Parallel Hierarchical Visualization of Large Time-Varying 3D Vector Fields

Hongfeng Yu* Chaoli Wang* Kwan-Liu Ma*

Department of Computer Science
University of California at Davis

ABSTRACT

We present the design of a scalable parallel pathline construction method for visualizing large time-varying 3D vector fields. A 4D (i.e., time and the 3D spatial domain) representation of the vector field is introduced to make a time-accurate depiction of the flow field. This representation also allows us to obtain pathlines through streamline tracing in the 4D space. Furthermore, a hierarchical representation of the 4D vector field, constructed by clustering the 4D field, makes possible interactive visualization of the flow field at different levels of abstraction. Based on this hierarchical representation, a data partitioning scheme is designed to achieve high parallel efficiency. We demonstrate the performance of parallel pathline visualization using data sets obtained from terascale flow simulations. This new capability will enable scientists to study their time-varying vector fields at the resolution and interactivity previously unavailable to them.

1. INTRODUCTION

Massively parallel supercomputers enable scientists to simulate complex phenomena in unprecedented detail. When scientists attempt to analyze and understand the data generated by large-scale simulations, the sheer size of the data is a major challenge. To address this challenge, many advances have been made for large-scale data visualization. However, most of the techniques were developed for the visualization of scalar field data, regardless of the fact that vector fields in the same data sets are equally critical to the understanding of the modeled phenomena. While vector field visualization has also been an active area of research [8, 11, 29, 19, 24], large-scale time-varying 3D vector fields have rarely been studied for several reasons. First, most of the effective 2D vector field visualization methods incur visual clutter when directly applied to depicting 3D vector data. Second, a large vector data set contains three times the data as its corresponding scalar field. A single PC generally does not have the memory capacity and processing power to enable interactive visualization of the data. Next, additional attention to temporal coherence is required for visualizing time-varying vector data. Finally, it is challenging to simultaneously visualize both scalar and vector fields due to the added complexity of rendering calculations and combined computing requirements. As a result, previous works in vector field visualization primarily focused on 2D, steady flow field, the associated seed/glyph placement problem, or the topological aspect of the vector fields. This paper presents our goal of visualizing large time-varying 3D vector fields using a parallel computer with scalable performance. The objective of our work is to provide scientists the capability to look at their data at the desired resolution and precision when a parallel computer is available to them.

Particle tracing is a commonly used method for portraying the structure and direction of a flow vector field. When an appropriate set of seed points are used, we can construct paths and surfaces from the traced particles to effectively characterize the flow field. Visualizing a large time-varying vector field on a parallel computer using particle tracing, however, presents some unique challenges. Even though the tracing of each individual particle is independent of other particles, a particle may drift to anywhere in the spatial domain over time, demanding interprocessor communication. Furthermore, as particles move around, the number of particles each processor must handle varies, leading to uneven workloads.

We present a scalable parallel pathline construction method for visualizing time-varying 3D vector fields. We advocate a high-dimensional approach by treating time as the fourth dimension, rather than consider space and time as separate entities. In this way, a 4D volume is used to represent a time-varying 3D vector field. This unified representation enables us to make a time-accurate depiction of the flow field. More importantly, it allows us to construct pathlines by simply tracing streamlines in the 4D space. To support adaptive visualization of the data, we cluster the 4D space in an hierarchical manner. The resulting hierarchy can be used to allow visualization of the data at different levels of abstraction as well as enable interactivity. This hierarchy also facilitates data partitioning for efficient parallel pathline construction. We demonstrate the performance of parallel pathline visualization of complex flow fields using both a small graphics cluster and a large general-purpose cluster. This new capability enables scientists to see their vector field

*Department of Computer Science, University of California at Davis, One Shields Avenue, Davis, CA, 95616. {yuho,wangcha,ma}@cs.ucdavis.edu

SC07 November 10-16, 2007, Reno, Nevada, USA

data in unprecedented detail and with higher interactivity.

The contributions of our works are the following. First, the 4D representation for time-varying vector fields facilitates time-accurate pathline tracing, and tracing in this 4D space is conceptually more intuitive and practically easier to implement than traditional methods. Second, we introduce a supplemental space partitioning grid to make it possible to use a single PC to perform hierarchical clustering of a large 3D vector field. Finally, using the clustering results, we are able to derive an even partitioning of the vector field for highly scalable parallel pathline tracing.

2. BACKGROUND AND RELATED WORK

In the modeling of many scientific and engineering problems, vector fields are used to describe moving fluids or changing forces, where a vector (i.e., a direction with magnitude) is assigned to each point in the spacetime domain. Effective visualization of time-varying 3D vector fields is critical for the understanding of complex phenomena and dynamic processes under investigation. Streamline generation with seed point placement is a popular method for visualizing vector fields. A *streamline* is a curve tangent to the field at all points. In practice, a streamline is often represented as a polyline (a series of points) iteratively elongated by bidirectional (i.e., forward and backward) numerical integration. The integration starts from a *seed point*, and continues until the current polyline comes close to another streamline, hits the domain boundary, reaches a critical point, or generates a closed path. A valid placement of streamlines consists of saturating the domain with a set of tangential streamlines in accordance with a specified density, determined by the separating distance between the streamlines.

Vector fields remain an active area of visualization research. Existing techniques can be classified into glyph and field-line based methods [29, 16], dense texture methods [19, 24], clustering-based methods [21, 7, 6], and topology-based methods [8, 18]. Glyph or field-line visualization is particularly effective for the visualization of isolated regions in the vector field. Dense texture methods can give more realistic depictions of a vector field but the fundamental occlusion problem has not been solved. Clustering-based methods enable us to convey the structure of the vector field at different abstraction levels. However, the existing algorithms have been restricted to the visualization of steady, not time-varying, vector fields. Topology extraction and visualization for 3D vector fields are still ongoing research.

Taking into account the temporal dimension of the vector field makes the visualization even more challenging. Early investigations mostly centered around dense texture advection [15, 19, 9, 24]. In previous work [26, 22], a time-varying vector field is represented as a steady field in the spacetime domain that separates the spatial from the temporal dimension. It is noted that some conventional operations in the spatial domain, such as the distance calculation of two points, cannot be performed in the spacetime domain. However, these operations are essential for visualization calculations, such as streamline placement [10, 13] and clustering [21, 6]. Unlike the previous research, we achieve the spatial and temporal coherence in a high-dimensional space. More specifically, we construct a 4D steady field to represent a time-varying 3D vector field, and each vector component in the new field has the same physical scale. As a result, this representation allows us to directly apply the techniques previously developed for visualizing (and simplifying) steady vector fields to make time-accurate, coherent visualization of time-varying vector fields. It is conceptually more intuitive, and practically easier, than conventional representations.

Parallel computing has been widely used in visualizing large-scale steady or time-varying scalar field data. Typical volume visualization methods, such as raycasting, volume rendering, and isosurface rendering, have benefited from utilizing a cluster of PCs for parallel rendering for performance speedup [27, 1, 25, 28]. Nevertheless, less work has been done on parallel methods for vector field visualization. Early examples include the use of multiprocessor workstations, such as Cray C90, Convex C3240, and SGI systems, to parallelize particle tracing [11, 12]. There are also a few research efforts focusing on parallel line integral convolution (LIC) [4, 30]. More recently, Muraki et al. [17] presented a scalable PC cluster system for enabling simultaneous volume computation and visualization, which includes 3D time-varying LIC volumes for animation. Ellsworth et al. [5] described a method for interactive visualization of particles from terabytes of computational fluid dynamics (CFD) data using a PC cluster. Bachthaler et al. [2] proposed a parallel scheme to visualize flow fields on curved surfaces, where a hybrid sort-first and sort-last algorithm is used to achieve a scalable rendering in terms of both visualization speed and data size. In this paper, we present a clustering-based data partition scheme and a parallel representative streamline generation algorithm for large-scale time-varying 3D vector field visualization.

Out-of-core techniques are commonly used in large-scale data applications where data sets are larger than main memory. These algorithms focus on achieving high I/O performance to access data stored on disk. For vector field visualization, Ueng et al. [23] presented an approach to compute streamlines of large unstructured grids. They used an octree to partition and restructure the raw data for fast data fetching during streamline construction and achieving small main memory footprints. Bruckschen et al. [3] described a technique for real-time particle traces of large time-varying data sets. In the preprocessing stage, the particle traces are computed and stored into disk files for efficient data retrieval. In the rendering stage, the precomputed traces are read interactively. More out-of-core techniques for scientific visualization and computer graphics applications are reviewed in [20]. While also facilitating I/O operations, our data partition and distribution scheme is mainly designed for achieving optimal parallel scalability.

