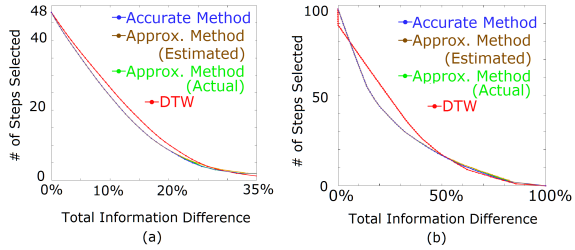


**Figure 13:** *The curves of total **InfoD** of our accurate method, approximate method (estimated total InfoD and actual total InfoD; $t = 8, 10$ for (a), (b)), and DTW: (a) Isabel-TC; (b) Vortex.*

### 4.4. Comparing with Other Methods

**Two Base-Line Methods.** We will compare against two base-line methods: uniform sampling in time steps, which is straightforward, and a greedy algorithm. For the latter, we implemented the following method *Greedy* similar to the greedy polygonal curve approximation [VW93]: start with all time steps selected; in each iteration, remove the time step that has the lowest error of reconstructing (by linear interpolation) from the two neighboring time steps that remain. We first compared our accurate method with these two base-line methods (all under VI) on our synthetic datasets; the results are given in Appendix II (in Supplementary Materials). In summary, while these two methods are simple, the qualities of their results are unstable and can be very bad, with no guarantee. On the other hand, our accurate method always provides a guarantee on the optimality. In the following, we will compare our approximate approach with these two base-line methods on real datasets, under both VI and RMSE.

**In-Core Data** The DTW method [TLS12] is most closely related to our approach. However, their cost function and the reconstruction method are different (see Sec. 2). Also, it is *not* restricted to always selecting the first and last time steps as our methods. We have implemented DTW with InfoD (under VI) as the cost function and compared with our methods on smaller datasets; see Fig. 13. Note that in DTW if a skipped time step $r$ is mapped to and "reconstructed" as some closest selected time step $i$, then its reconstruction is $X'_r = X_i$ (we call this **DTW's constant interpolation**) and we compute the error of *this reconstruction* against its original data $X_r$, i.e., $Diff_r = VI(X_r; X'_r)$ where $X'_r = X_i$ (rather than $X'_r = \text{Interpolate}(X_i, X_j, r)$ as in our methods). This reflects the *same measure*, i.e., the *total reconstruction error* that DTW optimizes. (We also optimize the *total reconstruction error* with linear

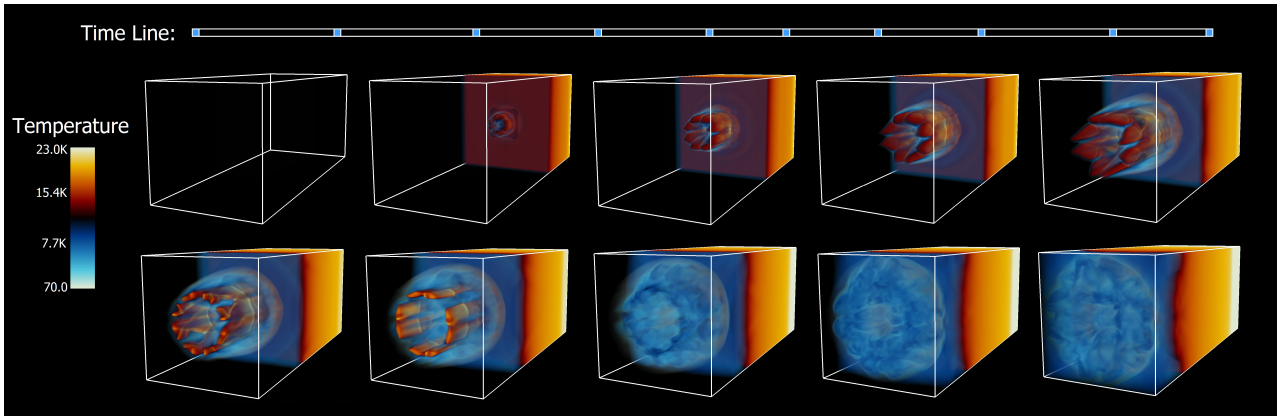


**Figure 11:** *Selecting 10 time steps from Radiation using our approximate method under VI ($t = 12$). The selected time steps {1, 29, 58, 80, 102, 117, 135, 156, 180, 200} are shown in row-first order.*

interpolation being our reconstruction.) As seen in Fig. 13 our accurate and approximate methods are most of the time better in total reconstruction error than DTW, meaning that linear interpolation is typically a better reconstruction method. The running time of DTW was slightly worse than our approximate method (450s vs. 430s for Isabel-TC, and 153s vs. 138s for Vortex); both methods are much faster than our accurate method (4908s and 3713s respectively). The same comparisons but under RMSE are shown in Fig. 26 in Appendix III. The results are similar.

