A Combined Eulerian-Lagrangian Data Representation for Large-scale Applications

Franz Sauer, *Member, IEEE,* Jinrong Xie, *Member, IEEE,* and Kwan-Liu Ma, *Fellow, IEEE*

Abstract—The Eulerian and Lagrangian reference frames each provide a unique perspective when studying and visualizing results from scientific systems. As a result, many large-scale simulations produce data in both formats, and analysis tasks that simultaneously utilize information from both representations are becoming increasingly popular. However, due to their fundamentally different nature, drawing correlations between these data formats is a computationally difficult task, especially in a large-scale setting. In this work, we present a new data representation which combines both reference frames into a joint Eulerian-Lagrangian format. By reorganizing Lagrangian information according to the Eulerian simulation grid into a "unit cell" based approach, we can provide an efficient out-of-core means of sampling, querying, and operating with both representations simultaneously. We also extend this design to generate multi-resolution subsets of the full data to suit the viewer's needs and provide a fast flow-aware trajectory construction scheme. We demonstrate the effectiveness of our method using three large-scale real world scientific datasets and provide insight into the types of performance gains that can be achieved.

Index Terms—Flow visualization, particle data, volume data, multi-resolution, large-scale data, data structures.

1 Introduction

The use of both Eulerian and Lagrangian data representations is becoming increasingly popular among today's scientists. Many current scientific simulations utilize "hybrid codes" which represent data in both reference frames as the simulation runs [1], [2]. As a result, scientists often choose to output simulation data in both formats. Since the Eulerian and Lagrangian specifications each have a unique set of advantages and disadvantages, scientists tend to switch between each to study different phenomena.

In the Eulerian representation, values are measured on a fixed spatial grid spanning the simulation domain (scalar or vector fields). Such a grid is useful as it allows scientists to study data fluctuations at unchanging locations in the system of study. However, obtaining information from areas that lie between grid points is limited to interpolation. The Lagrangian representation follows the movement of discrete parcels throughout the domain (particle data). The spatial evolution of these parcels provides a unique perspective that is missing in the Eulerian specification. However, due to their free-form nature, it is not always guaranteed that parcels will be present in a particular area of interest. More information can be found in [3].

Traditionally, Eulerian and Lagrangian data is stored and accessed separately for different analytical tasks. However, drawing information from both representations is gaining popularity as it allows scientists to utilize the advantages of both reference frames [4], [5], [6], [7], [8], [9]. These examples show the benefit of being able to exploit the interplay between each format to gain a better understanding of the data and its properties. Unfortunately, the differences inherent in each format make this combination computationally intensive, especially at large scales. This is namely due to the lack of structure generally found in raw simulation particle data and makes correlating sets of particles with associated Eulerian grid points difficult. Scientists are looking for new solutions to this growing problem.

Current techniques, such as octrees and k-d trees, can spatially organize both the Lagrangian particles and unstructured Eulerian grids with respect to the overall simulation domain. Such organization techniques have been shown to improve computational efficiency when working with either of these data formats (Section 2.1). If both the Lagrangian and Eulerian data are organized using similar techniques, correlations between the data types can be found more efficiently by matching values in similar spatial locations. While this can be a useful workaround, it still treats both data representations as separate entities and uses an intermediate mechanism to draw connections. In this work, we present a new application of indexing-based data structures that can combine both reference frames into a joint Eulerian-Lagrangian representation. By eliminating the "middle-man", we can achieve significant improvements over existing methods and provide an efficient out-of-core means of sampling, querying, and operating with both representations simultaneously.

This combined Eulerian-Lagrangian representation also opens new venues in the area of intelligent multi-resolution data sampling. As we push towards ever increasing dataset sizes, the ability to extract and visualize subsets of the full data becomes a necessity. This scalability concern is twofold as it relates to both the computational ability of the system to handle the data amount as well as the perceptual ability of users to understand the amount of information presented. By combining the data formats, we can efficiently sample subsets from one representation using information found in the other. Such sampling schemes can provide more physical meaning over schemes which use arbitrary random

• *Franz Sauer, Jinrong Xie, and Kwan-Liu Ma are with the Department of Computer Science, University of California at Davis, Davis, CA, 95616. E-mail: fasauer@ucdavis.edu, jrxie@ucdavis.edu, ma@cs.ucdavis.edu*

Manuscript received April 19, 2005; revised September 17, 2014.

sampling or selection based on information from only one of the reference frames.

In this paper, we present our new combined Eulerian-Lagrangian data representation and make the following contributions:

- We introduce a new application of indexing-based data structures that can combine the information from both the Eulerian and Lagrangian reference frames into a single format.
- We develop a set of accompanying analytical tools to perform fast disk queries and other out-of-core operations using information from both representations simultaneously.
- We extend this design to efficiently sample multi-resolution subsets from large-scale data.

We demonstrate the effectiveness of our method through case studies using real world large-scale datasets in fusion, combustion, and cosmology and illustrate its numerous advantages. Furthermore, we provide performance results to give an estimate into the amount of improvement that can be gained by adopting this design.

2 Background

When it comes to managing large-scale data, nearly all techniques choose to intelligently restructure or partition the data to improve the efficiency of various analytical tasks. Many methods have been shown to work well when dealing with Eulerian or Lagrangian datasets separately (Section 2.1). However, due to fundamental differences between the reference frames, using these established techniques to support operations involving both data types simultaneously will result in certain drawbacks (Section 2.2).

2.1 Related Work

What little work exists on utilizing both the Eulerian and Lagrangian reference frames simultaneously is often limited to domain specific areas. Many large-scale scientific flow simulations in areas such as fusion [1] or combustion [2] utilize the popular particle-in-cell method [10] for computation and to represent data in both reference frames. However, the output from such simulations is nearly always in the form of Eulerian volume data or Lagrangian particle data which are processed and analyzed separately depending on the task at hand. Other notable examples include a numerical modeling framework for Tribology simulations [11], the deformation and interaction of structures and fluids [12], [13], and ocean current mixing [14]. Once again, these works focus on combining the reference frames for domain specific modeling and simulation purposes, rather than the postprocessing analysis needed to fully study the rich datasets produced. Aimed at computer graphics, Fan et al. [15] utilized a joint method that combines Lagrangian-like grids with an Eulerian solver that can simulate deforming solids. Their work utilizes both reference frames, but does not incorporate the traditional particle/volume data paradigm as is the focus of our work. Overall, a generalized framework focused on combining each of the references frames to enhance analysis and visualization of existing datasets has yet to be thoroughly explored, especially in a large-scale setting.

When it comes to postprocessing frameworks designed to support efficient analysis and visualization, nearly all approaches focus on organizing either data type on its own. The vast majority of techniques are in the form of a hierarchical space partitioning data structure and have been applied successfully for either case. The octree is one popular form of a uniform hierarchical structure and can be used to adaptively partition space for techniques such as fast isosurface generation [16], particle collision detection [17], volume rendering [18], and streamline construction [19]. Since each adaptive tree level is uniformly partitioned, the octree works best (i.e. requires the least depth) when objects are evenly distributed throughout the simulation domain.

On the other hand, many approaches favor a non-uniform hierarchical partitioning. Yu et al. [20] utilize an adaptive binary hierarchy of grid locations based on local flow patterns for parallel pathline construction. Wächter and Keller [21] utilize a bounding interval hierarchy (an extension of the standard k-d tree) for the efficient organization of geometric primitives to improve ray tracing. More recently, Garth and Joy [22] developed their cell tree approach, which is a further extension of the bounding interval hierarchy approach, for fast cell searching in unstructured Eulerian grids. Furthermore, Andrysco and Tricoche [23] developed an efficient storage scheme for these tree-like structures called matrix trees.

Some notable exceptions include the work done by Ellsworth et al. [24] which use a space filling curve to organize large-scale particle data and minimize disk seeks, and Treib et al. [25] which use blocking and a GPU based compression/decompression scheme for large-scale volume visualization. Another common method is to use statistical techniques to perform in situ sampling of the full resolution data. Woodring et al. [26] apply such a technique to identify and save subsets of large-scale particle data directly from the simulation itself. In addition, Peterka et al. [27] choose to organize the data temporally as well by utilizing 4D blocks representing spatial and temporal neighborhoods for parallel particle tracing. Lastly, Su et al. [28], [29] utilize bitmap indices for efficient data sampling and correlation analysis between variables. All of these approaches are effective in handling either Eulerian or Lagrangian data, but make no attempt to combine the two for enhanced analysis and visualization as we do in this work.

The use of multi-resolution techniques for managing large-scale visualizations is also heavily relied on in order to alleviate both computational and perceptual scalability concerns. In terms of computational complexity, data reduction techniques are often employed to intelligently collect subsets from the full dataset on disk. For example, Fraedrich et al. [30] utilize an octree based level-of-detail approach to handle large-scale cosmological data, and Hopf et al. [31] use a hierarchical PCA based approach for particle splatting. In terms of perceptual complexity, the amount of information presented to the viewer must also be limited. For example, Bürger et al. [32] employ an importance based approach to selecting and presenting particles representative of specific flow features of interest. In this work, we demonstrate how we can utilize our combined data representations to quickly and intelligently gather new multi-resolution subsets of large-scale datasets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2016.2620975, IEEE Transactions on Visualization and Computer Graphics

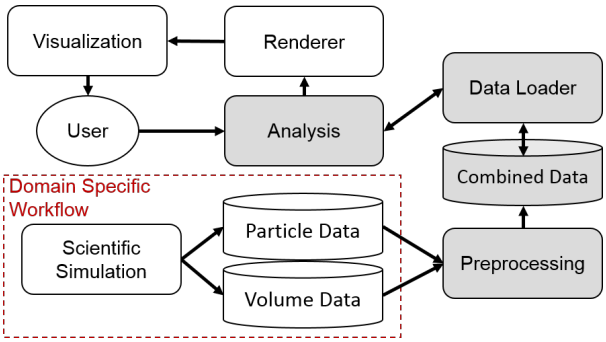

Fig. 1. An overview of a typical workflow using our design. Sections highlighted in gray represent components utilizing the techniques presented in this paper.