3. ALGORITHM OVERVIEW

Figure 1 gives an example of a time-varying 2D vector field that illustrates our basic approach. Given a n-d large time-varying vector field, we treat it as a unified $(n+1)$-d steady vector field, where the time is considered as the $(n+1)$st dimension. Then, we create an adaptive grid from the $(n+1)$-d data representation. The grid resolution adapts to the feature size of the local flow. That is, those regions having more uniform patterns will use coarser grids, while regions with more distinct patterns will use finer grids. This approximation outputs a manageable scale of adaptive grid as the input to the hierarchical clustering algorithm. The clustering merges neighboring grid cells of similar patterns and thus creates a binary cluster tree.

At runtime, the user specifies the number of seed points

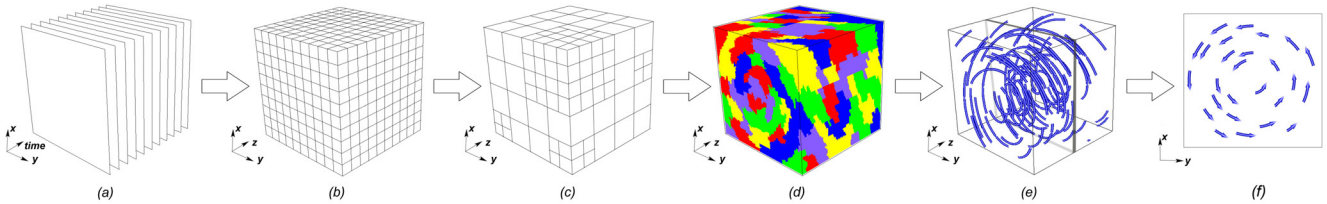


Figure 1: Visualizing a time-varying 2D vector field using a hierarchical 3D representation of the spacetime field. A sequence of 2D vector fields (a) is represented as a 3D steady field (b), from which we build an adaptive grid (c). The cells in the adaptive grid are then clustered into a hierarchical data representation (d). Streamlines are derived for a particular level of abstraction (e) from the original 3D field. Finally, projecting the streamlines back to the 2D space gives a coherent pathline animation (f).

for particle tracing. We traverse the binary cluster tree and obtain the seeds from the clustering of the adaptive grid representation. The streamlines are then traced in parallel in the original $(n+1)$-d vector data to generate numerically-accurate results. We carefully devise a data partition and distribution scheme based on the hierarchical clustering results to ensure the workload balance among processors. Our solution can effectively avoid heavy communication overhead brought by typical parallel particle tracing algorithms.

In the rest of the paper, we describe key stages of our algorithm in detail. First, we address the issue of streamline generation for large steady 3D vector fields. Then, we discuss how to extend the 3D streamline generation algorithm to 4D spacetime vector fields. After that, we present our data partition and distribution scheme for parallel streamline generation and pathlet rendering. Implementation details and test results will follow.

4. STREAMLINE GENERATION FOR LARGE STEADY VECTOR FIELDS

Most previous streamline placement methods explicitly rely on streamline calculation, which are highly sensitive to the data precision and require the complete data set as input in the calculation. This makes those methods computationally intensive when the size of the input vector field is large. Another category of methods, vector field clustering [21, 6], is successful in finding representative streamlines based on the clustering analysis. These methods share some characteristics with traditional clustering methods in data mining. The advantages of these methods are: first, it does not require the in-depth knowledge about the flow field. Second, a vector field is decomposed in a hierarchical fashion, so that the visualization can contain both the global structures and local details at different resolutions. Third, although not discussed in the original papers, these methods have the potential scalability for large vector data. Moreover, they are able to yield approximate results with controllable error bounds through some general strategies, such as sampling and partitioning the vector field. In this section, we first present a typical vector field clustering technique - the simplified representation method [21] for steady vector fields. Then we demonstrate how to extend this method to handle large steady vector fields.

4.1 Vector Field Simplification

The vector field simplification method introduced by Telea and van Wijk [21] is a typical agglomerative hierarchical clustering method. First of all, the whole domain of an input vector field is decomposed into clusters, each of which is a connected subdomain. Before clustering, each cell (i.e., voxel) in the vector field is considered as a separate cluster. Then, a bottom-up clustering algorithm is performed iteratively such that in each iteration, the two most resembling neighboring clusters are merged into a larger cluster. This merging process repeats until we arrive at a single cluster that covers the entire domain. A binary tree is generated during this clustering process, indicating which two clusters are merged at each iteration. Each tree node has a *level* attribute, l, indicating when the cluster is created. By default, all leaves have the level of 0 and the root has the level of $N-1$, where N is the number of cells in the initial data set.

The key step of this clustering algorithm is to evaluate the similarity of two clusters and merge them. Each cluster has a representative vector. Initially, for each cell, the representative vector is its corresponding cell's vector data, with its origin at the center of the cell. The similarity evaluation of two clusters compares directions, magnitudes, and positions of their representative vectors. When merging two clusters into a new one, the representative vector for the new cluster is the area-weighted (in 2D) or volume-weighted (in 3D) average of the merged cluster. Thus, the position of the representative vector is always the gravity center of a cluster. Furthermore, the clustering can be orthogonal to (or along) the underlying flow field by changing related parameters.

At the visualization stage, the user first selects the clusters by choosing the level in the binary cluster tree. Then, the selected clusters are visualized by computing representative streamlines from the cluster's center up and downstream, or by rendering their representative vectors directly (e.g., in a glyph style). In this way, the vector field is simplified and represented by a number of polylines in the final visualization. The number of polylines can be controlled by selecting different levels in the cluster tree: the leaves are the original data cells which provide the finest information. The tree nodes in higher levels represent coarser information and contain representation errors. Figure 2 illustrates the whole procedure for this clustering-based visualization.

4.2 Adaptive Grid Construction

The hierarchical clustering calculation is time consuming. When the size of the input 3D vector field is large, such as in the gigabytes range or larger, since the output binary cluster tree has a size comparable to that of the input data, a single PC with limited memory space would not be usable for visualizing large time-varying vector data, unless an out-of-core method is used. To circumvent such demanding space and

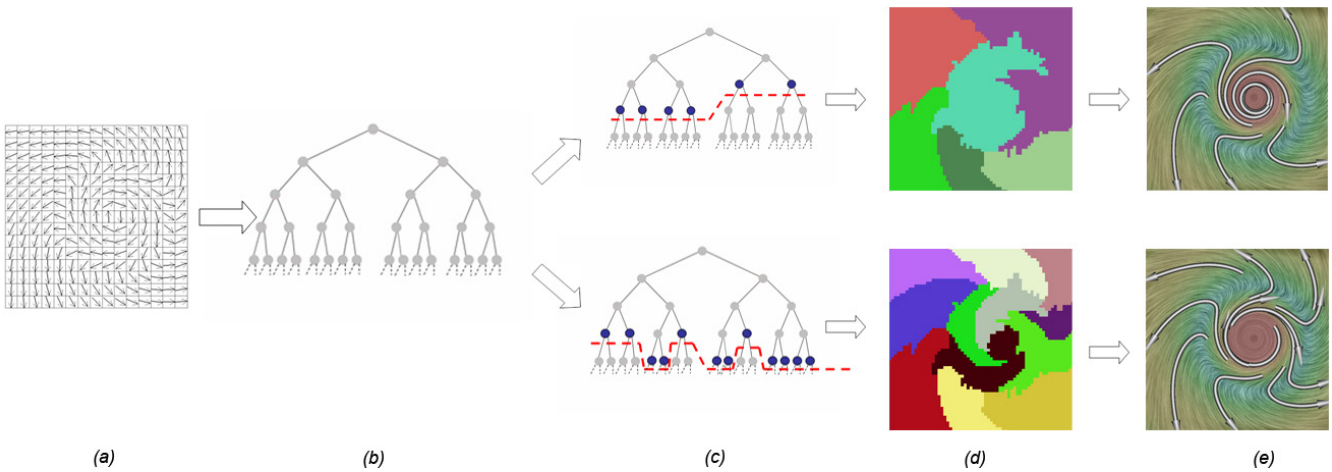


Figure 2: Given a vector field (a), a binary cluster tree (b) is generated via clustering. Each tree node represents a cluster at a different level. With different levels specified by the user at runtime, a list of tree nodes (blue dots) are selected (c) and the corresponding clusters are displayed in (d). Consequently, (e) shows the representative streamlines generated from the clusters, blended with the LIC textures as background.

compute requirements, we partition the vector field to derive an adaptive grid that is much coarser than the original grid and use it instead for the hierarchical clustering. This works well because the number of pathlines needed to faithfully depict the flow field is generally small. A coarser-level partitioning of the vector field is thus sufficient in most of the cases. The user chooses clusters from the cluster tree for pathline visualization, and each chosen cluster is essentially a simplified or lower resolution version of the underlying vector field.

