A Novel Information-Aware Octree for the Visualization of Large Scale Time-varying Data.

(Technical Report CS-TR-4778 and UMIACS-TR-2006-03)

Jusub Kim and Joseph JaJa

.

Institute for Advanced Computer Studies,
Department of Electrical and Computer Engineering,
University of Maryland, College Park, MD 20742,
E-mail: {jusub, joseph}@umiacs.umd.edu

Abstract

Large scale scientific simulations are increasingly generating very large data sets that present substantial challenges to current visualization systems. In this paper, we develop a new scalable and efficient scheme for the visual exploration of 4-D isosurfaces of time varying data by rendering the 3-D isosurfaces obtained through an arbitrary axis-parallel hyperplane cut. The new scheme is based on: (i) a new 4-D hierarchical indexing structure, called *Information-Aware Octree*; (ii) a controllable delayed fetching technique; and (iii) an optimized data layout. Together, these techniques enable efficient and scalable out-of-core visualization of large scale time varying data sets. We introduce an entropy-based dimension integration technique by which the relative resolutions of the spatial and temporal dimensions are established, and use this information to design a compact size 4-D hierarchical indexing structure. We also present scalable and efficient techniques for out-of-core rendering. Compared with previous algorithms for constructing 4-D isosurfaces, our scheme is substantially faster and requires much less memory. Compared to the Temporal Branch-On-Need octree (T-BON), which can only handle a subset of our queries, our indexing structure is an order of magnitude smaller and is at least as effective in dealing with the queries that the T-BON can handle. We have tested our scheme on two large time-varying data sets and obtained very good performance for a wide range of isosurface extraction queries using an order of magnitude smaller indexing structures than previous techniques. In particular, we can generate isosurfaces at intermediate time steps very quickly.

Index Terms

Time-Varying Data Visualization, Large Data Set Visualization, Iso-surface extraction

I. Introduction

As the speed of processors continues to improve according to Amdahl's law, researchers are performing large scale scientific simulations to study very complex phenomena at increasingly finer resolution scales. Such studies have resulted in the generation of datasets that are characterized by their very large sizes ranging from hundreds of gigabytes to tens of terabytes, multiple superposed scalar and vector fields, thereby generating an imperative need for new interactive visualization capabilities. Consider for example the fundamental mixing process of the Richtmyer-Meshkov instability in inertial confinement fusion and supernovae from the ASCI team at the Lawrence Livermore National Labs [1]. This dataset represents a simulation in which two gases, initially separated by a membrane, are pushed against a wire mesh. These are then perturbed with a superposition of long wavelength and short wavelength disturbances and a strong shock wave. The simulation produced about 2.1 terabytes of data, which shows the characteristic development of bubbles and spikes and their subsequent merger and break-up over 270 time steps. The resolution of each time step is $2,048 \times 2,048 \times 1,920$ or about 8 GB. Such high resolution simulations allow elucidation of fine scale physics; in particular, when compared with coarser resolution cases, the data allows observations of a possible transition from a coherent to a turbulent state with increasing Reynolds number. Current visualization systems and techniques are usually targeted for several orders of magnitude smaller datasets, and do not scale to the terabyte-sized datasets.

Isosurface rendering is an important visualization technique that enables the visual exploration of volumetric data using surfaces. Since the introduction of the Marching Cubes algorithm [2] that performs a complete scan of all the data cells, a variety of more efficient algorithms have appeared in the literature. These algorithms include a preprocessing step that constructs an indexing structure to speed up the identification of active cells (cells cut by the isosurface). Such techniques use either spatial data structures such as the octree [3], or the span space [4], or the interval tree [5], or the seed propagation technique [6]. The case when the data is too large to fit in main memory has recently received significant attention. There are two main research directions that have been pursued to deal with such large datasets. The first focuses on the development of out-of-core efficient implementations of internal memory algorithms (e.g. [7], [8]), while the second research direction makes use of parallel processing to achieve good performance on multiprocessor systems (e.g. [9], [10], [11]).

For time-varying data, a 4-D isosurface can be defined as the set of points that satisfy $f(x,y,z,t) = c$, for a given isovalue c. The problem of extracting linear approximations to 4-D isosurfaces from sampling over a structured grid was addressed by Weigle and Banks [12] and Bhaniramka et al. [13] but these techniques are computationally quite expensive and are only practical for very small datasets. Instead of dealing with the extraction of 4-D isosurfaces, most of the other work on time-varying data has focused on the problem of generating isosurfaces for each of the given time steps separately. For example, Chiang [14] develops an out-of-core version of the hierarchical temporal tree described in [15] by incorporating an out-of-core version of the interval tree indexing structure. Another example is the temporal branch-on-need (T-BON) octree described in [16] based on the branch-on-need octree structure [3].