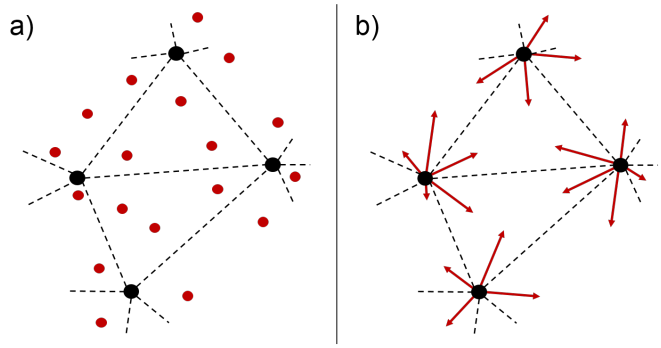

Fig. 2. a) A 2D example of an unstructured Eulerian grid (black) and a set of Lagrangian particles (red) as separate entities. b) The two data types can be combined by representing the particles as a set of vectors originating from its nearest Eulerian grid point.

2.2 Correlating Particle and Volume Data

While the aforementioned techniques have been be successful in their respective areas, they do not focus directly on tasks that require quickly building correlations between the Eulerian and Lagrangian representations. Correlating particles with unstructured grid data traditionally involves matching data values according to their spatial position. In other words, for any given particle, we need to determine which grid point is closest in terms of Euclidean distance, reducing this problem to a nearest neighbor search. As described previously, the vast majority of tasks that require solving the nearest neighbor problem (often for interpolation purposes) utilize hierarchical space partitioning to identify a local neighborhood and limit the number of data points that need to be searched.

However, when it comes to correlating large-scale particle and volume data, these structures have a number of drawbacks. First, trees will need to accommodate the increasing size of simulation data by becoming deeper. Depending on the type of tree that is used, this can increase storage overheads since the spatial boundaries of certain nodes may need to be stored on disk. Next, this type of partitioning only limits the number of objects that need to be searched (as opposed to eliminating the search entirely). Lastly, there is the need for inter-leaf searching. It is not uncommon for a particle and its associated grid point to be found in separate portions of the spatial partitioning, and therefore are part of separate leaves in the data structure. Searching all potential candidates involves not only searching the current partition, but all neighboring partitions as well. The work presented in this paper focuses on developing a method without these limitations.

3 Methods

Like many traditional techniques, our data representation is based on a reorganization of raw simulation data into a more computationally friendly format. The main difference and advantage of this approach is that the particle data is spatially organized in relation to the Eulerian simulation grid itself allowing particle information to be "embedded" into each grid location as a point cloud. Such a reorganization allows for extremely fast access of both correlated data types by exploiting data locality on disk and in memory.

3.1 Overview

A typical workflow using our design can be seen in Figure 1. Each scientific field of study has its own domain specific workflow for producing simulation results. In the end, the data is output separately as volume and particle counterparts. Just like traditional reorganization techniques, our current implementation utilizes a preprocessing step to transform this raw simulation data into our desired format. However, it is also possible to eliminate this step by organizing the data in situ during the simulation time and saving it directly in the combined format. This is discussed later in Section 3.2.5. Once in the desired format, the combined data is accessed using an intelligent loader which can perform out-of-core techniques to extract desired subsets from the full dataset. This is then further processed using analysis tools and rendered into a comprehensive visualization.

3.2 Data Structure

The joint data structure consists of a spatial reordering of the particle information according to the Eulerian simulation grid itself. As separate entities, the particle and grid data have no predetermined correlation between one another. However, we can associate each particle with a unique grid location based on their Euclidean separation distance. As a result, each grid point will have an associated set of particles which can be represented as a point cloud. As shown in Figure 2, this point cloud is described as a set of vectors originating from the grid location and pointing towards the particle position. Instead of simply assigning particles to its associated grid point, using this vector based representation has advantages when performing certain joint operations. These are described in more detail in later sections.

3.2.1 Eulerian-Lagrangian Unit Cell

Such a representation results in a "unit cell" which contains all of the Eulerian and Lagrangian information associated with it. The Eulerian information includes the location of the grid point, any scalar or vector values associated with it, and a list of neighboring grid points. The Lagrangian information includes a list of vectors pointing to each particle position as well as an ID and any scalar or vector values associated with it. These unit cells are then used as discrete building blocks to represent larger portions of the domain.

One can think of the Eulerian grid point as being at the center of a cell whose bounds are represented via a Voronoi decomposition. Any location within the cell boundary has this particular grid point as its nearest neighbor. The shape of the cell bounds depends on the number and distance of neighboring grid points. Since the information from both frames is already contained within each cell, we can use out-of-core techniques to load and operate on individual or groups of unit cells at a time. This is especially necessary when the full dataset sizes are too large to fit in memory.

3.2.2 Storage on Disk

While both the grid and particle data have a 3D representation, they must be stored in a 1D linear fashion on standard hard drives. This is done by sorting information such that all entities associated with a unit cell (e.g., the associated particles) are grouped into contiguous chunks. Since the number of entities per unit cell is not constant, we use a set of helper files which contain start read indexes and counts for locating the information corresponding to a particular unit cell. This is similar to the "compressed sparse row (CSR)" data format which is commonly used to store large matrices. This allows us to organize the data on disk so that all of the information corresponding to a particular unit cell can be accessed quickly and independently.

We store the Eulerian grid (the xyz positions of each grid point) in a linear fashion using the same ordering as produced by the simulation. Since there are exactly three values for each grid point, we can easily determine where in the file to seek to for the associated information given the grid point index. We also store a list of local neighbors to complete the mesh-like representation of the Eulerian grid. We sort each neighboring index by grouping together all the neighbors for each grid location, and ordering the groups according to the original grid index. Since the number of neighbors per grid location might not be constant, we include an additional helper file as described earlier.

Next, we store the time-varying data on a per timestep basis (a separate group of files representing different data variables for every available timestep). If the simulation produces particle and volume data at different time resolutions, then this simply results in a subset of timesteps on disk which are missing corresponding Eulerian or Lagrangian variables. More details are discussed in Section 3.3.3. We save each Eulerian variable in a linear manner. Each variable is placed in its own file, and since the number of values per grid point is constant per variable, no helper file is needed.

Lastly, we store the Lagrangian particle information. Since each unit cell consists of a (non-constant) number of particles, we store the particle information in a manner similar to the local neighbor lists. We reorder the particle data by grouping all particles associated with a unit cell together, and order the groups according to the Eulerian grid point index. The particle positions are saved relative to the position of its associated grid point in the vector based format described earlier. The particle variables (ID's, scalar values, etc.) are saved in the same order as the particle positions. Since the number of particles per unit cell is not constant, we need one additional helper file containing a start read location and particle count for each unit cell.

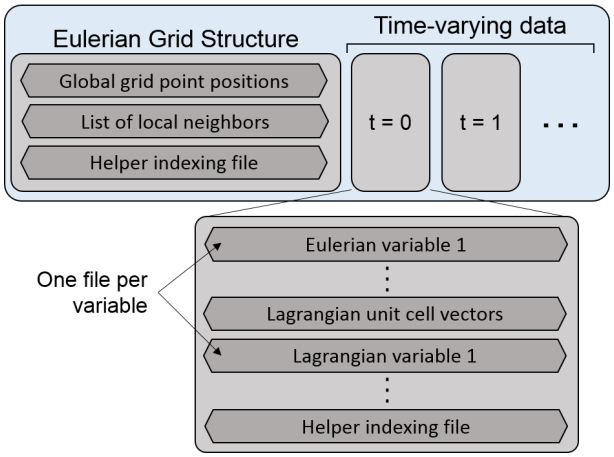

Fig. 3. A representation of the joint data structure as files on disk. Each dark gray block represents a separate file.

The reason we do not interleave all the data associated with our unit cell (and save information as separate files) is to minimize disk seeks and the amount of unnecessary information placed in main memory when fetching data blocks from disk. In many instances, users will only be studying a few variables at a time allowing the system to easily ignore all other variables and those respective files. An overview of this file representation can be seen in Figure 3. This data representation allows the system to quickly access all of the Eulerian and Lagrangian information associated with a particular unit cell of interest in a time that is independent of the size of the dataset. The ability to operate on all of the information contained in each unit cell separately is crucial for certain out-of-core operations.