The adaptive grid is constructed as follows: first, we subdivide the original vector data into data blocks with an equal size of $m \times m \times m$, where m depends on the resolution of original data and is usually 8 or 16 in our experiments. Then, we attempt to evenly subdivide each data block into eight octants. We evaluate the dissimilarity of two neighboring octants as in [21]. The representative vector in each octant is the average of all vectors inside. The dissimilarity measure takes into account the directions and positions of the representative vectors. If the maximum dissimilarity among the eight octants is less than a threshold e, we stop the subdivision. Otherwise, we perform the subdivision recursively until the maximum dissimilarity is below the given threshold, or the octant reaches the minimal grid size of $n \times n \times n$, where n is 2 or 4 in our experiments.

In this way, we generate an adaptive grid from the original vector data, in the sense that the grid resolution adapts to the feature size of the local flow. In the adaptive grid, each data block constitutes a cell, which is the input to the clustering algorithm. This adaptive grid of manageable scale enables us to perform hierarchical clustering of a large 3D vector field on a single PC. Note that the adaptive grid is only used for the clustering purpose. To ensure correctness, subsequent streamlines are derived for a particular level of abstraction from the original 3D vector data, not from the adaptive grid. Figure 3 gives a 2D example of the hierarchical clustering and streamline generation using the original voxel-level grid and an adaptive grid. As we can see, streamlines generated from the adaptive grid capture the features of the underlying flow field quite well, and agree with the streamlines generated from the original grid.

5. PATHLET GENERATION FOR LARGE TIME-VARYING VECTOR FIELDS

For a time-varying vector field, applying the clustering method directly to each time step gives us a sequence of representative streamline images. Animating these images, however, does not give us a temporally accurate visualization. To achieve accurate results for time-varying vector fields, particle tracing needs to be performed in the space-time domain instead of separately with each time step. In this section, we first discuss the correlation between spatial and temporal coherence, and then present a novel way to represent a time-varying 3D vector field as a steady vector field in the 4D space. This high-dimensional representation converts the particle tracing problem, originally stated in the spacetime domain, to a problem stated in a unified 4D domain. Consequently, our clustering method, based on the adaptive grid representation, can be applied directly to generate representative pathlets from large time-varying 3D vector fields.

5.1 Spatial and Temporal Coherence

Spatial and temporal coherence is critical for achieving the accurate visualization of an unsteady flow field. These two types of coherence are conventionally expressed in streamline and pathline visualization, respectively. Given a time-dependent n-d flow field $\mathbf{v}(\mathbf{x}, t) \in \mathbb{R}^n$ where t is time and $\mathbf{x}$ is a point in $\mathbb{R}^n$, a pathline is the path of a particle motion in this field. More precisely, it is the solution of the following differential equation

$$\frac{d(\mathbf{p}(t))}{dt} = \mathbf{v}(\mathbf{p}(t), t) \tag{1}$$

for a given starting position $\mathbf{p}(0)$, where $\mathbf{p}(t)$ is the position of the particle at time t. Based on the definition of pathline, advecting particles with some visual cue (e.g. color) along the pathlines can generate images with strong temporal coherence.

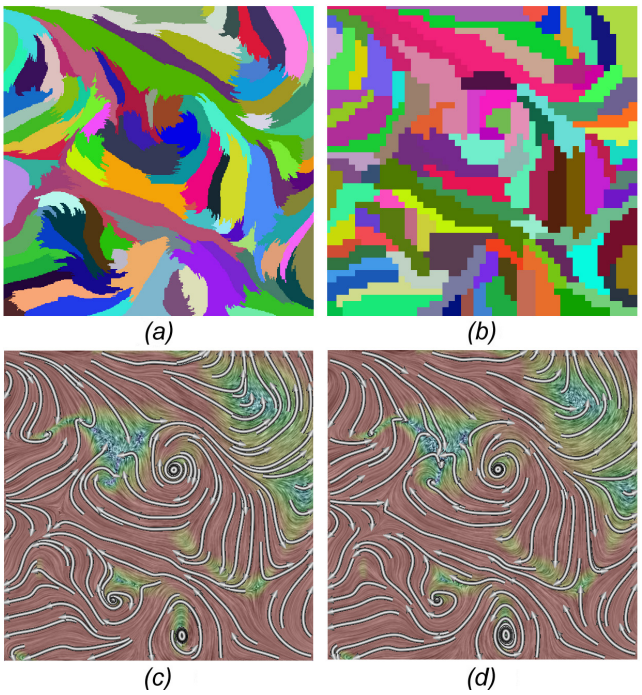


Figure 3: The comparison of clustering and streamline generation results using the original voxel-level grid and an adaptive grid. The original grid resolution is 400×400 and the clustering result is shown in (a). The initial block size for the adaptive grid is 8×8 and the clustering result is shown in (b). The streamlines generated based on the clustering results from the original grid and the adaptive grid are displayed in (c) and (d), respectively. Both (c) and (d) generate a total of 100 streamlines. The comparison shows that, overall, the streamlines generated in (d) approximate (c) quite well.

A streamline has an almost identical definition as Equation 1, except that t is the parameter along the curve, and does not represent the physical time. Streamlines illustrate the pattern of an instantaneous flow field and are only spatially coherent. There is no temporal correlation between streamlines at two consecutive time steps, because two points on the same instantaneous streamline may belong to different streamlines in the next time step. The problem of using streamlines to visualize time-varying flow field has been discussed in [19].

A pathline segment in a time interval ϵ is referred as to *pathlet*. If we plot the streamlines at a particular instance of time t and the pathlets in $[t-\epsilon, t]$ on an image, the difference between the streamlines and the pathlets depends on ϵ. When ϵ is close to zero, the pathlets are degraded to particle snapshots without any spatial correlation. When ϵ is small enough such that the flow changes slightly, the streamlines and the pathlines nearly coincide. As ϵ keeps increasing, the difference between the streamlines and the pathlets becomes more noticeable, so does the variation of the flow.

Therefore, spatial and temporal coherence can be established together by utilizing pathlets in a small time interval ϵ: the pathlets coincide with the instantaneous streamlines, thus demonstrating the spatial coherence. In addition, since each pathlet belongs to a pathline, the temporal coherence

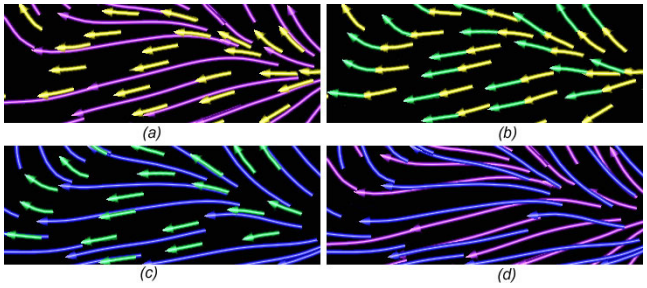


Figure 4: (a) and (c) show two consecutive time steps, where long curves are instantaneous streamlines, and short arrows are pathlets. In each time step, the pathlets are along instantaneous streamlines and are spatially coherent. (b) shows that the pathlets' movements are continuous and the temporal coherence is maintained. (d) shows that the instantaneous streamlines of two time steps are not temporally coherent.

can be achieved by animating the movement of the pathlets. Such spatial and temporal coherence is shown in Figure 4.

5.2 High-Dimensional Representation

Constructing pathlets at each time step and animating them over time can achieve both spatial and temporal coherence. The placement of the pathlets is one of the key issues that determine the quality of the consequent visualization. In the spatial domain, similar to the seeding issue in streamline generation, optimal placement of pathlets can lead to an effective visualization that provides both the global structure of the flow field and the local details in the active flow regions, while avoiding the visual clutter and occlusion. In the temporal domain, similar to the seeding issue in particle tracing, since some pathlets leave the flow field as time evolves, the new pathlets need to be injected in order to maintain a constant coverage of pathlets over time. On the other hand, some pathlets need to be removed to avoid visual clutter.