In this paper, we address the problem of exploring 4-D isosurfaces of time varying data by rendering the 3-D isosurfaces obtained through an axis-parallel hyperplane cut. Such a cut is defined by a constraint of the form $x[y,\ z,$ or $t]$=α, for a user-specified α. Note in particular that a temporal cut $t = \alpha$ may fall in between two consecutive time steps in which case we will use interpolation on the fly to generate the corresponding isosurface. Our approach provides a rich environment that enables users to more effectively observe patterns and trends along the temporal dimension through the use of arbitrary spatial cuts $x[y,$or $z]$=α, and to render isosurfaces for any intermediate time step, without explicitly constructing the simplices needed for extracting the 4-D isosurface or decomposing hypercubes into simplices, as in [12], [13]. Carrying out such a plan on a large scale dataset requires a space efficient and scalable indexing structure that allows the fast retrieval of *active data blocks* from disk, that is, blocks that are necessary to build the 3-D isosurface corresponding to the parameters specified by the user, an *isovalue* and an *axis-parallel hyperplane*. None of the existing indexing structures can be easily extended to effectively solve such a problem, except possibly for the T-BON structure [16] but in this case we can only use temporal cuts along the given time steps. Moreover, The T-BON structure consumes a large amount of space that grows linearly with the number of time steps.

We present a new space and time efficient indexing structure called the *Information-Aware Octree* (IA-Octree) based on a new technique for assessing the relative variability of the data along the spatial and temporal dimensions. Our indexing structure can be viewed as a data-dependent version of the 4-D Octree (a generalization of the standard 3-D octree), where the data variability is captured through an entropy measure. The entropy is an information-theoretic notion that can be used to measure the information content of a sequence of values generated by a random event. Our experimental results show that this approach can effectively capture spatial and temporal coherence leading to a structure that is very compact and yet quite effective in identifying active blocks. In particular, our tests on two large time-varying data sets show that the IA-Octree is an order of magnitude smaller than the T-BON structure that cannot even handle all the queries that our structure can.

In addition to the indexing structure, our contributions include a *controllable delayed fetching* technique, and an optimized data layout scheme for the initial dataset. Our delayed fetching technique postpones data access from disk until a subtree of a certain size is fully traversed, after which the data is retrieved in an optimal order that exploits our particular data layout. Each such round of delayed fetching is followed by isosurface extraction and rendering on the resulting subspace, leading to scalable visualization that can be controlled by the delayed fetching parameter (the size of the subtree). Moreover, our preprocessing is very efficient and requires only a single sequential scan of the initial dataset.

The rest of this paper is organized as follows. We discuss major related out-of-core techniques in Section 2 and describe our new indexing structure and out-of-core techniques in Section 3. A summary of our experimental results is given in Section 4 and we conclude in Section 5.

II. Previous Out-of-Core Techniques

Due to their electromechanical components, disks have two to three orders of magnitude longer access time than random access main memory. A single disk access reads or writes a block of contiguous data at once. The performance of an out-of-core algorithm is often dominated by the number of I/O operations, each involving the reading or writing of disk blocks. Hence designing an efficient out-of-core visualization algorithm requires a careful attention to data layout and the organization of disk accesses in such a way that active data blocks are moved in large contiguous chunks to main memory. During the past few years, a number of out-of-core techniques have appeared in the literature to handle several visualization problems. For example, out-of-core isosurface extraction algorithms for static datasets are reported in [7], [8], [9], [11]. Of more interest to us is the previous work on out-of-core algorithms dealing with time-varying data. For example, Chiang [14] proposes an out-of-core isosurface extraction algorithm based on a time hierarchy to store the metacells whose field values are close enough to each other for the corresponding time interval. This hierarchy uses the Binary-Blocked-I/O interval trees (BBIO) [8] as secondary structures to support I/O optimal interval searches. Note that no indexing structure based on interval trees or span space seems to be suitable to deal with the problem studied here as one of our constraints involves a spatial cut.

However the work reported by Sutton and Hansen [16], which introduced the *Temporal Branch-on-Need-octree (T-BON)* to extract isosurfaces for each time step separately, can be generalized to solve our problem for *cuts along the grid lines*. In particular, their strategy is to build a Branch-On-Need-Octree (BONO) [3] for each time step, storing general common infrastructure of the trees (i.e., branching factors and pointers to children) in a single file. Unfortunately, the T-BON cannot handle temporal cuts not among the given time steps, and its size increases linearly with the number of time steps resulting in a large data structure that does not exploit any type of possible temporal coherence of the data. Another related work is the PHOT data structure developed in [17]. While such a data structure achieves asymptotically optimal internal memory search, its size is substantially larger than our data structure and its out-of-core performance is comparable or not as good as ours depending on the query type.