3.2.3 Storage Overhead

Representing simulation data in this format on disk does incur an additional storage overhead. This is due to the helper files which allow the system to immediately locate the corresponding (neighbor or particle) information for a particular unit cell. Since all other files are simply a reordering of the raw simulation data values, they do not take up any additional space. Recall that each helper file consists of a list of start read indexes as well as a list of counts. We include one helper file for the Eulerian grid structure to help index the neighbor lists and additional helper files (one per timestep) to help index the particle data. The total amount of information contained in each helper file is two integer values times the total number of grid points (unit cells). If there are n grid points and t_p particle timesteps in total, then the additional storage required scales linearly with each:

$$2n(t_p + 1) \text{ integer values}$$

In comparison, a hierarchical space partitioning structure would also include an additional storage overhead. For example, an octree needs to save information pointing to the objects (grid points and particles) contained in the space partition represented by each of its leaf nodes. If we assume a full octree with depth, $d \geq 0$, then the total number of nodes becomes $(8^{d+1} - 1)/7$ with 8^d nodes as leaves. Since a full octree is a structured data structure (akin to regular

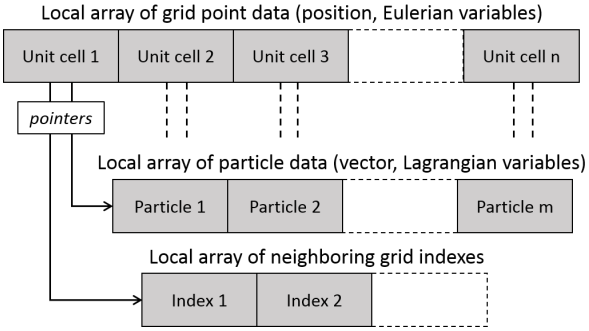

Fig. 4. A representation of the joint data structure once in memory.

grid indexing) we do not need to store the addresses of children nodes in our file since it can be computed on the fly. Next, we can reorganize and linearly group together the raw simulation data according to the tree leaf nodes. Similar to the helper files used in our implementation, each leaf only needs to store a start read location and count once per timestep for the Lagrangian data. However, we need two helpers files for the Eulerian data, one to index the grid point neighbor list and one to index which grid points fall into each leaf node. As a result the additional storage required then becomes:

$$2 * 8^d(t_p + 2) \text{ integer values}$$

Assuming a generally even distribution of data points in the domain we can then choose the depth of the octree so that each leaf node contains approximately one grid point. In this case, $8^d \sim n$ and we end up with a storage overhead that scales similarly for both the octree and our technique. However, an unbalanced octree or a non-uniform partitioning technique could result in an even higher storage overhead since it must contain additional information specifying the exact shape of the hierarchical domain decomposition.

3.2.4 Structure in Memory

Although information is stored as separate files on disk, once loaded into memory, the data for each unit cell is stored more closely for easy access. Figure 4 shows an overview of this structure for a single timestep. A block of memory is used to store basic unit cell information, such as the position of the grid point and any Eulerian variables, as well as the number of associated particles and neighboring grid points. The helper indexes are now replaced by pointers to a particle array containing their position vectors and any Lagrangian variables, and an array of neighboring grid indexes. Each unit cell memory block that is loaded from disk is grouped into an array which represents larger portions of the overall domain.

Special care must be taken when reading the neighboring grid indexes into memory. Since this method supports conditional unit cell queries (Section 3.3.1), it is possible that a unit cell does not meet the requirements of the query and is never loaded into memory. This results in a mismatch between the indexes contained in the list of local neighbors (on disk) and the index of those neighbors in the array as (in memory). As a result, we need to generate and store a map between the grid index of the neighboring cell that is read and the correct index of that cell in the array in memory. This is constructed concurrently as data is read from disk according to the conditional query and an example can be seen later in Algorithm 1. Our technique references this map when it needs to locate neighboring unit cells.

3.2.5 Preprocessing Costs

Like many other structural implementations, the one presented in this paper uses a preprocessing step to reorder raw data into the format described in the previous sections. The main cost and function of this preprocessing step requires determining which particles are associated with a particular Eulerian grid point and then sorting them according to the corresponding unit cell ID. This is done via a nearest neighbor search which minimizes the Euclidean distance between particles and grid points. To minimize the computational cost of the nearest neighbor search, we spatially partition the domain according to a regular grid, populating each region as data is read from disk. This reduces the number of elements that need to be searched. Note that the nearest neighbor search step is only necessary when working with irregular Eulerian meshes. If it uses a regular grid, then the partitioning step automatically assigns each particle to a grid point. Once each entity is assigned to a unit cell, they are sorted by ID, placing associated entities into contiguous chunks. Counts of the number of entities per unit cell are maintained during the sorting process to generate the helper files. Performance tests of each preprocessing step is discussed later in Section 4.4.

While the preprocessing costs are only a small one time cost, it is possible to eliminate them by directly saving data from a simulation into this format. This can be done with very little modification to the simulation itself because the approach used here orders information based on the already defined Eulerian simulation grid. In a simulation using a distributed domain decomposition (which describes a vast majority of today's scientific simulations), each processing node can independently organize particles into vector based point clouds and dump its own image of the domain space into our desired format. In fact, simulations that utilize the particle-in-cell method [10] in their computation already have some form of particle to grid association built in. The only additional inter-node communication costs involved would be a "gather" of the number of particles associated with each grid point in order to construct the helper files described earlier. Future work is still necessary to quantify the computational and memory costs imposed on moving the preprocessing cost in situ.

3.3 Data Operations

Next, we describe some of the fundamental data operations provided by our design that can be utilized by analytical tasks that use both particle and volume representations simultaneously. One of the main driving motivations behind this work is the ability to operate on unit cells individually, allowing all the following tasks to be done in an out-of-core manner with large-scale datasets.

3.3.1 Conditional Unit Cell Queries

Having immediate access to corresponding particle or volume data is the primary application of this data structure. Whether one is working in either frame, the unit cell based format allows for a quick and easy retrieval of data in the counterpart representation. For example, a user could identify a feature of interest in the volume data, which is represented as a set of Eulerian grid points. We can then use the indexes of the grid points as well as the helper file to query the associated particle data for that feature from disk. This will load only the particles which are spatially contained within the feature of interest allowing users to investigate additional properties using Lagrangian data values.

Another useful task is the ability to query both Eulerian and Lagrangian information based on variables found in either representation directly from disk. Using our method, we can choose which unit cells to load into memory based on any number of raw or derived values. For example, we could load only unit cells whose Eulerian values lie within a certain range. This is done by iterating over every unit cell and reading only the Eulerian data values in question to test against the user defined criteria. If the criteria is met, the rest of the information associated with the unit cell (its position, Lagrangian information, etc.) is then also loaded into memory. Note that this is possible because all of the information associated with a specific unit cell can be quickly located. If the criteria is not met, then the cell is skipped via a seek operation to the location of the next unit cell in our files. An example of such a query is given as pseudocode in Algorithm 1. Recall that we also need to keep an updated map of correct neighboring indexes since some unit cells may not be loaded into memory as described previously in Section 3.2.4.

We can also take the reverse approach and query unit cells based on Lagrangian information. In this case, we read Lagrangian variables from all of the particles associated with a particular unit cell and use the distribution of values in the testing criteria. Note that since the system maintains references between particles loaded into memory and their corresponding unit cells, this type of query can also be performed quickly. Methodically extracting subsets from the full dataset is the basis behind feature extraction in many scientific fields. Our design extends this into both reference frames by being able to efficiently isolate Eulerian features using Lagrangian information and vice versa.

Algorithm 1 Eulerian Conditional Query Example

```
1:  procedure CONDITIONAL UNIT CELL LOAD
2:      num_skipped = 0
3:      for each unit cell with index i do
4:          s = readEulerian(i)                    ▷ read Eulerian value
5:          if conditional(s) = true then          ▷ meets condition
6:              readGridPosition(i)
7:              r, c = readGridHelper(i)           ▷ (read start, count)
8:              readNeighborList(r, c)
9:              r', c' = readParticleHelper(i)
10:             readParticleData(r', c')
11:             update_map(i, num_skipped)         ▷ build map
12:         else                                   ▷ fails condition
13:             update_map(i, "not loaded")
14:             num_skipped += 1
15:         end if
16:     end for
17: end procedure
```

3.3.2 Transforming Between Reference Frames

The next task involves correlating Lagrangian and Eulerian data values with one another and the ability to map information into its counterpart reference frame. Since both data types are already organized into unit cells, this conversion can be done extremely quickly. For example, we can map Lagrangian variables onto the Eulerian grid points by averaging the information of all of the particles within each unit cell. Furthermore, we can use the distance of the particle to the grid point as a weighting factor for improved accuracy. Since each particle's position is already represented as a vector, its distance to the associated grid point can be quickly computed by taking the magnitude, allowing the system to weigh closer particles more strongly. If there are N particles in the unit cell, each represented as a vector v_i with data value d_i, then the resulting mapped value can then be computed as:

$$\sum_{i=1}^{N} w_i d_i \text{ where } w_i = \frac{1/|v_i|}{\sum_{i=1}^{N} 1/|v_i|}$$

Some applications of this transformation include the ability to color isosurfaces (represented in the Eulerian reference frame) using variables found in the Lagrangian particle data or to quickly analyze differences between particle and volume variables at common locations in the domain.