A pathline can be treated as a curve in spacetime. Such a curve is commonly referred to as the *trajectory*. Given that a pathlet is a pathline segment, the above two issues can be solved if we address the trajectory placement issue in spacetime. This is because, given a particular type of trajectory placement, the projections in both the spatial and temporal domain reflect the corresponding placement.

Intuitively, a trajectory placement algorithm can be derived from streamline placement in a high-dimensional space. However, due to different scalars and physical meanings between the space axes and the time axis, it is non-trivial to explicitly construct trajectories and treat them as the streamlines. On the other hand, a n-d curve can be projected from a curve in $(n+1)$-d space, or in $(n+1)$-d spacetime domain. Therefore, the task of finding a trajectory placement algorithm becomes easier if we can find streamlines in the high-dimensional space, whose low dimensional projections are identical to pathlets. In this paper, we call such curves *pseudo-trajectories*.

We present a high-dimensional representation of a time-varying vector field to capture pseudo-trajectories. In this study, we assume a time-varying 3D vector field is defined in the Cartesian grid (i.e., uniform rectangular grid). Then, given a n-d unsteady vector field:

$$\mathbf{v}(x_1,\ldots,x_n,t) = \begin{pmatrix} v_1(x_1,\ldots,x_n,t) \\ \vdots \\ v_n(x_1,\ldots,x_n,t) \end{pmatrix} \quad (2)$$

where t is time and $(x_1,\ldots,x_n)$ describes the $\mathbb{R}^n$ domain, we construct a $(n+1)$-d steady vector field by connecting the grid vertices of every two consecutive time steps:

$$\mathbf{v}(x_1,\ldots,x_n,x_{n+1}) = \begin{pmatrix} v_1(x_1,\ldots,x_n,x_{n+1}) \\ \vdots \\ v_n(x_1,\ldots,x_n,x_{n+1}) \\ v_{n+1} \end{pmatrix} \quad (3)$$

where t is treated as a spatial axis x_{n+1}, $(x_1,\ldots,x_{n+1})$ describes the $\mathbb{R}^{n+1}$ domain, $(v_1,\ldots,v_n)$ are the same as the original vectors, but v_{n+1} is unknown. If a particle $\mathbf{p}(p_0,\ldots,p_{n+1})$ is released in this $(n+1)$-d steady field, in n-d space it travels along the original unsteady flow field. Along the $(n+1)$st axis, its traveling is dependent on v_{n+1}:

$$p_{n+1}(t+\Delta t) = p_{n+1}(t) + \int_t^{t+\Delta t} v_{n+1}(\mathbf{p}(t))dt \quad (4)$$

According to the construction of our $(n+1)$-d Cartesian grid, along the $(n+1)$st axis the particle $\mathbf{p}$ travels across one grid cell, which corresponds to one time step. The time interval between two consecutive time steps is referred to as τ. Thus, replacing Δt with τ in Equation 4, we obtain:

$$\int_t^{t+\tau} v_{n+1}(\mathbf{p}(t))dt = p_{n+1}(t+\tau) - p_{n+1}(t) = 1 \quad (5)$$

There are an infinite number of functions satisfying Equation 5. But since v_{n+1} reflects a uniform time elapse, it is safe to assume that $F(t)$, the antiderivative of v_{n+1}, is a linear function of t:

$$F(t) = \alpha t + k \quad (6)$$

where α and k are constants. According to the first fundamental theorem of calculus, Equation 5 can be written as

$$\int_t^{t+\tau} v_{n+1}(\mathbf{p}(t))dt = F(t+\tau) - F(t) = \alpha(t+\tau-t) = 1 \quad (7)$$

That is,

$$\alpha = \frac{1}{\tau} \quad (8)$$

Therefore,

$$v_{n+1} = \frac{d}{dt}F(t) = \alpha = \frac{1}{\tau} \quad (9)$$

This means at each $(n+1)$-d grid vertex, we only need to use this constant value as the new vector component v_{n+1}. From the above derivation, we can see that streamlines in this $(n+1)$-d field are exactly the pseudo-trajectories we want. The pathlets in the original n-d space can be obtained by slicing the streamlines/pseudo-trajectories along the direction of v_{n+1}. In this way, we convert a time-varying vector field to a high-dimensional steady field.

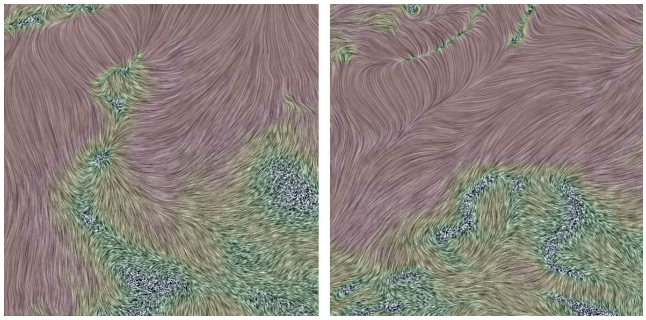


Figure 5: Images at two different time steps, which are generated by projecting the 3D LIC volume back to 2D at each time step. The animation is spatially and temporally coherent.

A number of steady flow visualization algorithms can be easily extended to this high-dimensional space. By applying them to this data representation and projecting the high-dimensional results back to the original space, we are able to derive the accurate time-dependent visualization. For instance, applying LIC can generate dense texture advection that is time-accurate, as shown in Figure 5. Since our focus is pathlet placement, we can also cluster this high-dimensional data representation directly.

5.3 4D Hierarchy Construction

When applying the simplified representation method to our data representation, we choose appropriate parameters [21] (e.g., $A = 0.9, B = 0.9$ for the supernova data set) to form clustering along streamlines in high-dimensional space, i.e., pseudo-trajectories. Mapping the clustering in the high-dimensional space back to the original space can generate spatial and temporal coherent clustering due to the following two reasons. First, the additional vector component in our data representation is a constant. The similarity of two vectors mainly depends on the original vector components. Therefore, the flow clustering of the original field at any time instant is still along the instantaneous streamlines; thus, the spatial coherence is achieved. Second, the additional vector component ensures the clustering in the high-dimensional space along the pseudo-trajectory. Therefore, the flow decomposition of the original field also moves along the pathlines, and the temporal coherence is achieved. Figure 6 demonstrates these two types of coherence with a 2D circular flow.

The effective visualization of the clusters is to render the representative streamlines rather than to show the area (in 2D) or the volume (in 3D) covered by the clusters directly. To obtain such a visualization, one intuitive approach is to first obtain the lower dimensional clusters from the high-dimensional results, and then compute the representative streamlines of the lower dimensional clusters, as shown in Figure 7 (a). However, the cluster centers in the lower dimensional space are not necessarily correlated over time. Therefore, the streamlines tracking from the centers are not correlated temporally. We present an alternative solution to address this issue. First, we compute the representative streamlines from the high-dimensional clusters. Then, we

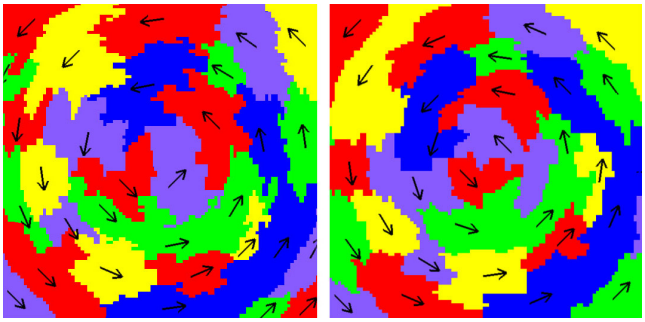


Figure 6: The 2D clustering results of a circular vortex flow at two different time steps (the right one is the later time step), which are obtained by projecting the 3D clustering result in Figure 1 (c). The simplified representation method is used to form the 3D clustering result along the pseudo-trajectory. The 2D cluster coincides with the results presented by Telea and van Wijk, but also moves along the flow over time.

project the streamlines into the lower dimensional space to obtain the representative streamlines of the lower dimensional clusters, as shown in Figure 7 (b). The representative streamlines l_1 and l_2 are also the pathlets of the same pathline, and thus are coherent spatially and temporally.