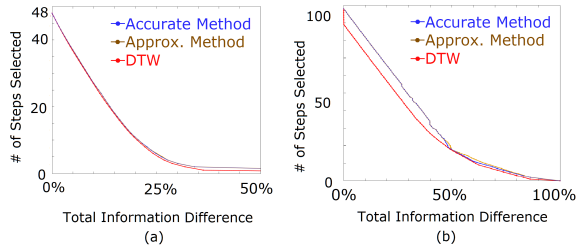


**Figure 14:** *The curves of total InfoD of our accurate method, approximate method ($t = 8, 10$ for (a), (b)) and DTW for (a) Isabel-TC and (b) Vortex, all under **DTW's metric**, which* favors DTW. *For our approximate method they are actual total InfoD. For DTW they are the same as those in Fig. 13.*

There is an issue if we treat the above total reconstruction error as the quality measure in comparing our methods with DTW, since our methods are under the metric using *linear* interpolation while DTW is under a metric using DTW's *constant* interpolation (call it **DTW's metric**). Therefore, if DTW selects the *same* set of time steps as ours, the resulting errors can be *different* due to different interpolations! This is undesirable, and we should use the **same metric** to make the comparisons valid. Ideally, that metric should also be *independent* of the choice of interpolation. However, if we had known such a metric, we would have made our methods optimize under that metric, and one could still argue that comparing with DTW under that metric is unfair to DTW since it does not optimize under that same metric. To address this issue, we compared under the **same, DTW's metric**. Doing so *favors DTW* and is *unfair to us*, but at least the comparisons are valid. The results are shown in Fig. 14 for total InfoD and in Fig. 27 (in Appendix III) for total RMSE. As expected, we are worse than DTW (which is now *optimal*), but we are *quite close*, showing that our qualities are still *close to optimal* even under this metric.

In summary, we advocate our approximate method: it is faster than DTW with similar qualities. Its runtime is dominated by $O(t^2TN)$ (Sec. 4.2), which is *linear* in $T$ for a fixed $t$, while DTW's runtime $O(T^2N + T^3)$ [TLS12] is dominated by $O(T^2N)$ (the DP time $O(T^3)$ is negligible), which is *quadratic* in $T$. The speed gap will become larger for larger data; when the data becomes out-of-core, we get significant run-time advantages (see next).

**Out-Of-Core Data** DTW does not consider I/O issues, and cannot handle out-of-core data. With a direct adaptation to the out-of-core setting, every pair of the time steps are needed to compute the costs, using $O(T^2N/B)$ I/O operations. We have implemented and run this method on TeraShake and Radiation, with InfoD and RMSE as the cost function for the "total reconstruction errors", and compared with our approximate method ($t = 12$) under the same two cost functions (recall: our I/O cost is $2TN/B = O(TN/B)$). The curves and run-time results are shown in Fig. 15. As seen, the minimized "total reconstruction errors" under InfoD/RMSE of the two methods were similar (ours was slightly better), but we always had a huge run-time advantage (e.g., 2.1 hours vs. 21.6 hours for Radiation under InfoD in Fig. 15(a)). We also compared with uniform sampling and *Greedy* in Fig. 15 (where their reconstructions are linear interpolation). We see that their qualities can be quite bad (see (b),(d)). Also, with a direct adaptation to the out-of-core setting, *Greedy* needs $O(T^2N/B)$ I/O cost to obtain the solutions (with the curve a by-product), which was much slower than our method. Uniform sampling needs *no time* for the solutions. For the curve, it needs to go over the whole dataset once, i.e., $O(TN/B)$ disk reads, for each value of $k$, for a total of $O(T^2N/B)$ I/O cost (it was faster than *Greedy* and DTW since there are gaps in the possible $k$ values: $T, T/2, T/3, \cdots$ rather than $T, T-1, T-2, \cdots$).

As mentioned before, when comparing our method with DTW,

the total reconstruction error in Fig. 15 is not a desirable quality measure since their metrics are different due to different interpolations. To fix, again we compared them under the **same, DTW's metric**, which *favors DTW* and is *unfair to us*; the results are shown in Fig. 16 for total InfoD and in Fig. 28 (in Appendix III) for total RMSE. Again, we are slightly worse than DTW but *very close*, showing that our approximate method still achieves *similar, close to optimal* qualities even under this metric.