Once again, we can take the reverse of this approach and map Eulerian data values onto Lagrangian particles. Traditionally, this is done by searching for and then interpolating in between nearby grid points. Our joint data structure accelerates this procedure since, for each particle, the associated (nearest) grid point and all of its neighbors are already determined and can be quickly fetched from disk if not already in memory. While any number of interpolation schemes can be easily used with this design, we choose to interpolate using the associated grid point and its neighbors that lie in the same direction as the particle of interest relative to the unit cell center. In other words, if v is the vector pointing to the particle location and u_j is the vector pointing to a neighboring grid point, we only use that neighbor as a reference for interpolation if $v \cdot u_i > 0$. Overall, if d is the Eulerian data value of the associated grid point and there are M neighbors that meet the previous criteria with data value d_j, then the interpolated value can be computed as:

$$w_0 d + \sum_{j=1}^{M} w_j d_j \text{ where } w_j = \frac{1/|u_j - v|}{1/|v| + \sum_{j=1}^{M} 1/|u_j - v|}, u_0 = 0$$

Note that if no particle data is present and the Eulerian grid contains a flow field, one could apply this scheme to particle advection/pathline generation as long as the unit cell that each particle is associated with is continually updated as it travels through the domain. Furthermore, the *material*

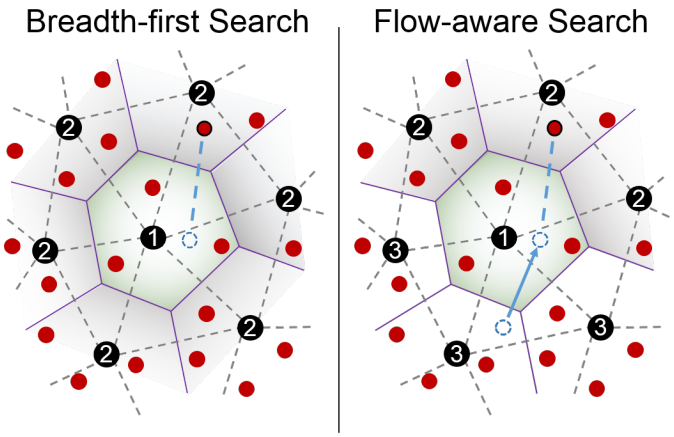

Fig. 5. Particles are drawn in red while grid points are drawn in black. Numbers indicate the search priority of nearby grid locations. Left) Use the location of the particle in the previous timestep (blue). We first search the grid location that contained the particle in the past. Since the particle is not found, the search continues to all neighbors. Right) We use the location of the particle in the previous two timesteps (blue). We can prioritize searching neighboring grid locations along the particle's direction of travel.

derivative could be used to map the evolution of Eulerian variables along the path of this particle.

3.3.3 Interpolating Inconsistent Temporal Resolutions

It is possible for a simulation to produce Eulerian and Lagrangian data at different temporal resolutions. When this occurs, some timesteps are missing one of the two representations. However, as we load multiple timesteps into memory, we can use interpolation techniques to fill in the missing components. In the case where particle data is dumped more frequently than grid data (the more common case), Eulerian variables for each unit cell are interpolated in the missing temporal regions. We treat each unit cell separately and use linear interpolation with a step size matching the number of additional particle timesteps. While we use a linear interpolation scheme in our current implementation, higher order interpolation is always an option as well.

A similar technique can be used when grid data is dumped more frequently than particle data (the less common case). Once again, the positions of the particles as well as any Lagrangian variables are treated as a trajectory and interpolated linearly in the missing temporal regions using a step size that matches the number of additional Eulerian timesteps. Special care must be taken in this circumstance because the particles can switch between different associated unit cells over time. It is crucial for the system to have knowledge over which unit cells contain which particles at every timestep, even those that are interpolated. As a result, we use intelligent local neighborhood searching when constructing trajectories from the particle data and can quickly interpolate and assign new particle positions to an associated unit cell if necessary. This is described in more detail in the next section. In the end, we can use these interpolation methods to perform joint analyses tasks at a temporal resolution matching the higher output frequency of the two representations.

3.3.4 Constructing Trajectories from Particles

While not entirely a joint operation that requires both data representations, constructing time-varying trajectories from the particle data is another important task. Our joint data structure can help to reduce the computation necessary to form these structures. Since scientific simulations run on a large-scale distributed system, raw particle data is nearly always dumped in an arbitrary ordering that differs across timesteps. This forces postprocessing systems to rely on a particle ID to match particles temporally into a coherent trajectory. Searching the full particle list for a particular ID can be very time consuming. One solution is to sort the particle data for every timestep so that a particle's ID is implied by its index. Not only is this an extremely expensive preprocessing task, it also does not have the advantage of ordering particles based on their location in the domain.

In our design, the particle data is already organized according to the Eulerian simulation grid. If we assume that the change in position of each particle is small between subsequent timesteps, then we can reduce the search time by looking first in a small neighborhood around the previous location of the particle. This is done by first searching the original unit cell for a particle with the same ID. If the particle cannot be located, the search continues to the neighboring unit cells. This continues in a breadth-first manner until the particle is found. An example of such a search can be seen in the left side of Figure 5. The unit cell (a hexagon in this case) which contained the previous location of the particle is searched first (as indicated with a "1"). Since the particle is not found, the search continues to all neighbors (as indicated with a "2"). In this case, the particle is found.

We can also take this one step further by considering information from the previous two timesteps. This allows us to narrow our search based on not only the prior position of the particle, but its direction of travel as well. If we make the assumption that the trajectory of a particle does not change very rapidly, we can prioritize our search in neighboring grid cells along the direction of travel. In other words, if we represent the past direction of travel of the particle and the location of neighboring unit cells as vectors, we can take the dot product and search cells that return a positive result first. Once again, if the new particle is not found, the search continues outwards in a breadth-first manner. An example of this "flow-aware" breadth-first search can be seen in the right side of Figure 5. The unit cell which contained the previous location of the particle is searched first (as indicated with a "1"). Since the particle is not found, the search continues to the three neighbors that lie along the direction of travel (as indicated with a "2"). In this case the particle is found. If the particle were still not found, the remaining neighbors will be searched next (as indicated with a "3"), followed by a repeat of this procedure for unit cells two neighboring steps away, etc.

As previously mentioned, when interpolating particles between two known timesteps, we can use these local neighborhoods to accelerate identifying which unit cells the interpolated points are associated with. More specifically, all the unit cells that were searched (either in the breadth-first or the flow-aware approach) when locating the particle in the next timestep become candidates for associating an

interpolation position with a unit cell. Once found (via a nearest neighbor search) the new interpolated particle as well as its interpolated Lagrangian variables can be stored in our unit cell based format and operated on as usual.

3.4 Multi-resolution Sampling

Scientists are also interested in exploring the entire domain. However, this is not always possible with large-scale datasets. As a result, we extend our design to extract multi-resolution representative subsets from the full dataset for users to explore. While generating a random sampling is always an available technique, our method opens up the use of more intelligent sampling schemes. We focus on large-scale particle data where occlusion can become a big problem and because randomly selecting particles can potentially result in an uneven sampling from different portions of the domain.

Since our joint data structure organizes particle data according to the Eulerian simulation grid, we can easily control the spatial sampling of particles. Choosing a constant percentage of all of the particles in each unit cell will ensure that we retain the relative particle densities from each part of the simulation domain. For example, we can load exactly half of the particles per unit cell. Adjusting this ratio will allow the system to generate multi-resolution subsets of the particle data. Since our joint data structure can treat unit cells independently from one another, the desired subset of data can be extracted directly from disk.

On the other hand, we can also purposefully sample more particles in interesting portions of the domain. Instead of sampling a constant ratio of particles from each unit cell, we can evenly sample the same number of particles. This will load high densities of particles where the Eulerian grid density is higher. In nearly all cases, unstructured simulation grids are specifically designed so that more grid points are placed in regions of interest (e.g., near an airfoil wing in a flow simulation) to give higher detail in those locations. For example, we can load exactly 5 particles from each unit cell resulting in more data retrieved from important parts of the domain. Choosing more or fewer particles per unit cell will generate this "biased" form of a multi-resolution sampling. Note that if a unit cell contains fewer than the desired number of particles then we simply load all available particles. Other techniques would need to use an intermediate mechanism to correlate the two frames before being able to generate such a sampling.

4 Results

Next, we apply these techniques to a number of real world large-scale datasets in the fields of fusion research, combustion, and cosmology. We choose these datasets not only because they contain both an Eulerian and Lagrangian representation, but because they come from entirely different domains and each exemplifies different ways our system can be used to operate with various joint datasets. An overview of the sizes of the datasets used for testing can be seen in Table 1. We chose both structured and unstructured grids with varying sizes to show that our design is effective in a variety of cases. Lastly, we provide performance results for each of the example tasks applied.

TABLE 1
Overview of the dataset sizes used for testing.

	Fusion	Combustion	Cosmology
Grid Type	unstructured	structured	unstructured
Num. Grid Points	3.5M	1.3B	7.9K
Num. Particles	40K	40M	2M

4.1 Fusion Dataset

The first dataset we use comes from XGC1, a large-scale fusion simulation developed by scientists at Princeton Plasma Physics Lab [1]. The simulation run used emulates the Alcator C-Mod fusion device located at the MIT Plasma Science and Fusion Center. Studying the highly-turbulent systems that arise from the magnetic confinement of plasma can help lead to the development of practical fusion energy. This particular dataset includes an unstructured Eulerian grid which represents the torus-shaped fusion device and includes a number of field measurements, such as electromagnetic field strengths and relative plasma densities. In addition, the simulation outputs a smaller subset of Lagrangian particles representing the physical ions and electrons that make up the plasma. Since the dataset has many more grid points than particles, each unit cell contains anywhere from 0 to 3 particles depending on its location in the domain.