We should note that octrees have been extensively used for direct volume rendering of time varying data. For example, Shen et al. [18] propose the Time-Space-Partitioning (TSP) Tree for direct volume rendering. They build an octree and each node of the octree is a binary time tree in which each node contains the mean and variance value across the time span that the node represents. The value contained in each node of the time tree is used for fast volume rendering trading off image quality. In spite of their use of the octree structure, none of the out-of-core techniques used for direct volume rendering of time-varying data seem to be applicable to the problem studied here.

We end this section by distinguishing our work and that involving the direct computation of the 4-D time-varying isosurfaces, which is more general than the problem addressed in this paper. Weigle and Banks [12] describe a recursive contour meshing strategy for n-dimensional grids, involving the decomposition of the hypercubes into n-simplices, followed by contouring these simplices into (n-1)-simplices to satisfy a given constraint, and the process is repeated for additional constraints . This strategy is very computationally demanding and is not feasible for any data of large size. Bhaniramka et.al [13] present an algorithm for constructing isosurfaces in any dimension. Their algorithm constructs the isosurface within a hypercube by finding the convex hull of an appropriate set of points and retaining the portion of its boundary which lies inside the hypercube. Although their algorithm leads to some useful applications, the approach of extracting isosurfaces in four-dimensional space and taking only a slice defined by one or more constraints is computationally prohibitive for large scale data. The reported experimental results in both papers are for very small data sizes and the execution times mentioned are substantially slower than ours, but on the other hand they address a more general problem.

III. Information-Aware Octree

The IA-Octree is essentially a space-driven 4-dimensional indexing structure based on a new technique that enables a good estimate of the relative resolutions of the spatial and temporal dimensions. We describe in this section the structure of the IA-Octree, the corresponding data layout, and how the indexing structure is used to effectively handle our query types. In what follows, we will refer to the 4-D subvolume represented by a leaf node as a *hypercell*, which consists of a time series of *3-D subcubes*, each subcube belongs to a single time step and will be assumed to be of size equal to a disk page.

A. Dimension Integration

We present an entropy-based dimension integration technique. Entropy [19] is a numerical measure of the uncertainty of the outcome for an event x, given by $H(x) = -\sum_{i=1}^{n} p_i \log_2 p_i$, where x is a random variable, n is the number of possible states of x, and p_i is the probability of x being in state i. This measure indicates how much information is contained in observing x. The more the variability of x, the more unpredictable x is, and the higher the entropy. For example, consider the series of scalar field values for a voxel v over the time dimension. The temporal entropy of v indicates the degree of variability in the series. Therefore, high entropy implies high information content, and thus more resources are required to store or communicate the series. Note that the entropy is maximized when all the probabilities p_i are equal.

We use the entropy notion to determine the relative sizes of the dimensions of the hypercell (that is, a leaf in our octree). Higher entropy of a dimension relative to the other dimensions implies that this dimension needs to be split at finer scales than the other dimensions. For example, if a temporal entropy is twice as much as the spatial entropy, we design the hypercell to be of size $s \times s \times s \times \frac{s}{2}$ ($x \times y \times z \times t$), where s is the size of the spatial dimension of the hypercell.

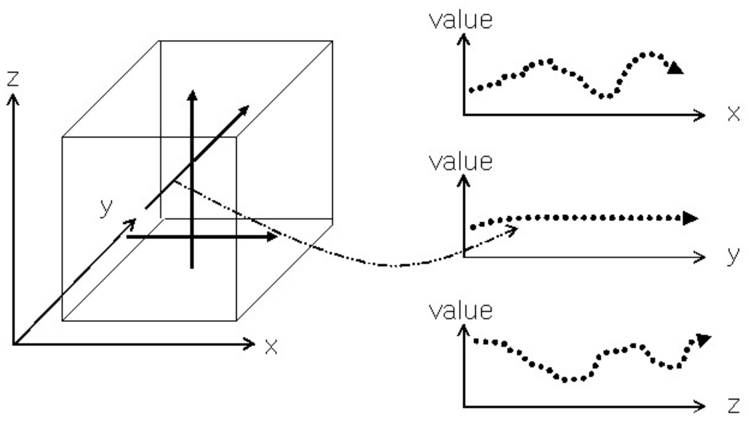


Fig. 1. Entropy estimation in each dimension. Note that the y dimension has almost zero entropy in this example.