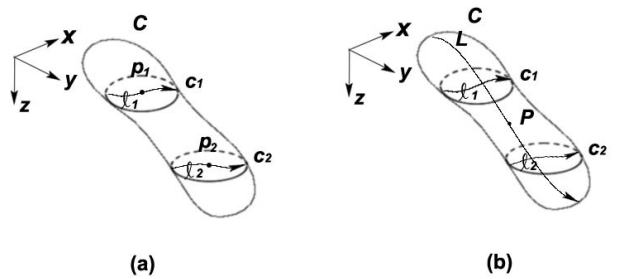


Figure 7: A 2D unsteady vector field is represented as a 3D steady field. C is a 3D cluster. c_1 and c_2 are the 2D projections. (a) p_1 and p_2 are the gravity centers of c_1 and c_2, respectively. The streamlines l_1 and l_2 tracing from them are not necessary correlated. (b) P is the gravity center of C. L is the representative streamline of C from P. The streamlines l_1 and l_2 projected from L are correlated.

A 2D example of the spatial and temporal coherence achieved using our high-dimensional data representation is given in Figure 8. In this figure, (a) and (b) show the visualization of 2D clustering results of two consecutive time steps, derived from the 3D hierarchy construction. The flow structure can be clearly expressed with the movement of the pathlets. (c) superimposes the same regions of (a) and (b) indicated by the red bounding boxes. As we can see, the pink curves are connected with blue ones, which clearly demonstrates the temporal coherence of the time-varying vector field.

To compute the $(n+1)$-d representative streamlines, we use each cluster's gravity center as the seed point and trace the streamlines up and downstream. The bounding box of each cluster is used to limit the length of the streamline. Each initial cluster's bounding box is the corresponding bounding box of the cell. When merged, the new cluster's bounding box is the bounding box of the union of the child bounding boxes. The bounding box is in $(n+1)$-d, where the bounds at the $(n+1)$st axis actually represent the lifespan of a cluster. The root of the cluster tree corresponds to the entire lifespan of the time-varying vector field. After the clustering is completed, given any time t and simplification level l, we can easily find the clusters such that t is in their bounding boxes, and l is greater than or equal to their levels but less than the level of their parents.

6. PARALLEL STREAMLINE GENERATION

After obtaining the seeds from the clustering of the adaptive grid representation, the streamlines are then traced in the original vector field to ensure that the results are numerically accurate. When a field becomes too large to fit into memory, or the computational cost of streamline generation is too high, interactivity cannot be achieved using only a single PC. A traditional solution for this is to distribute the data and the computation among multiple processors. Even though particle tracing is embarrassingly parallel on a shared-memory machine [11], it is nontrivial on a distributed-memory machine because particles can move from one subdomain to the other, incurring frequent interprocessor communication. This is because a particle may frequently travel among partitions assigned to different processors, causing heavy interprocessor communication. In this section, we present our data partition and parallel streamline generation algorithm based on the hierarchical clustering results. Our algorithm achieves a balanced workload and minimizes the communication overhead between processors.

6.1 Data Partition and Distribution

The hierarchical clustering of the vector field provides us with a viable solution to address the issue of interprocessor communication. This is based on an important observation of the relationship between streamlines, clusters, and the underlying flow field. As we can see in Figure 3, a representative streamline is largely contained inside its corresponding cluster, since the cluster is formed along the flow direction. Therefore, if we distribute the clusters among multiple processors, the representative streamline calculations are also distributed, and can be performed independently among processors. Thus, no interprocessor communication is necessary for parallel particle tracing.

Figure 9 illustrates our clustering-based data partition and distribution scheme. First, we select a coarse level from the binary cluster tree, where the number of clusters is usually two or three times the number of processors. The blue nodes in Figure 9 (a) represent the selected clusters in the cluster tree. Then, we estimate the workload associated with each cluster, and partition the clusters in a way so that each processor gets a similar amount of workload. This is achieved by partitioning the whole binary cluster tree into subtrees, where the blue nodes as well as the clusters are assigned to the processors, as shown in Figure 9 (b). At runtime, given a user-specified tree level, each processor selects the clusters by searching its local subtree for particle tracing. Note that the minimal number of streamlines is determined by the initial coarse level chosen. Such a number (e.g. six or twelve as shown in Figure 9) is usually much smaller than the number of streamlines visualized. If a smaller number

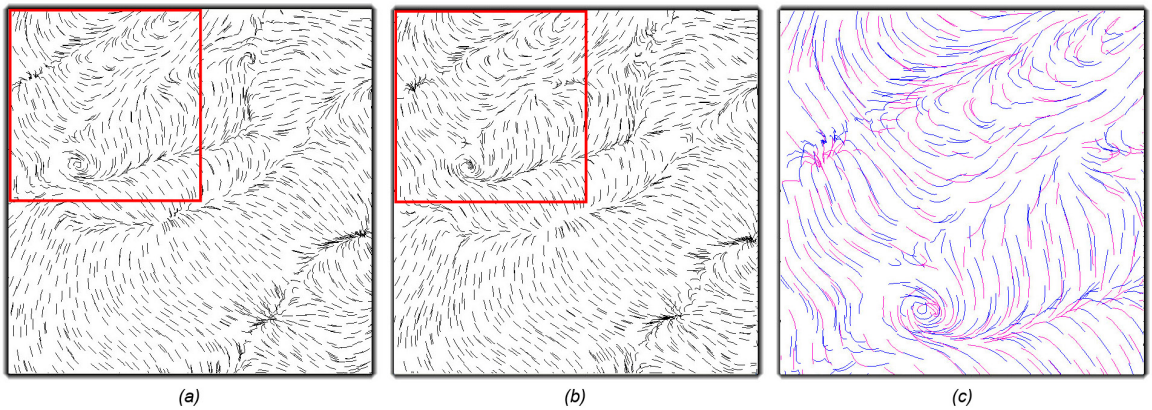


Figure 8: Visualizing a time-varying 2D vector field using the 3D hierarchy construction. (a) and (b) show the rendering of two consecutive time steps. To demonstrate the temporal coherence, we show in (c) the superimposition of the same regions of (a) and (b). They are indicated by the red bounding boxes and rendered in blue and pink, respectively.

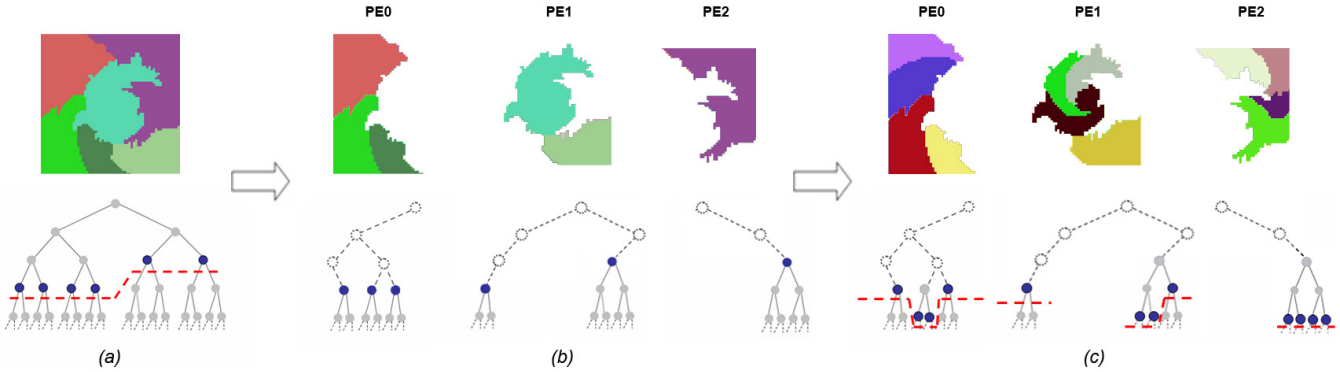


Figure 9: Our clustering-based data partition and distribution scheme. The dotted lines and blue nodes in the binary cluster tree indicate the boundary of each subtree. (b) and (c) show the distribution of six and twelve clusters among three processors (PE0, PE1, and PE2), respectively.

of streamlines is specified, then a cluster at a coarser level may actually be distributed across multiple processors, thus introducing interprocessor communication. However, such a scenario is not typical and is unlikely to happen.