Being able to correlate the macroscopic field measurements of the Eulerian data with the individualized properties of plasma particles becomes useful in studying this complex system. We can use the conditional queries provided by our method to quickly load particles according to specific Eulerian variables directly from disk. An example of this can be seen in Figure 6. In the Eulerian reference frame, we look at the density of the plasma which is noticeably higher in the core of the fusion device. In the Lagrangian reference frame we look at the Larmor radius of each particle, which represents the gyrokinetic motion of a magnetically confined object. We can choose to only load unit cells that fall within a desired range of plasma density directly from disk. As a result, any particles corresponding to the complex shapes represented by these unit cells are also loaded. We are then free to analyze this particular data subset (volume data, particle data, or both). For example, we can learn that the average Larmor radius of particles corresponding to the higher density plasma is significantly larger than those corresponding to the lower density regions.

Another method we can use to directly correlate the two reference frames is to map the Eulerian data values onto the physical particles themselves. We can use the mapping schemes provided by this method to quickly assign an Eulerian data value to each particle using its associated grid point and neighbors. Once both variables are represented in the same reference frame, correlations can then be investigated in greater detail. Using the same variables as the previous example, we can project the plasma density onto each individual particle and see how it correlates with the Larmor radius. As shown in Figure 7, we can see a clear correlation between these two variables now that they are both represented in the Lagrangian reference frame. Our system has the advantage of being able to make such a projection very quickly. In the case where the number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2016.2620975, IEEE Transactions on Visualization and Computer Graphics

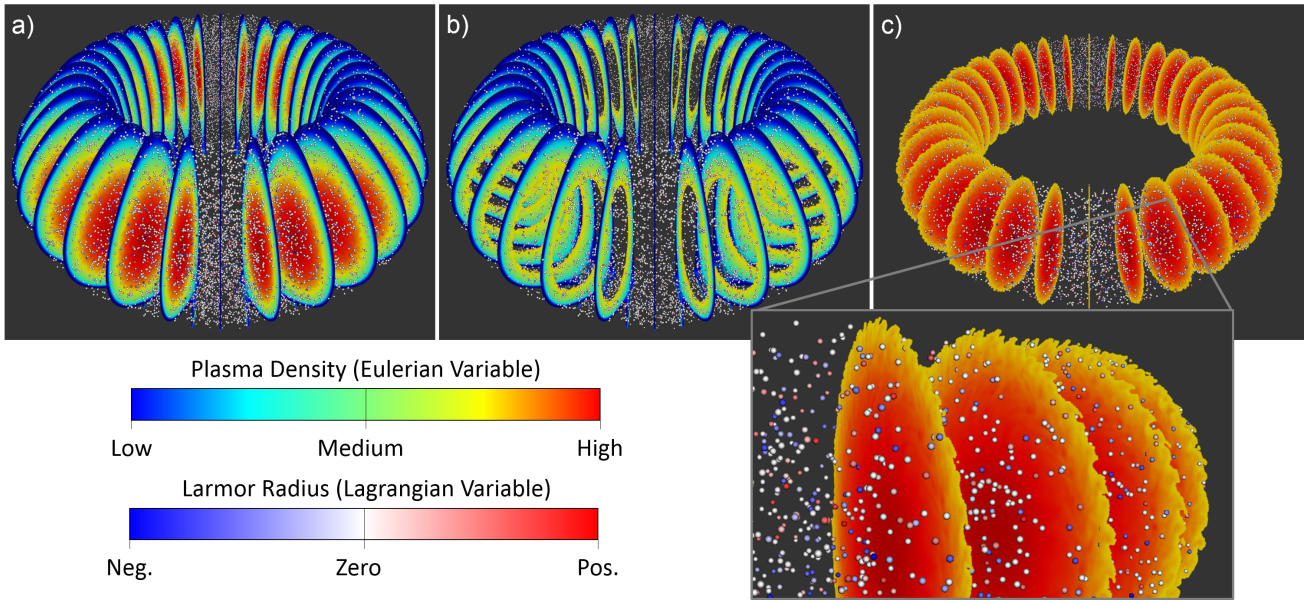

Fig. 6. a) No conditional query used. The Eulerian grid data is drawn on representative toroidal slices and colored according to the plasma density. The particles are drawn as circular sprites and colored according their Larmor radius. b) Conditionally querying unit cells that exhibit a low plasma density. c) Conditionally querying unit cells that exhibit a high plasma density.

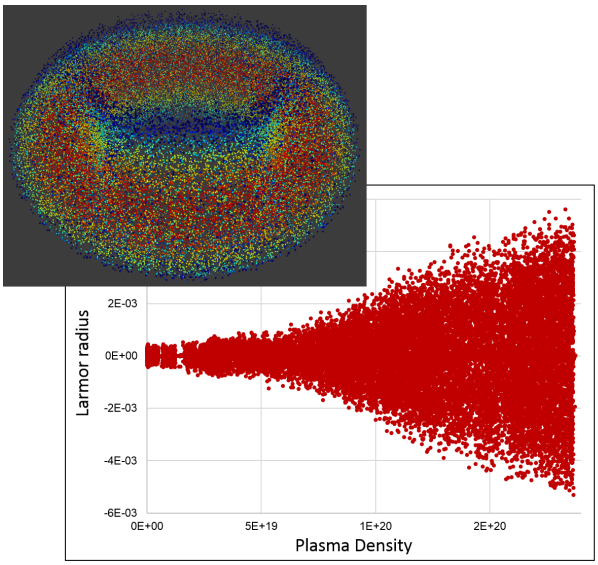

Fig. 7. Upper) Mapping the Eulerian plasma density variable onto each particle. Lower) A scatter plot showing the correlation between the Larmor radius and plasma density. Each point on the scatter plot represents an individual particle.

particles exceeds the number of grid points, such a projection would preferably be done in reverse with both variables represented in the Eulerian reference frame.

4.2 Combustion Dataset

The second dataset we use comes from S3D, a peta-scale combustion simulation developed by scientists at Sandia National Labs [2]. This particular simulation run depicts a 3D highly-turbulent lifted ethylene jet flame. Studying such systems can lead to a better understanding of processes like autoignition and becomes essential in developing next generation engines of high efficiency. This dataset includes a large structured Eulerian grid which records variables such as the mixture ratio of fuel and oxidizer. In addition, the simulation produces a number of massless tracer particles which measure the relative ratios of different chemicals present in the flame as well as other factors like temperature. We choose this combustion simulation to show the advantages of our method when dealing with structured (regular grid) volume data as well. In this case, we do not need to store grid locations or neighbor information since their position in the 3D domain is implied by their index. However, we still organize all Lagrangian information into the unit cell based representation as previously described. This large dataset exemplifies the need to perform queries of data subsets since each timestep can contain much more information than the amount of available memory.

Once again, correlating macroscopic field measurements with the individualized properties of the tracer particles can reveal interesting patterns. For example, we can identify which portions of the jet are simply mixing versus which are mixing and burning by correlating mixture ratio (Eulerian) with the temperature of the flame (Lagrangian). Figure 8 shows an example of querying unit cells that correspond to an equal mixture of fuel and oxidizer. This allows us to load and display particles corresponding to the complex structures formed when these two compounds mix. Coloring the particles according to their temperature distinguishes which portions of the jet are experiencing mixing and burning (red) from those that are just mixing (blue).

Next, we can study the movement of these tracer particles by constructing time-varying trajectories from the data. Once a set of particles corresponding to a particular Eulerian feature of interest is extracted, we can use either the simple breadth-first search method or the flow-aware search method to accelerate the time it takes to locate particles with the same ID's in subsequent timesteps. Throughout this

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2016.2620975, IEEE Transactions on Visualization and Computer Graphics

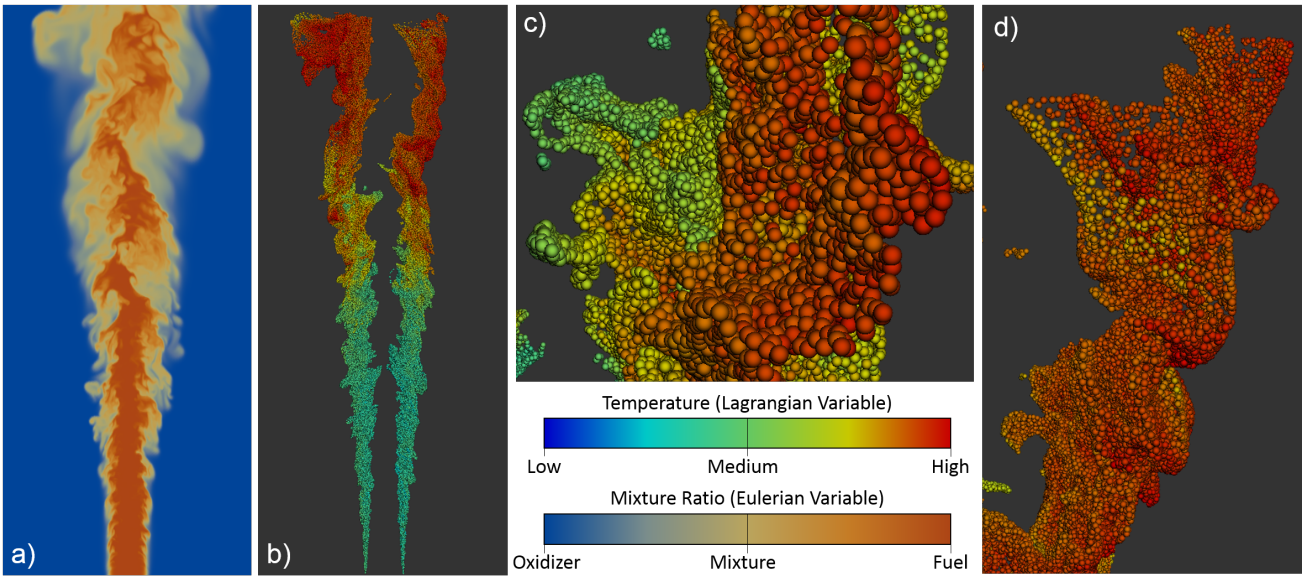

Fig. 8. a) A 2D slice showing the Eulerian mixture ratio variable. Fuel is injected from the bottom of the image. b) Conditionally querying unit cells corresponding to a specific mixture ratio between fuel and oxidizer. The high temperature regions show where burning is occuring. c/d) Zoomed views show the complex structures from which the particles are queried.