Figures 1 and 2 show how this entropy-based dimension integration leads to an indexing structure for the 3-D case. Figure 1 shows an extreme case in which the values along the y dimension remain almost constant over all possible (x, z) values (that is, the entropy of y is almost zero) while each of the x and z dimensions has some degree of variability. The hypercell size and the corresponding hierarchical indexing

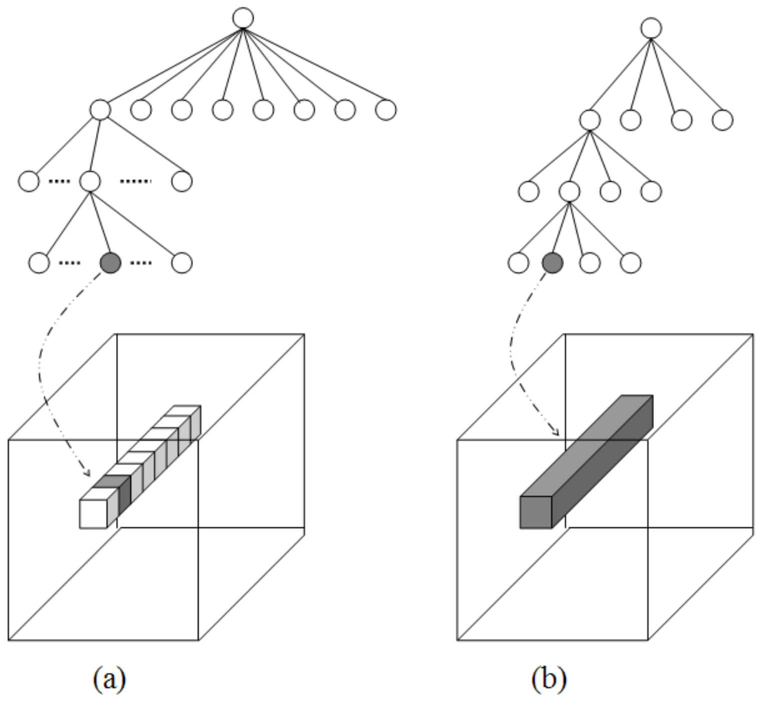


Fig. 2. Different hypercell sizes and corresponding hierarchical indexing structures for the data of Figure 1: (a) standard hypercell; (b) information-aware hypercell.

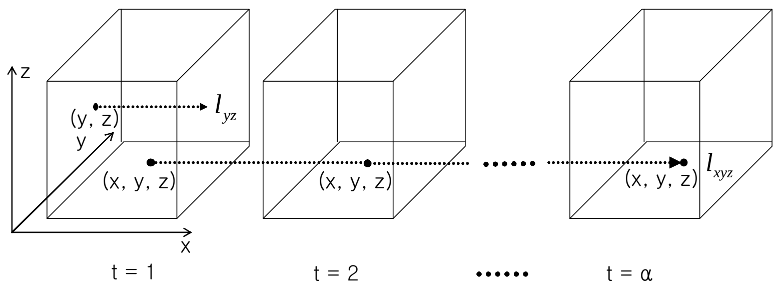


Fig. 3. A sampled subvolume along a series of time steps. Entropy computation of x dimension is performed along the line l_{yz} and averaged over all (y,z) values, while temporal entropy is computed along l_{xyz} and averaged over all (x,y,z) values.

structure will be designed as shown in Figure 2 (b), that is, it has a quadtree structure unlike the standard octree of Figure 2 (a) in which the hypercell has the same size in each dimension.

To estimate the entropy, we select a set of time steps, referred to as *reference time steps*, and compute spatial entropies for the corresponding 3-D volumes. These time steps are selected uniformly from the overall time series (or can be adaptively sampled at a higher rate in the time domain of high temporal entropy for more correct estimation). For each corresponding 3-D volume, we compute spatial entropies for a random subset of subvolumes. Consider for example one sampled subvolume shown in Figure 3. Assuming the scalar field values $v \in [1, 2, \ldots, n]$, the entropy E_x is defined as follows (direction l_{yz} is parallel to the x-axis and is anchored at the point (y, z)) :

$$P_v^{yz} = \frac{\sharp \text{ of occurrences of the scalar field value } v}{\text{Total } \sharp \text{ of voxels on the direction } l_{yz}} \tag{1}$$

$$E_x^{yz} = -\sum_{v=1}^{n} P_v^{yz} \log_2 P_v^{yz} \tag{2}$$

$$E_x = \frac{\sum_{y,z} E_x^{y,z}}{\sum_{y,z}} \tag{3}$$

where $\sum_{y,z}$ is over the grid points in the (y,z) plane.