For data partitioning and distribution, workload estimation is the single most important factor that affects the scalability and efficiency of our parallel algorithm. Since there is no runtime communication cost involved if we distribute clusters to processors, the workload is then mainly dictated by the cost of streamline generation. The cost of generating streamline depends on the number of streamlines, as well as the length of each streamline. Both of them vary when the user specifies different tree levels at runtime. It is clear that the number of streamlines equals the number of clusters chosen for the rendering. In addition, a cluster at a coarser level can split into several smaller ones in a finer level, as illustrated in Figure 9 (b) and (c). Thus, if we evenly distribute the clusters at a coarser level, then the workload could be severely unbalanced among processors when the user picks a finer level for rendering. To solve this problem, we add a new attribute to each cluster tree node that records the total number of its descendent nodes. This value is actually the total number of subclusters of a tree node, and can be easily calculated during the stage of binary cluster tree generation. Moreover, because a streamline is formed along its corresponding cluster, the length of the streamline is proportional to the length of the diagonal of the cluster's bounding box. The bounding box is one of a tree node's attributes by default and can be easily obtained. Finally, in our approach, the workload associated with a cluster is estimated as a linear function of the number of subclusters and the length of the diagonal of its bounding box.

6.2 Data Boundary Approximation

A cluster generated using the hierarchical clustering algorithm on the adaptive grid captures the local flow direction and feature. The boundary of such a cluster may go to the finest voxel level when the local feature becomes subtle. However, for the purpose of data partitioning, it may not be necessary to assign the data exactly according to the cluster boundary because of the following reasons. First, it requires a sophisticated data structure to extract the data exactly following the cluster boundary. Second, there is no need to determine the exact portion of data. An approximate boundary will suffice as long as it covers the original boundary of the cluster. Therefore, we advocate an approximate solution here, where we first partition the input volume in an octree style (a sixteen tree in 4D), and then use the bound-

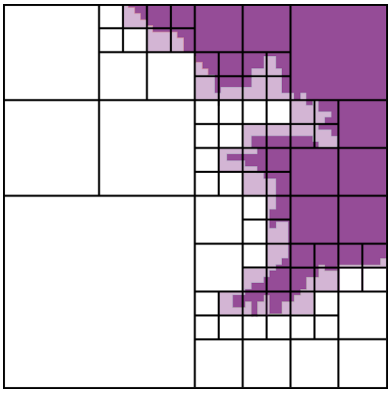


Figure 10: The exact boundary (in dark purple) of a cluster and the actual boundary (in light purple) of the data distributed to a processor.

aries of octree nodes to approximate the actual boundary of a cluster. Figure 10 shows a 2D case where the exact boundary of a cluster is shown in dark purple, and the actual boundary of the data distributed to a processor is shown in light purple. This treatment would bring some overhead of the data distributed to processors, which depends on the block size of the octree leaf nodes. However, the storage overhead is well offset by the simple and easy data partitioning and distribution we gain from this approximation. Actually, our experiments show that the storage overhead introduced by the octree boundary approximation is quite reasonable too. For example, the overhead is 8.5% for the supernova data set, when the octree has a level of seven. The octree itself is around 2MB in size.

6.3 Streamline Generation and Pathlet Rendering

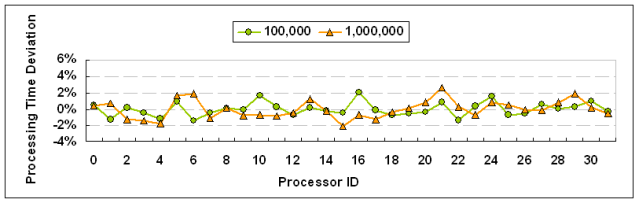
After data distribution, each processor receives portions of the binary cluster tree, the octree for data boundary approximation, as well as the original data blocks assigned to it. For parallel particle tracing, given a user-specified tree level, each processor traverses its partial binary cluster tree, and gets the seed point and the bounding box of each cluster. Then, a processor reads the data blocks according to the octree approximation and traces the particles. Note that the complexity for particle tracing is similar to the case without data boundary approximation, since visiting neighboring data blocks of the siblings or parent octree nodes is a constant time operation. Our clustering algorithm allows each processor to trace each particle independently, delimitated by its corresponding cluster's boundary. All pathlets generated at different processors are then gathered by a host processor. The actual rendering of pathlets is done by the host processor equipped with advanced graphics hardware.

7. RESULTS AND DISCUSSION

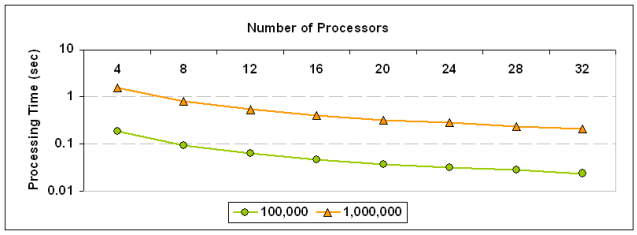
Table 1 lists the two time-varying vector data sets used in our experimental study. For both test data sets, the initial block size for the adaptive grid generation is set to $16 \times 16 \times 16 \times 4$ and the minimal grid size is $4 \times 4 \times 4 \times 1$. We tested our parallel particle tracing method on two PC clusters with different configurations. The first one is a Mac Pro cluster, consisting of eight PCs connected by the Gigabit Ethernet. Each PC has two 2.66GHz Dual-Core Intel Xeon processors, sharing 8.0GB of memory. The second machine is a Cray XT3 MPP system. It has 2068 computing nodes linked by Cray SeaStar Interconnect. Each node has two 2.6GHz AMD Opteron processors, sharing 2.0GB of memory. In our test, we used up to 256 processors for studying the scalability of our method.

data set	$x \times y \times z$	t	total size
supernova	$864 \times 864 \times 864$	100	720.8GB
solar plume	$504 \times 504 \times 2048$	30	174.4GB

Table 1: The two time-varying vector data sets used in our experiment.



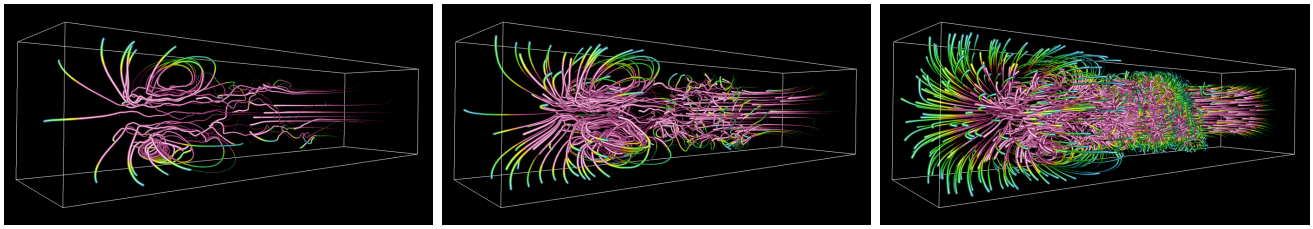
(a)



(b)

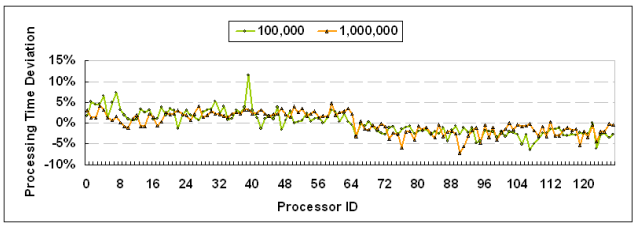
Figure 11: Load balancing and scalability results on the supernova data set with a single time step. (a) shows the processing time deviated from the average on each of 32 Mac Pro processors with two different numbers of streamlines generated: 100,000 and 1,000,000, respectively. The total processing time is 0.0247s for 100,000 streamlines, and 0.2075s for 1,000,000 streamlines. (b) shows the tracing time in the log scale with different numbers of processors for two different numbers of streamlines.