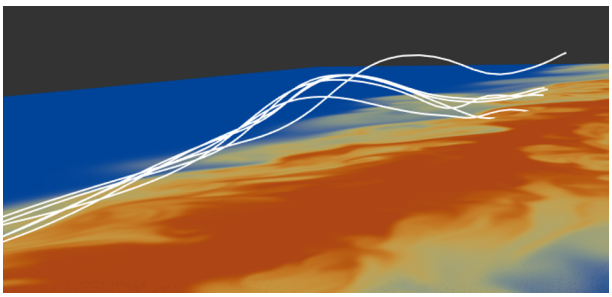

Fig. 9. An angled view of a set of Lagrangian trajectories drawn near a 2D Eulerian slice. This particular group of particles originated from a single Eulerian feature of interest.

process the particle data corresponding only to the selected group of particles is retrieved from disk. Figure 9 shows an example of such a result over the course of 800 timesteps and shows evolution of the original Eulerian feature used to query the particles. More detail comparing the performance of the two search methods is discussed in later sections.

4.3 Cosmology Dataset

The third example we use is a cosmology dataset that comes from the SLAC National Accelerator Laboratory's Dark Sky Simulation early data release [33]. Such simulations are essential in understanding the evolution of matter throughout the history of the universe. We choose this dataset to show how our design also applies to datasets that are not split into Eulerian and Lagrangian reference frames in the traditional sense. This large-scale N-body simulation computes sets of coherent structures called halos to represent groups of gravitationally bound bodies. These halos each have a 3D position associated with them as well as other derived variables. We treat these halos as the Eulerian grid locations when applying our method. In addition, this dataset is different from the previous two since the number of particles far exceeds the number of grid points.

We use this dataset to test the multi-resolution sampling schemes as the many particles can cause visual problems with occlusion. Furthermore, larger versions such a dataset can cause performance limitations and forces analysis tools to operate on representative data subsets. Each of the two sampling schemes discussed previously can be seen in Figure 10. In the first case, we use the unbiased sampling scheme to extract an even percentage of particles from every unit cell directly while loading the data from disk. This is done to maintain the relative particle densities in different parts of the domain. In the second case, we use the biased sampling scheme to extract a constant number of particles from each unit cell (e.g., exactly 50 particles from each unit cell). This will sample more particles from regions containing an increased number of unit cells (halos) and more effectively highlights densely packed clusters.

4.4 Performance Tests

We provide timing results for each of the preprocessing steps for each test dataset as well as timing results for the case studies shown in the previous sections. We compare our joint techniques against a full octree (uniform partitioning) and a k-d tree (non-uniform partitioning). For each test, all of the data initially started entirely on disk and we load only the necessary unit cells or tree leaves to generate the result. Presented times are the result of averaging several trials and measures were taken each time to clear the memory file cache of any residual data from prior test runs. Tests were done on a desktop computer using a standard 7200 rpm HDD and a 3.2 GHz Intel Core i7-3930K processor.

We choose the octree and k-d tree as a comparison because they are popular techniques that have been used successfully for particle and field data. These techniques can be arranged such that their storage space on disk is

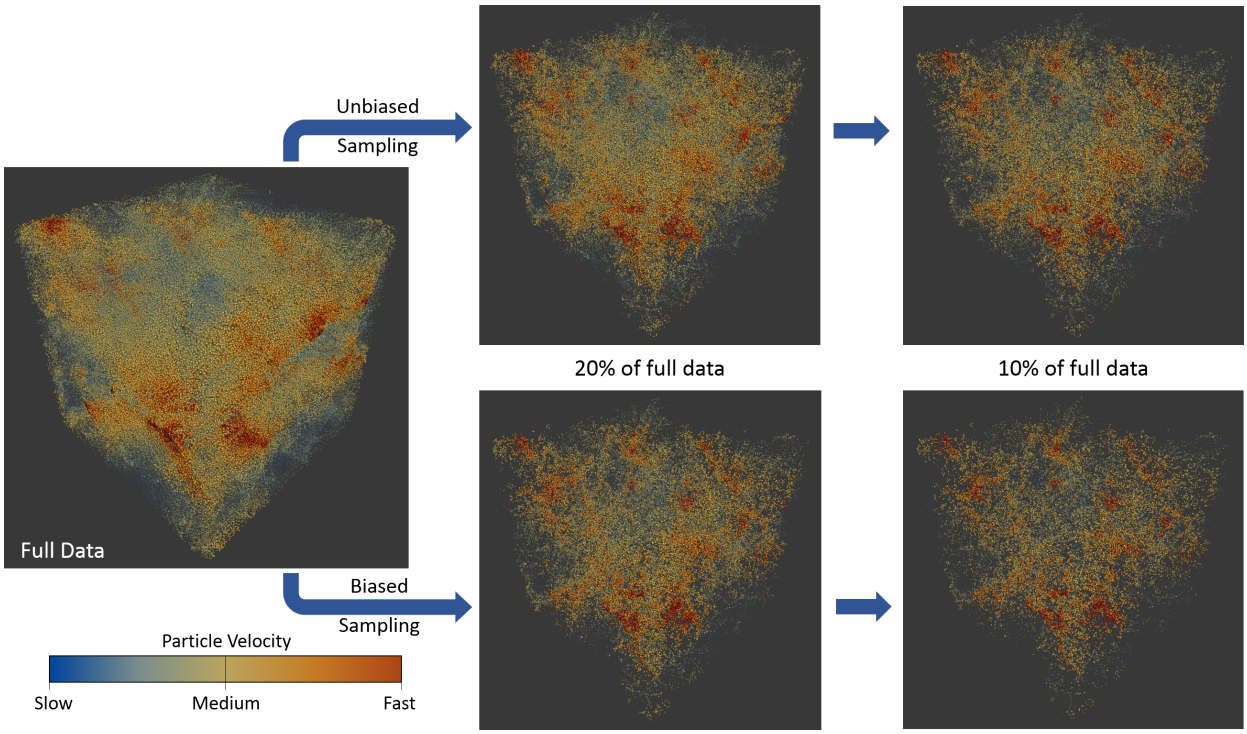

Fig. 10. Comparing two different multi-resolution sampling schemes. The full set of particles is shown on the left. The top shows the unbiased sampling scheme where the relative particle densities are maintained throughout the domain. The bottom shows the biased sampling scheme which samples more particles in regions that contain more grid points (halos).

very similar to that of our joint representation, giving a fair comparison. We choose a full octree (similar to regular grid indexing) and choose a depth such there is approximately one Eulerian grid point per leaf node. This full uniform partitioning removes the need to explicitly store the boundaries of each partition, making the storage overhead nearly identical to our joint representation. For the k-d tree, one standard method is to use the data entities themselves to mark the splitting planes throughout the domain alternating in a repeated xyz fashion. Since the boundaries are encoded into the data values themselves, this technique also does not require explicitly storing the boundaries of each partition making it also nearly identical to our joint representation.

While other, possibly more complex, techniques may have slight improvements over an octree or k-d tree (often at the expense of additional storage requirements) we feel that these are a good baseline to compare against in order to gain an estimate into the types of performance gains our method can achieve. In theory, one could always utilize more storage space to reduce queries and search times, but since most large-scale simulations are I/O bound, storage overheads must be kept to a minimum (as provided by the full octree, k-d tree, and our joint representation).

4.4.1 Preprocessing

The time it takes to compute each of the preprocessing steps to generate our joint representation (as described in Section 3.2.5) can be found in Table 2. The table also provides the preprocessing times to generate the octree and k-d tree as a comparison. These results represent the time it takes to preprocess one timestep and do not reflect file load or save times since all data must always be loaded into memory to perform the preprocessing for each of the techniques.

The results show that preprocessing our joint representation takes longer than preprocessing an octree or k-d tree as one would expect. This is because our representation already precomputes and encodes which particles are associated with each grid point, rather than just partitioning nearby entities into groups. The preprocessing time also depends on the number of grid points and particles. For example, the sorting/indexing step for the fusion case is very fast since this only needs to be done on the small number of particles. The combustion dataset does not require a nearest neighbor search since it lies on a regular grid, but takes the longest out of the three since it contains many entities than the other datasets.