On the other hand, the entropy of the temporal dimension E_t is computed by taking values at the same voxel along the time-axis as follows.

$$P_v^{xyz} = \frac{\sharp \text{ of occurrences of the scalar field value } v}{\text{Total } \sharp \text{ of voxels along the temporal direction } l_{xyz}} \tag{4}$$

$$E_t^{xyz} = -\sum_{v=1}^{n} P_v^{xyz} \log_2 P_v^{xyz} \tag{5}$$

$$E_t = \frac{\sum_{x,y,z} E_t^{x,y,z}}{\sum_{x,y,z}} \tag{6}$$

where $\sum_{x,y,z}$ is the number of voxels in the subvolume at a time step.

If the number of the possible scalar field values is large (as for example would be the case for floating point scalar field values), we first *quantize* the original values into n values using a non-uniform quantizer such as the lloyd-max quantizer [20]. Note that this quantization is only used for the purpose of computing the entropies.

Since we are primarily concerned about establishing the relationship between the spatial and temporal dimensions, we compute the *spatio-temporal entropy ratio* defined as the ratio of the average spatial entropy to the temporal entropy. We compute the spatio-temporal entropy ratio in each of the sampled subvolumes and take their *harmonic mean*, which is then used for building our IA-octree.

We note that in general a time series of data volumes will consist of a number of temporal domains during which the spatio-temporal entropy ratio can be different. Our general strategy is to decompose the time series into a set of temporal regions, each of which will be characterized by its spatio-temporal entropy ratio. Hence we will build a separate IA-Octree for each temporal region, capturing the data volumes within each temporal region separately. This strategy will be illustrated in section 4 when we describe our experimental results.

B. Indexing Structure and disk layout

The starting point of our IA-Octree is the 4-D octree structure that usually divides the 4-dimensional space into 2^4 subspaces. We make use of the spatio-temporal entropy ratio to determine the branching factor and the size of the 4-D volumes at the leaves, and follow the branch-on-need strategy [3] as well. We delay the branching until it is absolutely necessary as in [3]; however the temporal branching is further delayed by a factor of the spatio-temporal entropy ratio if the ratio is more than 1 or expedited by that factor if it is less than 1. Moreover, our algorithm ensures that the lower subdivision in each branching dimension always covers the largest possible $2^k \times l_d$, where l_d is the size of the hypercell edge in the d-dimension (instead of exact power of two).

Each tree node contains the minimum and maximum values of the scalar fields in the region represented by the node, a pointer to the first child, and a branching factor. The size of the tree is reduced by pruning nodes in which the minimum and maximum values are the same because they do not contribute to isosurface extraction.

Before describing our data layout on disk, let's make the assumption that our highest priority is to get the best possible performance for cuts along the temporal dimension, followed by cuts along the z axis, followed by y and finally x. The layout can be easily adapted to any other performance goals. Under this assumption, the data layout is organized as follows. For each time step, we create a file that contains all the 3-D subcubes of the data volume of that step, organized in a lexicographic ordering using the left most coordinates (x,y,z) of each subcube. Note that each subcube is of size equal to a disk page (or disk

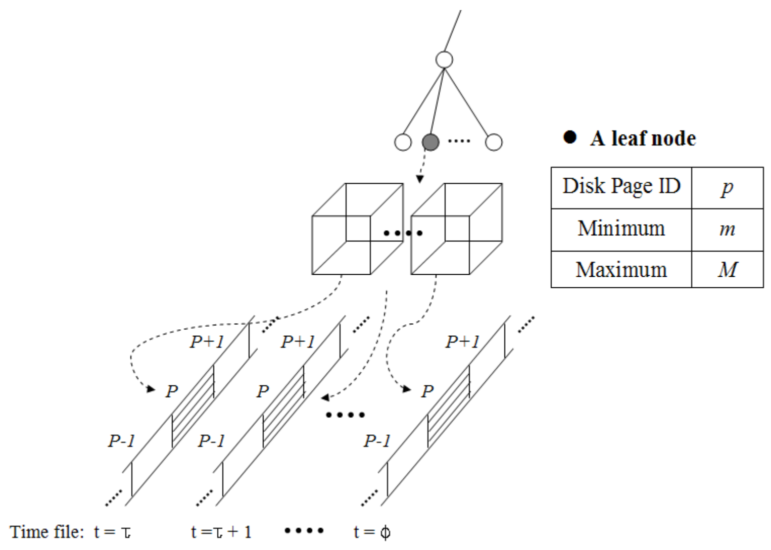


Fig. 4. Data layout on disk. Scalar field values within a hypercell are stored across time files but the disk page IDs are all the same in the corresponding time files. Note that a leaf node contains a disk page ID, minimum and maximum values of the scalar field over the hypercell but the range of each dimension is computed on the fly.