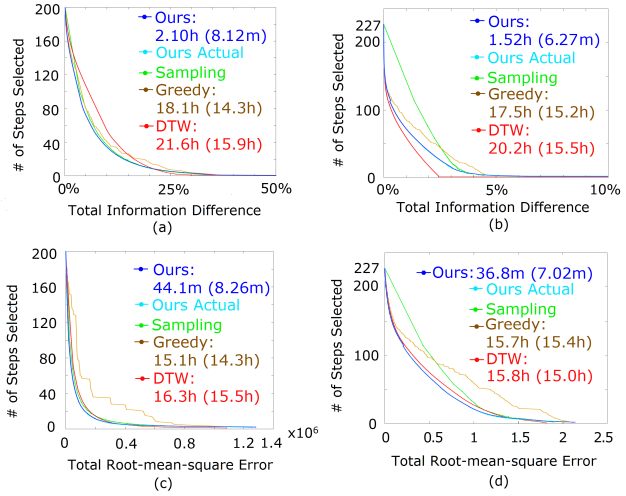


**Figure 15:** *The curves and running times ("total time (I/O time)") of our approximate method ($t = 12$), DTW, uniform sampling and* Greedy *for Radiation (left column, (a)(c)) and TeraShake (right column, (b)(d)). For our method, the InfoD/RMSE curves of "Ours" were* estimated upper bounds*, which were very close to the* actual *curves ("Ours Actual"). All methods used the times shown for the solutions (with the curve a by-product) except for* ***uniform sampling*** *(no time for solutions; times ("total (I/O)") for the curve: 2.89h (1.98h), 2.55h (2.07h), 2.11h (1.97h), and 2.17h (2.09h) for (a)-(d) respectively).*

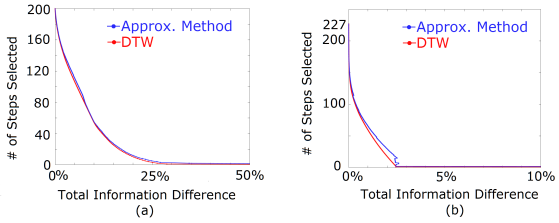


**Figure 16:** *The curves of* actual *total InfoD of our approximate method ($t = 12$) and DTW for (a) Radiation and (b) TeraShake, all under* ***DTW's metric****, which* favors DTW*. For DTW they are the same as those in Fig. 15(a)(b).*

Related to Fig. 15(b) (dataset TeraShake), the rendering results of selecting 10 time steps for TeraShake under InfoD using our method, DTW, uniform sampling and *Greedy* are respectively shown in Figs. 7, 8, 9 and 10. We can see that DTW and our method selected similar time steps and their rendering results are visually very close. Also, we see in Fig. 9 that uniform sampling selected more time steps after 140, which contain little information and are not representative (recall the dataset description in Sec. 4.2 under "Dataset: TeraShake"). The selection of *Greedy* (Fig. 10) was highly concentrated in between time steps 50 and 80, and ignored others of slower events. The resulting selection does not show the complete process as our method does. In summary, uniform sampling and *Greedy* had much worse selection qualities than ours (Figs. 9, 10 and 15(b),(d)), showing that these base-line methods are clearly not adequate.

Finally, we ran our method ($t = 12$) under InfoD on the three largest datasets, RadiationX with $X = 2, 4, 8$ (same dimensions as Radiation but the numbers of time steps are 2, 4 and 8 times). The results are shown in Table 4 (note the different time units: **h, m, s**). Same as observed in Sec. 4.2, the in-core computing time was much higher than the I/O time. For the former, the DP time is $O(T^3)$, which is *negligible* compared to the $O(t^2TN)$ term. The latter is linear in $TN$ (the dataset size) when $t$ is fixed ($t = 12$); the experiments confirmed that **the total time and I/O time were both roughly linear in $TN$ (also in $T$ here for the same $N$)**. The memory footprint is $O(tN)$ and thus the same for all entries in Table 4. More importantly, since the running time of DTW is dominated by I/O, whose cost is quadratic in $T$ ($O(T^2N/B)$), the running time of DTW on Radiation8 would be at least $8^2 \cdot 16 = 1024$ hours just for I/O. Comparing with 19.5 hours in our method, our run-time advantage is clearly far more significant.