The octree preprocessing time is on par with the spatial partitioning step since they perform very similar functions. The k-d tree however, takes longer than the octree since it must perform spatial sorting of the entities first to ensure a balanced split between the non-uniform partitions. While the joint representation incurs a longer preprocessing cost, this one time cost is overshadowed by the other advantages of the technique as well as the performance boost of numerous future tasks a scientist might perform when analyzing the data as described in the next section.

4.4.2 Case Studies

The performance tests comparing data structure in generating each of the case study examples are found in Table 3. The column on the right shows the speedup the joint representation provides over the k-d and octree respectively. Note that these results do include disk load times, since the amount of

TABLE 2
Preprocessing timing results.

Step	Fusion	Combustion	Cosmology
Spatial Partitioning	14.4 s	105.4 s	9.7 s
Nearest Neighbor Search	26.8 s	n/a	3.4 s
Sorting/Indexing	0.2 s	116.9 s	5.9 s
(Total)	41.4 s	222.3 s	19.0 s
Octree	15.2 s	116.4 s	10.2s
k-d Tree	21.3s	128.5 s	13.3s

data that needs to be loaded into memory varies depending on the case study and on the technique used. Overall, we can see that the octree and k-d tree perform similarly to one another since the data entities tend to be fairly uniformly distributed throughout their respective domains. The k-d tree performs slightly worse since its non-uniform nature incurs some extra computation in determining neighboring leaf nodes. Note that although the fusion dataset represents a torus shape, the data itself is stored in an "unwrapped torus" domain taking on a cylindrical shape (rather than a toroidal shape) and thus has very little empty space in a Cartesian bounding box.

TABLE 3
Timing results for the case studies.

Test	k-d	Octree	Joint	Speedup
Cond. Cell Query (fig. 6b)	14.9 s	14.5 s	3.7 s	4.02/3.92
Cond. Cell Query (fig. 6c)	13.7 s	13.4 s	3.1 s	4.42/4.32
Euler. to Lagr. map (fig. 7)	19.7 s	18.7 s	5.2 s	3.79/3.60
Cond. Cell Query (fig. 8b)	220.3 s	211.1 s	202.2 s	1.09/1.04
20% Unbias. Samp. (fig. 10)	0.6 s	0.6 s	0.6 s	1.00/1.00
10% Unbias. Samp. (fig. 10)	0.4 s	0.4 s	0.4 s	1.00/1.00
20% Bias. Samp. (fig. 10)	4.5 s	4.2 s	0.8 s	5.63/5.25
10% Bias. Samp. (fig. 10)	4.4 s	4.0 s	0.7 s	6.29/5.71

Beginning with the three fusion examples (Figures 6-7), it is evident that our joint structure significantly outperforms the other data structures. This is because although the octree and k-d based system have a significantly reduced search area, they must still perform the extra computation necessary to correlate particles to Eulerian grid points. This is necessary to determine which particles to keep in memory when performing a conditional query or to determine which grid points to use for interpolation. Furthermore, neighboring leaf nodes must be loaded into memory and searched as well because it is possible for a particle and its associated grid point to be found in different leaves of the tree. This computation has a large overhead in unstructured grids like the one used to represent this fusion simulation.

Furthermore, Figure 11 shows timing results of performing a typical conditional query in the fusion data as a function of the number of grid points that meet the condition (and as a result are loaded into memory along with their corresponding particles). We can see that since the joint data structure can operate on unit cells independently from one another it scales linearly with size. However, the octree and k-d tree must do additional computation associating particles with grid points and requires loading extra information into memory that may become eventually discarded (e.g., loading a particle only to discover later that it is associated with a grid point that does not meet the condition). Since leaf nodes cannot be treated independently from one another, the complexity of the task increases as we need to load larger subsets of the data into memory resulting in a non-linear scaling.

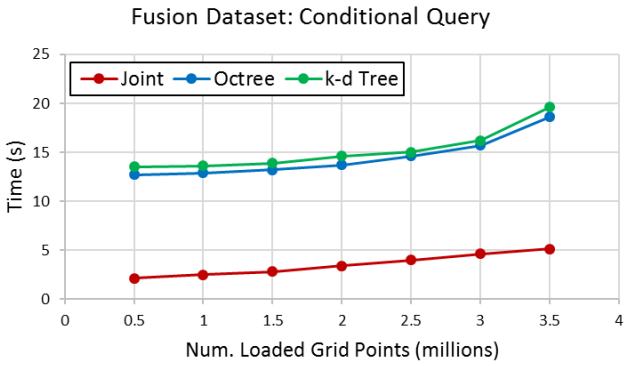

Fig. 11. A graph showing the timing results to generate the conditional queries in the fusion dataset examples as a function of the number of grid points that meet the condition.

Next, the timing results to load and generate the fusion conditional query (Figure 8) shows that there is not a significant improvement over the k-d tree or octree. This is because the extra computation (correlating particles to a grid location in the limited search area) becomes much simpler when working with a structured grid. In addition, we can compare the simple BFS trajectory construction method to the flow-aware BFS method as shown in Table 4. In all cases, the flow-aware method outperforms the simple one because we can save several disk reads by predicting which unit cells or leaf nodes likely contain our particles of interest.

TABLE 4
Timing results for trajectory generation (fig. 9).

Data Structure	Simple BFS	Flow-aware BFS	Speedup
k-d Tree	5.2 s	2.0 s	2.60
Octree	5.0 s	1.9 s	2.63
Joint	4.9 s	1.8 s	2.72

Lastly, we compare the timing results to load and generate the examples provided using the cosmology dataset (Figure 10). We can emulate the unbiased sampling in the k-d tree and octree by simply loading a percentage of the particles in each leaf node (rather than each unit cell). This would also retain the relative particle densities throughout the domain. Since the number of leaf nodes and unit cells is similar, the load times between the two techniques are also similar. However, in order to recreate the biased sampling scheme using the octree, particles must be associated with a particular grid point first. This results in a much higher computational cost and a longer load time.

Furthermore, Figure 12 shows timing results of this biased sampling as a function of the percentage of data loaded. The joint data structure once again scales linearly since unit cells can be treated independently. More particles must be loaded from disk when generating a larger biased

sampling. The timing results for the k-d tree and octree remain generally constant regardless of data subset size since nearly all particles must be loaded into memory and associated with a grid point before the sampling can be generated. Note that 100% biased sampling is a special case where all the data is loaded into memory and there is no longer a need to associate particle and grid positions making the timing results between the two techniques equivalent.

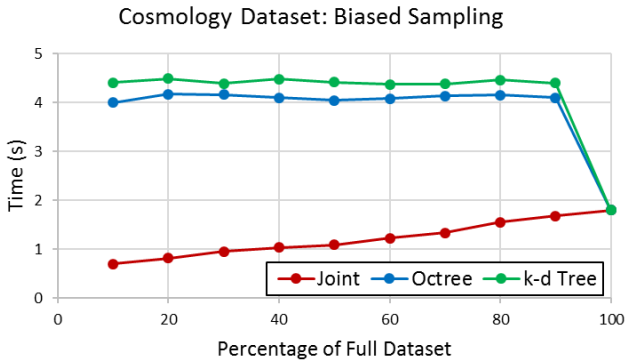

Fig. 12. A graph showing the timing results to generate the biased multi-resolution samplings in the cosmology dataset examples as a function of the percentage of data loaded.

Overall, our method performs either equal to or better than the hierarchical space partitionings for the joint particle/volume examples provided. It is clear that the major benefit of this design is for use in unstructured grids, where correlating particles with grid data becomes inherently difficult. As simulation sizes continue to grow, the use of unstructured grids will only increase since they can capture fine granularities in specific domain locations while sparsely sampling others.

5 Discussion

The above results demonstrate the effectiveness of this method when performing analysis tasks that require both the Eulerian and Lagrangian aspects of a particular dataset. The unit cell based format eliminates the need for our system to associate particles with a particular grid point making the retrieval of information from both reference frames fast and efficient. Moreover, the ability to operate on each unit cell individually makes this method applicable to large-scale datasets since we can quickly load desired data subsets into memory. There are also a number of ways this framework can be modified to broaden its applicability.

As an alternative to using the techniques from this paper in an out-of-core manner, we can analyze large-scale datasets by using a distributed system with a sufficient number of compute nodes. Using our design, groups of unit cells can be distributed ensuring that all necessary Eulerian and Lagrangian information is locally available.The system can then process unit cells in parallel, speeding up querying, sampling, and performing other joint operations on the data.

It is important to mention that this data structure is not designed to solve the popular "cell location" problem. In other words, the problem asks to find the associated unstructured grid point given an arbitrary location in the 3D domain. Instead, this method focuses on the quick retrieval and processing of both data representations by removing the need to perform the cell location problem for the particle data. However, other analysis and visualization techniques, such as volume rendering, need to solve the cell location problem in a more general sense in order to handle large-scale data. As a result, we can build schemes that solve this problem on top of this method if necessary. For example, one could use the hierarchical space partitioning to organize the unit cells as they are loaded into memory.

Another limitation comes from the vector based approach to represent Lagrangian particle locations. Since each particle is represented relative to its associated grid point, finding the global 3D location of a particle requires the grid's location to be in memory as well. This is generally not an issue since this data structure is designed specifically for operations that utilize both reference frames, and therefore, any associated grid information will be available in memory anyway. However, in the case where users want to only investigate the particle data, this design can be modified to store the 3D global location of the particle rather than as a vector. In this case, the particle's vector and its magnitude can be computed as needed in certain joint data operations.