Based on the high-dimensional hierarchical clustering results, our data partition and distribution scheme effectively facilitates load balancing. For example, Figure 11 (a) shows the processing time deviated from the average on each of the 32 Mac Pro processors, when two different numbers of streamlines are used for a single time step of the supernova data set. It can be seen that good load-balancing was achieved, because the processors spent approximately the same amount of time. Figure 11 (b) shows the performance speedup when the number of processors increases. The tracing time for 1,000,000 streamlines on 32 processors is 7.6 times faster than on 4 processors. For time-varying 3D vector fields, Figure 12 (a) shows the processing time deviated from the average on each of the 128 Cray processors, when two different numbers of pathlines are used for 100 time steps of the supernova data set. Again, we can see that a good load-balancing was achieved. Figure 12 (b) shows the performance speedup when the number of proces-

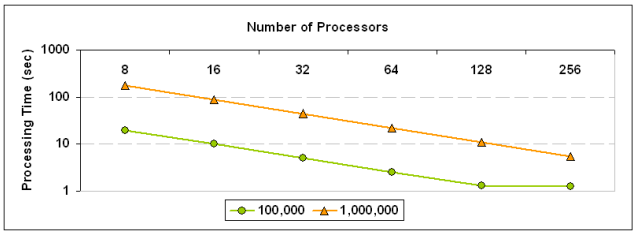


(a) (b) (c)

Figure 13: Levels of detail rendering of the solar plume data set at a single time step. There are a total of 40, 352, and 4,789 pathlets for (a), (b), and (c), respectively. As the number of pathlets increase, more details of the flow structure are revealed, at the expense of possible visual cluttering.



(a)



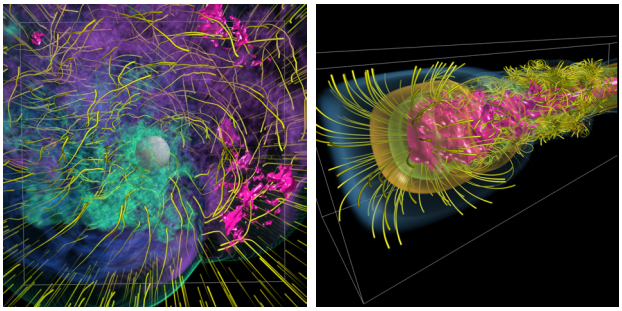
(b)

Figure 12: Load balancing and scalability results on the supernova data set with 100 time steps. (a) shows the processing time deviated from the average on each of 128 Cray processors with two different numbers of pathlines generated: 100,000 and 1,000,000 for each time step, respectively. The total processing time is 1.3563s for 100,000 pathlines, and 11.2678s for 1,000,000 pathlines. (b) shows the tracing time in the log scale with different numbers of processors for two different numbers of pathlines.

sors increases. The tracing time for 1,000,000 pathlines on 256 processors is 31.4 times faster than on 8 processors.

To visualize the pathlets, we used glyphs with the tapering effect [10] rather than arrows. We utilized illuminated lines [14] to render the glyphs, which not only accelerates the rendering, but also achieves more pleasing 3D effects. Figure 13 shows the rendering results of the solar plume data set. In the figure, the magnitudes of the vectors map to the colors of the pathlets. As we can see, pathlets derived from our hierarchical clustering algorithm capture the essence of the internal flow structure. For instance, in Figure 13 (a), there are only a few pathlets showing on the right part of the image, due to the uniform flow structure around that volume region. Furthermore, Figure 14 shows the spatially and temporally coherent rendering results at selected time steps of the supernova data set.

Our visualization allows the scientists to manipulate the pathlet using different representations such as slicing, adjusting transfer function, and superimposing with volume rendering to further explore the internal flow structure. Figure 15 (a) shows the flow on the equatorial plane of the supernova simulation by slicing the pathlets. The pathlets are blended in a depth-accurate fashion with the volume rendering of the supernova's angular momentum scalar field. We added a halo effect to distinguish the pathlets more clearly from the volume rendering. It is clear that the flow is moving clockwise in the outer region of the shock wave, and there is a counter-rotating flow in the inner region. Both of these are the phenomena that the scientists were trying to verify and observed in detail. Another example on the solar plume data set is shown in Figure 15 (b).



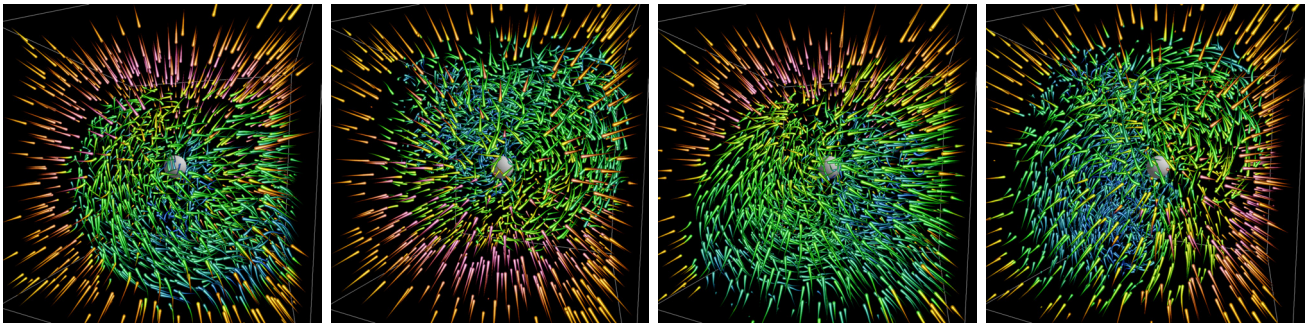
(a) (b)

Figure 15: The view of pathlets mixed with the volume rendering. We add the volume rendering of the supernova's angular momentum in (a) and the solar plume's velocity magnitude in (b) to provide a context for each.

task	**input, output, time**
grid construction	720.8GB, 98MB, 5.5hrs
task	**time**
pathline clustering	15mins
task	**# lines, output**
pathline clustering	1,000,000, 100MB
	100,000, 14.6MB
	10,000, 1MB

Table 2: The preprocessing results of the supernova data set using a single Mac Pro PC.

In adaptive grid construction, the original data is first partitioned into M/G blocks, where M is the total number of cells in the original data, and G is the total number of cells in the initial grid. The adaptive octree (sixteen tree) con-



(a) (b) (c) (d)

Figure 14: Rendering of the supernova data set at four selective time steps. There are a total of 2,486, 2,622, 2,673, and 2,938 pathlets for (a), (b), (c), and (d), respectively. The 3D pathlets are temporally coherent since they coincide with the instantaneous streamlines in 4D.

struction of each block requires $O(G)$ time in the worst case. Thus, the overall complexity of adaptive grid construction requires $O(M)$ time. The size of the adaptive grid is between $O(M/G)$ and $O(M/Q)$, where Q is the total number of cells in the minimal grid. In practice, the size is relatively close to $O(M/G)$. The time to construct the adaptive grid for the supernova data set is listed in Table 2.

The complexity of our high-dimensional clustering algorithm is the same as the one presented in [21], which requires $O(N \log N)$ time. In our approach, N is the total number of cells in the adaptive grid. As listed in Table 2, it takes 15 minutes to cluster all the pathlines for the supernova data set using a single Mac Pro PC. The binary cluster tree constructed has a size of $O(PT)$, where P is the maximum number of pathlets shown in a time step, and T is the number of time steps. Obviously, the upper bound for PT is N. In practice, PT is usually much smaller than N. For example, when the maximum number of pathlines is 1,000,000, the size of the binary cluster tree is 100MB for the supernova data set, which is split and distributed among processors. The height of the cluster tree is $O(\log PT)$, which is also the time to find each cluster with a particular level of detail and a time step. Therefore, this low time complexity allows us to interactively visualize time-varying 3D vector fields at different levels of detail using the binary cluster tree.

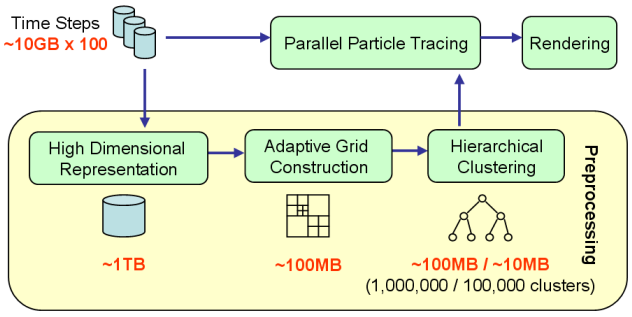


Figure 16: The pipeline of our parallel hierarchical visualization of terascale time-varying 3D vector fields.