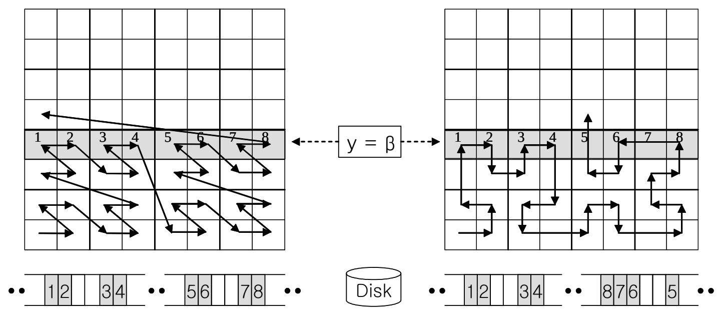
block). The scalar field values within a spatial subcube are also organized in a lexicographic order within the disk page. Now, consider a 4-D hypercell defined at one of the leaves of our octree. This hypercell consists of a time series of 3-D spatial subcubes, and hence they each appear in a different time file. However note that they all have the same offset within their files, and hence we only need this offset to identify these subcubes spread among different time files (see Figure 4). The temporal range as well as the spatial ranges of a hypercell can be trivially computed on the fly during query processing, and hence there is no need to store these ranges at the nodes. We use a simple buffer management system which manages all the disk pages into and out of the disk.

The use of the lexicographical order instead of the more prevalent z-order or Hilbert-Peano order [21] is due to our query types, which include a geometric constraint $x[y$, or $z]=\alpha$. While the z-order and the Hilbert-Peano order may be better choices for regional queries, the lexicographical order results in better performance on average for our hyperplane query types. For illustration purposes, Figure 5 shows how three different layouts affect the disk I/O efficiency for a hyperplane query in the 2-D case. Disk I/O efficiency can be expressed by how many contiguous disk pages are accessed for a given query. The lexicographical order achieves much higher contiguity in disk accesses required by the hyperplane query.

C. Tree Traversal and Controllable Delayed Fetching

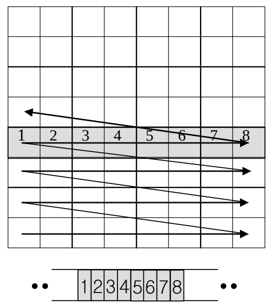
To optimize disk accesses and make rendering scalable, we organize the access of active data blocks in large contiguous chunks, each of which corresponds to a subspace of the original data as shown in Figure 6 for the 3-D case. We essentially delay the retrieval of a hypercell until a subtree whose root is at a preset level is completely traversed. We describe the details for an x, y, or z cut since the temporal cut is much simpler (given our data layout). We traverse the IA-Octree in depth-first order by checking whether the node's minimum and maximum values span the isovalue and the user-specified hyperplane intersects the node. Once we reach the active hypercells at the lowest level, we insert each hypercell into a *priority queue* using the priority key (t,z,y,x) in lexicographic ordering. More specifically, for an active hypercell in which the x,y, and z coordinates of the left-most corners are (α,β,γ) and the time dimension spans $(t,\ t+\theta)$ with disk page ID p (i.e., offset within the time file), we insert $\theta+1$ entries, each of which consists of $(t,\gamma,\beta,\alpha,p)$, $(t+1,\gamma,\beta,\alpha,p)$,..., and $(t+\theta,\gamma,\beta,\alpha,p)$.

The actual data fetching of the active hypercells is delayed until the depth-first order tree traversal revisits the nodes at a preset level. For example, disk accesses for the active hypercells in the subvolume



(a) z-order

(b) hilbert-peano order



(c) lexicographical order

Query \ Order	z	hilbert-peano	lexicographical
x= α	1	2	1
y= β	2	2	Column size
Avg.	1.5	2	~ Column size / 2

(d) The number of contiguous disk block accesses

Fig. 5. Disk access patterns for a hyperplane query in a 2-D case in three different disk layouts. Grey blocks correspond to the ones satisfying the hyperplane query y=β.