## 5. Conclusions, Limitations, Future Work

We have presented a novel approach to select key time steps for time-varying dataset. Our algorithms can work under any metric that measures the selection quality; we define such a metric based on information theory, which could extract unknown features and is effective in general. With dynamic programming, we compute the most representative sets of time steps of all sizes. We present the results as a storyboard to guide the user to explore the data. We extend our method to a novel out-of-core approximate algorithm with optimal I/O cost and much faster in-core computation. Specifically, the I/O cost is optimal $O(T(N/B))$ and the in-core running time is $O(t^2TN + T^3)$. As seen, typically we can achieve a good accuracy with a small $t$ (e.g. 12). This makes the in-core computing time $O(TN + T^3) = O(TN)$ (i.e., linear in the dataset size $TN$) when $N$ is asymptotically larger than $T^2$, which is mostly true in practice.

**Limitations:** For datasets with an extremely large number of time steps, i.e., $T^2$ asymptotically larger than $N$, the in-core computing

**Table 4:** *Run-Time Results of our approximate method ($t = 12$) under InfoD for the largest datasets. $T$: # time steps in the datasets; Total: total runtime in* ***hours (h)****; I/O: I/O time in* ***minutes (m)****; DP: time for dynamic programming in* ***seconds (s)****.* ***All entries had the same memory footprint 1.91GB.***

| Dataset | Size | $T$ | Total (**h**) | I/O (**m**) | DP (**s**) |
|---|---|---|---|---|---|
| *Radiation* | 27.4GB | 200 | 2.1 | 8.1 | 0.026 |
| *Radiation2* | 54.8GB | 400 | 4.5 | 17.8 | 0.19 |
| *Radiation4* | 109.6GB | 800 | 9.4 | 36.6 | 1.35 |
| *Radiation8* | 219.2GB | 1600 | 19.5 | 73.2 | 13.6 |

cost of our out-of-core algorithm becomes $O(TN + T^3) = O(T^3)$, which would be too expensive to be practical (even though the I/O is still optimal). Such scenario would be more likely in video processing (much larger $T$ (number of frames) and much smaller $N$ (number of pixels in a 2D image)). However, for time-varying volume data such scenario is much less likely to occur in practice.

**Extensibility and Future Work:** For possible extensions, our methods are readily applicable to other mesh types (e.g. unstructured grids) as long as the volume mesh stays the same for all time steps, though here we only focus on regular grids. Also, in principle our methods could be extended to work for multi-field or vector-field datasets, but it would be necessary to develop novel and meaningful notions of representative time steps (and their measures) depending on the underlying applications, which could be challenging. We currently use linear interpolation to estimate the missing time steps. Another direction is to use a better interpolation, e.g., utilizing spatial information or domain knowledge. Our methods are designed for data exploration/visualization after the simulation is done. It would be a challenge to see if some of our ideas could be used to re-design the techniques in the in-situ setting.

**Acknowledgments:** This work was supported in part by DOE Grant DE-SC0004874, program manager Lucy Nowell.

## References

[AFM06] AKIBA H., FOUT N., MA K.-L.: Simultaneous classification of time-varying volume data based on the time histogram. In *Proc. IEEE/Eurographics Symp. EuroVis '06* (2006), pp. 171–178.

[AM07] AKIBA H., MA K.-L.: A tri-space visualization interface for analyzing time-varying multivariate volume data. In *Proc. IEEE/Eurographics Symp. EuroVis '07* (2007), pp. 115–122.

[BS05] BORDOLOI U., SHEN H.-W.: View selection for volume rendering. In *Proc. IEEE Visualization* (2005), pp. 487–494.

[CFV*16] CHEN M., FEIXAS M., VIOLA I., BARDERA A., SHEN H.-W., SBERT M.: *Information Theory Tools for Visualization*. AK Peters/CRC Press, 2016.

[CM10] CORREA C. D., MA K.-L.: Dynamic video narratives. *ACM Transactions on Graphics (Proc. SIGGRAPH '10) 29*, 4 (2010), 88:1–88:9.

[CT91] COVER T. M., THOMAS J. A.: *Elements of Information Theory*. Wiley-Interscience, New York, NY, USA, 1991.

[FE17] FREY S., ERTL T.: Flow-based temporal selection for interactive volume visualization. *Computer Graphics Forum 36*, 8 (2017), 153–165.

[Gum02] GUMHOLD S.: Maximum entropy light source placement. In *Proc. IEEE Visualization* (2002), pp. 275–282.