In summary, Figure 16 illustrates our pipeline for parallel hierarchical visualization of terascale time-varying 3D vector fields. Through the construction of adaptive grid and hierarchical clustering on a single PC, we build a condensed, efficient representation from the original large-scale, time-varying data in the high-dimensional space for subsequent particle tracing and rendering. Our adaptive grid solution essentially allows us to perform clustering using only a single desktop PC and the sixteen tree partition in 4D provides us a way to index the terabyte data for out-of-core parallel particle tracing. In our current implementation, only particle tracing is conducted in parallel. However, since each block is processed independently, the construction of the adaptive grid can be easily parallelized by distributing the blocks among multiple processors. Additionally, there is also a need of research on I/O. The strategy presented in [28] can be used to reduce or hide I/O overhead in parallel pathline tracing.

8. CONCLUSION AND FUTURE WORK

The ability to visualize large-scale time-varying vector field data is not generally available. Many important aspects of the data are thus not looked at. In this paper, we have developed a scalable, parallel pathline construction method for the visualization of time-varying 3D vector fields. Unlike previous parallel particle tracing designs, our method is based on a new 4D representation of the time-dependent vector fields and a hierarchical clustering of the corresponding 4D space. As shown by our test results, our method allows for interactive navigation of a large time-varying flow field using a parallel computer while keeping the communication overhead to a minimum. Our algorithm achieves a well-balanced workload among the processors, leading to a highly scalable solution. Incorporating this new capability into an existing scalable parallel volume renderer, we are able to simultaneously visualize 3D scalar and vector fields at high fidelity and interactivity.

Our high-dimensional data representation unifies vector field visualization methods based on streamlines and pathlines. Many of the streamline-based visualization methods previously designed for steady flow data, such as streamline placement, streamsurface construction, LIC, and clustering, can be applied directly to our 4D data representation to generate both spatially and temporally coherent results for unsteady flow fields. Our current streamline generation design relies on the hierarchical clustering results derived from an adaptive grid, which generally leads to good approximation in contrast to the results derived from the original voxel-level grid. There is a need of quantitative study on the error introduced by this approximation. We would also like

to add the ability to construct feature surfaces from time-accurate pathlines as well as the ability to perform feature extraction and tracking, guided by interactive visualization of an evolving multifield data.

Acknowledgements

This research was supported in part by the NSF through grants CCF-0634913, CNS-0551727, OCI-0325934, and CCF-9983641, and the DOE through the SciDAC program with Agreement No. DE-FC02-06ER25777, DOE-FC02-01ER41202, and DOE-FG02-05ER54817. We wish to thank the anonymous reviewers for their constructive comments, and the Cray XT3 time provided by the Pittsburgh Supercomputing Center.

9. REFERENCES

[1] J. P. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. C. Law, and M. E. Papka. Large-Scale Data Visualization Using Parallel Data Streaming. *IEEE CG&A*, 21(4):34–41, 2001.

[2] S. Bachthaler, M. Strengert, D. Weiskopf, and T. Ertl. Parallel Texture-Based Vector Field Visualization on Curved Surfaces Using GPU Cluster Computers. In *Proc. Eurographics PGV Sym. '06*, pages 75–82, 2006.

[3] R. W. Bruckschen, F. Kuester, B. Hamann, and K. I. Joy. Real-Time Out-of-Core Visualization of Particle Traces. In *Proc. IEEE PGV Sym. '01*, pages 45–50, 2001.

[4] B. Cabral and L. C. Leedom. Highly Parallel Vector Visualization Using Line Integral Convolution. In *Proc. SIAM PPSC Conf. '95*, pages 802–807, 1995.

[5] D. Ellsworth, B. Green, and P. J. Moran. Interactive Terascale Particle Visualization. In *Proc. IEEE Visualization Conf. '04*, pages 353–360, 2004.

[6] M. Griebel, T. Preusser, M. Rumpf, M. A. Schweitzer, and A. Telea. Flow Field Clustering via Algebraic Multigrid. In *Proc. IEEE Visualization Conf. '04*, pages 35–42, 2004.

[7] B. Heckel, G. H. Weber, B. Hamann, and K. I. Joy. Construction of Vector Field Hierarchies. In *Proc. IEEE Visualization Conf. '99*, pages 19–26, 1999.

[8] J. L. Helman and L. Hesselink. Visualizing Vector Field Topology in Fluid Flows. *IEEE CG&A*, 11(3):36–46, 1991.

[9] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-Eulerian Advection for Unsteady Flow Visualization. In *Proc. IEEE Visualization Conf. '01*, pages 53–60, 2001.

[10] B. Jobard and W. Lefer. Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Proc. Eurographics Visualization in Scientific Computing Workshop '97*, pages 43–55, 1997.

[11] D. A. Lane. UFAT - A Particle Tracer for Time-Dependent Flow Fields. In *Proc. IEEE Visualization Conf. '94*, pages 257–264, 1994.

[12] D. A. Lane. Parallelizing A Particle Tracer for Flow Visualization. In *Proc. SIAM PPSC Conf. '95*, pages 784–789, 1995.

[13] Z. Liu, R. J. Moorhead, and J. Groner. An Advanced Evenly-Spaced Streamline Placement Algorithm. In *Proc. IEEE Visualization Conf. '06*, pages 965–972, 2006.

[14] O. Mallo, R. Peikert, C. Sigg, and F. Sadlo. Illuminated Lines Revisited. In *Proc. IEEE Visualization Conf. '05*, pages 19–26, 2005.

[15] N. Max and B. Becker. Flow Visualization Using Moving Textures. In *Proc. ICASW/LaRC Visualizing Time-Varying Data Sym. '95*, pages 77–87, 1995.

[16] A. Mebarki, P. Alliez, and O. Devillers. Farthest Point Seeding for Efficient Placement of Streamlines. In *Proc. IEEE Visualization Conf. '05*, pages 479–486, 2005.

[17] S. Muraki, E. B. Lum, K.-L. Ma, M. Ogata, and X. Liu. A PC Cluster System for Simultaneous Interactive Volumetric Modeling and Visualization. In *Proc. IEEE PVG Sym. '03*, pages 95–102, 2003.

[18] G. Scheuermann, H. Krüger, M. Menzel, and A. P. Rockwood. Visualizing Nonlinear Vector Field Topology. *IEEE TVCG*, 4(2):109–116, 1999.

[19] H.-W. Shen and D. L. Kao. A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields. *IEEE TVCG*, 4(2):98–108, 1998.

[20] C. T. Silva, Y.-J. Chiang, J. El-Sana, and P. Lindstrom. Out-of-Core Algorithms for Scientific Visualization and Computer Graphics. IEEE Visualization Conf. '02 Course Notes, 2002.

[21] A. Telea and J. J. van Wijk. Simplified Representation of Vector Fields. In *Proc. IEEE Visualization Conf. '99*, pages 35–42, 1999.

[22] H. Theisel, J. Sahner, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Extraction of Parallel Vector Surfaces in 3D Time-Dependent Fields and Application to Vortex Core Line Tracking. In *Proc. IEEE Visualization Conf. '05*, pages 631–638, 2005.

[23] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-Core Streamline Visualization on Large Unstructured Meshes. *IEEE TVCG*, 3(4):370–380, 1997.

[24] J. J. van Wijk. Image Based Flow Visualization. *ACM TOG*, 21(3):745–754, 2002.

[25] C. Wang, J. Gao, and H.-W. Shen. Parallel Multiresolution Volume Rendering of Large Data Sets with Error-Guided Load Balancing. In *Proc. Eurographics PGV Sym. '04*, pages 23–30, 2004.

[26] D. Weiskopf, G. Erlebacher, and T. Ertl. A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields. In *Proc. IEEE Visualization Conf. '03*, pages 107–114, 2003.

[27] B. N. Wylie, C. J. Pavlakos, V. Lewis, and K. Moreland. Scalable Rendering on PC Clusters. *IEEE CG&A*, 21(4):62–70, 2001.

[28] H. Yu, K.-L. Ma, and J. Welling. A Parallel Visualization Pipeline for Terascale Earthquake Simulations. In *Proc. ACM/IEEE SC Conf. '04*, 2004.

[29] M. Zöckler, D. Stalling, and H.-C. Hege. Interactive Visualization of 3D-Vector Fields Using Illuminated Streamlines. In *Proc. IEEE Visualization Conf. '96*, pages 107–113, 1996.

[30] M. Zöckler, D. Stalling, and H.-C. Hege. Parallel Line Integral Convolution. *Parallel Computing*, 23(7):975–989, 1997.