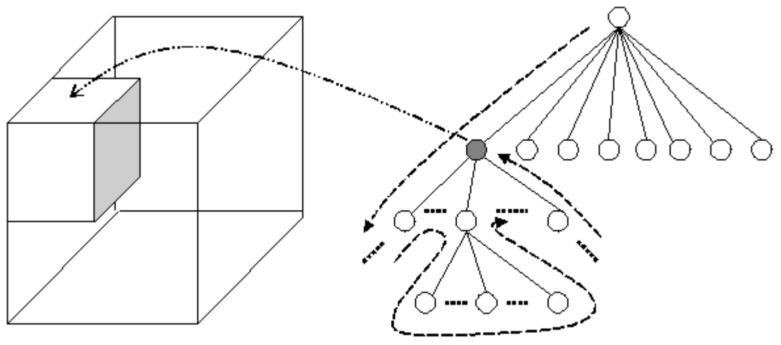


Fig. 6. Controllable delayed data fetching. Disk accesses for the active hypercells in the subvolume corresponding to the grey node are delayed until the traversal revisits the grey node.

in Figure 6 are delayed until the tree traversal revisits the grey node and then actual data fetching starts by popping the top entry of the priority-queue and issuing the corresponding disk access. This process continues until the priority-queue is empty after which the tree traversal proceeds in depth-first order. Note that the example shows a 3-D case for illustration purposes.

The controllable delayed fetching enables us to optimize the disk accesses and is also quite effective in enabling scalable rendering which we explain in the next section. We adjust the preset level at which delayed fetching is carried out depending on the sizes of the data and main memory, as well as rendering speed. Setting the prefetching level is a trade-off between efficient access (the higher the level the more efficient the disk access is) and scalability.

D. Isosurface Extraction and Rendering

Each time we fetch the data at a preset level, isosurface extraction for the corresponding subspace is performed. Scalar field values corresponding to all the active hypercells are loaded into a main memory.

Consider a spatial-cut query types specified by an isovalue and a hyperplane $x[y \text{ or } z] = \alpha$. If α is along a grid line, we extract the corresponding slice from each of the loaded subcubes, and organize all the slices together to constitute a 3-D volume. For example, for the $x = \alpha$ hyperplane query, we create a 3-D volume whose dimensions are equal to the subvolume's y, z, and t dimensions. Then we use the standard marching cubes algorithm [2] to extract triangles in the 3-D volume. However the volume is selectively traversed because we know which portions of the volume are filled with the data from the active hypercells. This partial assembly of slices and triangle extraction performed after every delayed fetching enables us to avoid forming a large 3-D volume in main memory for triangle extraction in the spatial-cut query. If α is not along a grid line, we extract the two consecutive slices containing α, and perform a linear interpolation pointwise, to generate the slice corresponding to α. The process is now completed as before. For temporal-cut queries, triangle extraction is straightforward for cuts along one of the given time steps since each loaded disk page corresponds to a 3-D subcube that satisfies the query. Otherwise, we load the two subcubes corresponding to consecutive time steps containing α, and interpolate to generate the appropriate subcube. The interpolated subcube is stored in the buffer management system to avoid re-interpolation when the subsequent queries are different only in isovalues.

Isosurface rendering is incrementally performed by rendering the extracted triangles whenever the number of triangles reaches some preset threshold value, concurrently with tree traversal, data movement, and triangle extraction.

E. Preprocessing

The preprocessing required to generate the data layout and extract information for the IA-Octree is simple. Let's make the assumption that the original volume data is stored in some fixed ordering (say along x, y, and then z) for each time step separately, and that the data volume corresponding to each time step may not fit in main memory. Our preprocessing involves a sequential scan of the input data, loading for each time step as large a subcube from the corresponding volume as possible. Once a large subcube is loaded, we traverse the indexing structure to determine the leaf nodes included in the large subcube. The corresponding leaf nodes are first sorted in lexicographical order followed by determining the minimum and maximum scalar field values at each selected leaf node. This pair of values are then inserted at the leaf node. Afterwards, the set of scalar field values corresponding to the leaf node are written back into a disk page. Since we process the selected leaf nodes in lexicographical order, the resulting order of disk pages follows the lexicographical order as well. Since the data set at a time step may not fit in main memory, we continue to read the original data by the subcubes and repeat the above process. Once the extreme scalar values in all the leaf nodes are determined, the values are propagated up through the tree. It is clear that our preprocessing involves a sequential reading and writing of the input data, and hence makes optimal use of the bulk data transfer of disks.

IV. Performance

To evaluate the performance of our scheme, we consider two large time-varying datasets: the Richtmyer-Meshkov dataset for time steps $100-139$, each downsampled by two along each spatial dimension, and the Five Jets dataset [22] consisting of 2000 time steps. Each time step of the Richtmyer-Meshkov dataset involves a $1024 \times 1024 \times 960$ grid with one-byte scalar fields and hence is of size 1GB, resulting in 40GB data set. The Five Jets dataset consists of $128 \times 128 \times 128$ grid with 4-bytes floating point values resulting in a total of 16GB.