[GW11] GU Y., WANG C.: Transgraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics (Vis'11) 17*, 12 (2011), 2015–2024.

[HXL*11] HU W., XIE N., LI L., ZENG X., MAYBANK S.: A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews 41*, 6 (2011), 797–819.

[Kit03] KITWARE, INC.: *The Visualization Toolkit User's Guide*, 2003.

[KSAG08] KRASKOV A., STOGBAUER H., ANDRZEJAK R. G., GRASSBERGER P.: Hierarchical clustering based on mutual information, 2008. http://arxiv.org/pdf/q-bio/0311039v2.pdf.

[LHV06] LEE C.-H., HAO X., VARSHNEY A.: Geometry-dependent lighting. *IEEE Transactions on Visualization and Computer Graphics 12*, 2 (2006), 197–207.

[LK02] LIU T., KENDER J.: Optimization algorithms for the selection of key frame sequences of variable length. In *Proc. European Conference on Computer Vision-Part IV (ECCV'02)* (2002), pp. 403–417.

[LS08] LU A., SHEN H.-W.: Interactive storyboard for overall time-varying data visualization. In *Proc. IEEE Symp. Pacific Visualization* (2008), pp. 143–150.

[LS09a] LEE T.-Y., SHEN H.-W.: Visualization and exploration of temporal trend relationships in multivariate time-varying data. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1359–1366.

[LS09b] LEE T.-Y., SHEN H.-W.: Visualizing time-varying features with TAC-based distance fields. In *Proc. IEEE Pacific Visualization Symposium* (2009), pp. 1–8.

[MAB*97] MARKS J., ANDALMAN B., BEARDSLEY P. A., FREEMAN W. T., GIBSON S. F., HODGINS J. K., KANG T., MIRTICH B., PFISTER H., RUML W., RYALL K., SEIMS J., SHIEBER S. M.: Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proc. ACM SIGGRAPH* (1997), pp. 389–400.

[MLF*16] MYERS K., LAWRENCE E., FUGATE M., BOWEN C. M., TICKNOR L., WOODRING J., WENDELBERGER J., AHRENS J.: Partitioning a large simulation as it runs. *Technometrics 58*, 3 (2016).

[MSSS98] MA K.-L., SMITH D., SHIH M.-Y., SHEN H.-W.: Efficient encoding and rendering of time-varying volume data, 1998. NASA/CR-1998-208424.

[ODM*04] OLSEN K., DAY S., MINSTER J. B., MOORE R., CUI Y.: IEEE visualization 2006 design contest, 2004. http://sciviscontest.ieeevis.org/2006/data.html.

[SB06] SOHN B.-S., BAJAJ C.: Time-varying contour topology. *IEEE Transactions on Visualization and Computer Graphics 12*, 1 (2006), 14–25.

[Sha48] SHANNON C.: A mathematical theory of communication. *The Bell System Technical Journal 27*, 3 (1948), 379–423.

[TFO09] TAKAHASHI S., FUJISHIRO I., OKADA M.: Applying manifold learning to plotting approximate contour trees. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1185–1192.

[TLS12] TONG X., LEE T.-Y., SHEN H.-W.: Salient time steps selection from large scale time-varying data sets with dynamic time warping. In *Proc. IEEE Symp. Large Data Analysis and Visualization (LDAV '12)* (2012), pp. 49–56.

[VW93] VISVALINGAM M., WHYATT J.: Line generalization by repeated elimination of points. *Cartographic J. 30*, 1 (1993), 46–51.

[WBK04] WANG W., BRUYERE C., KUO B.: IEEE visualization 2004 contest, 2004. http://vis.computer.org/vis2004contest/data.html.

[WLS13] WEI T.-H., LEE T.-Y., SHEN H.-W.: Evaluating isosurfaces with level-set-based information maps. *Computer Graphics Forum (Special Issue for EuroVis '13) 32*, 3 (2013), 1–10.

[WN08] WHALEN D., NORMAN M. L.: Competition data set and description, IEEE visualization 2008 design contest, 2008. vis.computer.org/VisWeek2008/vis/contests.html.

[WS09] WOODRING J., SHEN H.-W.: Multiscale time activity data exploration via temporal clustering visualization spreadsheet. *IEEE Transactions on Visualization and Computer Graphics 15*, 1 (2009), 123–137.

[WS11] WANG C., SHEN H.-W.: Information theory in scientific visualization. *Entropy 13*, 1 (2011), 254–273.

[WYM08] WANG C., YU H., MA K.-L.: Importance-driven time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1547–1554.