Besides eliminating the preprocessing step by outputting simulation data directly into the unit cell based format, future work will also focus on developing joint representations and analyses in temporal neighborhoods. While this method tends to treat individual timesteps separately from one another, studying local temporal variations using both reference frames simultaneously can provide new perspectives into the data. New data operations and data structures are essential in transforming our unit cell format into one that efficiently represents an entire Eulerian and Lagrangian temporal neighborhood.

6 Conclusion

This work presents a new joint Eulerian-Lagrangian data representation geared towards enhancing analysis and visualization tasks that utilize both particle and volume data. By organizing particles according to simulation geometry, we can efficiently load and operate with both reference frames simultaneously. This is especially useful in large-scale settings where operating on representative data subsets or in an out-of-core manner becomes a necessity. Testing this method with real-world datasets shows its usefulness when performing different joint operations. Furthermore, performance tests on this method demonstrates its ability to quickly load and generate results, especially in cases with unstructured grids. This approach provides a groundwork for developing alternate techniques that focus on combining these different data modalities for enhanced analysis.

Acknowledgments

We would like to thank Princeton Plasma Physics Laboratory for providing the fusion dataset, Sandia National Laboratory for the combustion dataset, and the 2015 Scientific Visualization Contest for the cosmology dataset. This research has been sponsored in part by the U.S. Department of Energy via grants DE-SC0007443 and DE-SC0012610.

REFERENCES

[1] M. Adams, S.-H. Ku, P. Worley, E. D'Azevedo, J. Cummings, and C.-S. Chang, "Scaling to 150k cores: Recent algorithm and performance engineering developments enabling xgc1 to run at scale," *Journal of Physics: Conference Series.*, vol. 180, no. 1, 2009.

[2] C. S. Yoo, E. Richardson, R. Sankaran, and J. Chen, "A dns study on the stabilization mechanism of a turbulent lifted ethylene jet flame in highly-heated coflow," *Proceedings of the Combustion Institute*, vol. 33, no. 1, pp. 1619–1627, Oct. 2011.

[3] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, K. Matkovi, and H. Hauser, "On the way towards topology-based visualization of unsteady flow – the state of the art," in *EuroGraphics 2010 State of the Art Reports (STARs)*, 2010.

[4] A. Agranovsky, D. Camp, C. Garth, E. W. Bethel, K. I. Joy, and H. Childs, "Improved post hoc flow analysis via lagrangian representations," in *Proceedings of Large Data Analysis and Visualization Symposium*, Nov. 2014, pp. 67–75.

[5] P. Crossno and E. Angel, "Isosurface extraction using particle systems," in *Proceedings of Visualization '97*, Oct. 1997, pp. 495–498.

[6] B. Jobard, G. Erlebacher, and M. Hussaini, "Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 211–222, Jul. 2002.

[7] B. Jönsson, J. Salisbury, and A. Mahadevan, "Extending the use and interpretation of ocean satellite data using lagrangian modelling," *International Journal of Remote Sensing*, vol. 30, no. 13, pp. 3332–3341, Jul. 2009.

[8] S. Patkar, M. Aanjaneya, D. Karpman, and R. Fedkiw, "A hybrid lagrangian-eulerian formulation for bubble generation and dynamics," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2013, pp. 105–114.

[9] F. Sauer, H. Yu, and K.-L. Ma, "Trajectory-based flow feature tracking in joint particle/volume datasets," *IEEE Trans. on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2565–2574, Dec. 2014.

[10] F. Harlow, "The particle-in-cell method for numerical solution of problems in fluid dynamics," *Math Comp Phys*, vol. 3, pp. 319–343, 1964.

[11] J. N. Mpagazehe, "A physics-based, eulerian-lagrangian computational modeling framework to predict particle flow and tribological phenomena," Ph.D. dissertation, CMU, May 2013.

[12] J. Guilkey, T. Harman, A. Xia, B. Kashiwa, and P. McMurtry, "An eulerianlagrangian approach for large deformation fluidstructure interaction problems, part 1: Algorithm development." in *Fluid structure interaction II*. Cadiz: WIT Press.

[13] T. Harman, J. Guilkey, B. Kashiwa, J. Schmid, and M. P, "An eulerianlagrangian approach for large deformation fluidstructure interaction problems, part 2: Multi-physics simulations within a modern computational framework." in *Fluid structure interaction II*. Cadiz: WIT Press.

[14] Y. Zhang and A. Baptista, "Selfe: A semi-implicit eulerian-lagrangian finite-element model for cross-scale ocean circulation," *Ocean Modeling*, vol. 21, no. 3-4, pp. 71–96, 2008.

[15] Y. Fan, J. Litven, D. I. W. Levin, and D. K. Pai, "Eulerian-on-lagrangian simulation," *ACM Trans. on Graphics*, vol. 32, no. 3, pp. 22:1–22:9, Jul. 2013.

[16] J. Wilhelms and A. V. Gelder, "Octrees for faster isosurface generation," *ACM Trans. on Graph.*, vol. 11, no. 3, pp. 201–227, 1992.

[17] R. Xu, L. Kang, and H. Tian, "A g-octree based fast collision detection for large-scale particle systems," in *Proceedings of the International Conference on Computer Science and Electronics Engineering*, 2012, pp. 269–273.

[18] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann, "Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2009, pp. 15–22.

[19] S.-K. Ueng, C. Sikorski, and K.-L. Ma, "Out-of-core streamline visualization on large unstructured meshes," *IEEE Trans. on Visualization and Comp. Graph.*, vol. 3, no. 4, pp. 370–380, Oct. 1997.

[20] H. Yu, C. Wang, and K.-L. Ma, "Parallel hierarchical visualization of large time-varying 3d vector fields," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, Nov. 2007.

[21] C. Wächter and A. Keller, "Instant ray tracing: The bounding interval hierarchy," in *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, 2006, pp. 139–149.

[22] C. Garth and K. I. Joy, "Fast, memory-efficient cell location in unstructured grids for visualization," *IEEE Trans. on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1541–1550, Nov. 2010.

[23] N. Andrysco and X. Tricoche, "Matrix trees," in *Proceedings of the 12th Eurographics/IEEE-VGTC Conf. on Visualization*, Aug. 2010.

[24] D. Ellsworth, B. Green, and P. Moran, "Interactive terascale particle visualization," in *Proceedings of IEEE Visualization*, Oct. 2004, pp. 353–360.

[25] M. Treib, K. Bürger, F. Reichl, C. Meneveau, and A. S. abd Rüdiger Westermann, "Turbulence visualization at the terascale on desktop pcs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2169–2177, Dec. 2012.

[26] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann, "In-situ sampling of a large-scale particle simulation for interactive visualization and analysis," in *Proceedings of the 13th Eurographics/IEEE-VGTC Conference on Visualization*, 2011.

[27] T. Peterka, R. Ross, B. Nouanesengsy, T.-Y. Lee, H.-W. Shen, W. Kendall, and J. Huang, "A study of parallel particle tracing for steady-state and time-varying flow fields," in *Proceedings of Parallel and Distributed Processing Symposium*, May 2011.

[28] Y. Su, G. Agrawal, J. Woodring, K. Myers, J. Wendelberger, and J. Ahrens, "Taming massive distributed datasets: Data sampling using bitmap indices," in *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 13–24.

[29] Y. Su, G. Agrawal, J. Woodring, A. Biswas, and H.-W. Shen, "Supporting correlation analysis on scientific datasets in parallel and distributed settings," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14. New York, NY, USA: ACM, 2014, pp. 191–202.

[30] R. Fraedrich, J. Schneider, and R. Westermann, "Exploring the millennium run - scalable rendering of large-scale cosmological datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1251–1258, Nov. 2009.

[31] M. Hopf, M. Luttenberger, and T. Ertl, "Hierarchical splatting of scattered 4d data," *IEEE Computer Graphics and Applications*, vol. 24, no. 4, pp. 64–72, Jul. 2004.

[32] K. Bürger, P. Kondratieva, J. Krüger, and R. Westermann, "Importance-driven particle techniques for flow visualization," in *Proceedings of the IEEE Pacific Visualization Symposium*, Mar. 2008.

[33] S. W. Skillman, M. S. Warren, M. J. Turk, R. H. Wechsler, D. E. Holz, and P. M. Sutter, "Dark Sky Simulations: Early Data Release," *ArXiv e-prints*, vol. 407.2600, Jul. 2014.

Franz Sauer is a fourth-year graduate student at the University of California, Davis, studying computer science and scientific visualization under Kwan-Liu Ma. His research interests include data visualization, large-scale scientific simulations, computer graphics, and physics. Sauer received a BS in physics from the California Institute of Technology.

Jinrong Xie received his Ph.D. in computer science from University of California, Davis. His research interests mainly include scientific visualization, large scale parallel graphics rendering and data analytics. He was working with Professor Kwan-liu Ma in the VIDI research group. Before joining UCDavis, Xie was a master student in the College of Computer Science and Technology, Zhejiang University.

Kwan-Liu Ma is a professor of computer science and the chair of the Graduate Group in Computer Science (GGCS) at the University of California, Davis, where he leads the VIDi research group and directs the UC Davis Center for Visualization. His research interests include visualization, high-performance computing, and user interface design. Ma received a PhD in computer science from the University of Utah. He is an IEEE Fellow.