We ran the tests on a single node of the University of Maryland Visualization Cluster, in which each node consists of two 3.0 GHz Xeon EM64 processors, 8GB main memory, 60GB local disk, and a NVIDIA6800 GPU with bi-directional 4Gbps data transfer rate to memory via PCI-Express bus. In all our experiments, we made use of only one of the two processors on the node.

Using the entropy measure, we obtained a spatio-temporal entropy ratio equal to 1.5 over the time steps $100-139$ for the Richtmyer-Meshkov dataset. We chose the size of a hypercell to be $16 \times 16 \times 16 \times 24$

making the temporal length of a hypercell 1.5 times the length of the spatial dimension. On the other hand, we divided the temporal domain of the Five Jets dataset into four time regions (see Figure 7) having respectively the spatio-temporal entropy ratios of 0.5, 1, 3, and 4. We built a separate IA-Octree on each time region. Each hypercell size was chosen to be $8 \times 8 \times 8 \times$ $[4, 8, 24\ \textit{or}\ 32]$ depending on the corresponding spatio-temporal entropy ratios.

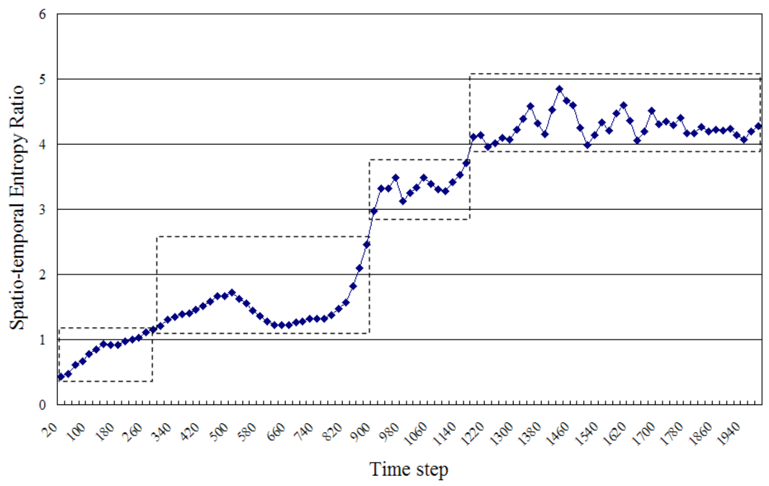


Fig. 7. Spatio-temporal entropy ratios computed at uniformly selected 100 reference time steps among the 2000 time steps in the Five Jets data. Each dashed box corresponds to a time region.

Our preprocessing time for constructing indexing structure and reorganizing data on disk was less than 60 minutes for the entire 40GB Richtmyer-Meshkov dataset and less than 20 minutes for the 16GB Five Jets dataset. This performance is quite good given the fact that it takes around 30 minutes and 10 minutes respectively just to sequentially read and write back each of the two data sets at peak I/O transfer rates.

We compare the performance between our IA-Octree and the T-BON on the specific queries that the T-BON was designed for. The sizes of the resulting structures for the two data sets are given in Table I.

	T-BON	IA-OCTREE
The Richtmyer-Meshkov dataset	25MB	2MB
The Five Jets dataset	74MB	9MB

TABLE I

COMPARISON OF INDEXING STRUCTURE SIZES.

As can be seen from the above table, our indexing structure is an order of magnitude smaller than the T-BON structure.

Tables II and III compare the performance of the T-BON and IA-Octree using multiple measures on the two datasets for the specific query types which can be handled by the T-BON. Besides the results shown in the tables, we have also achieved consistent results for other spatial-cut queries. Each spatial and temporal-cut query result was obtained over 10 representative isovalues (40, 60, ..., 220 for the Richtmyer-Meshkov and 251000, 251500, ..., 255500 for the Five Jets). The result for the temporal-cut query was obtained over 8 and 100 uniformly selected time steps from the Richtmyer-Meshkov and the Five Jets dataset respectively. As seen in the tables, the IA-Octree has achieved comparable (and sometimes better) performance using an order of magnitude smaller indexing structure size. Note that the results from both indexing structures are based on our particular disk layout.

On the other hand, our IA-Octree can also handle temporal cut queries at intermediate time step values with an overhead consisting of loading two time slices of a hypercell followed by linear interpolation to generate a hypercell slice for the intermediate value. Table IV shows the corresponding IA-Octree

Spatial-Cut Query at Z		T-BON			IA-OCTREE		
		MIN	AVG	MAX	MIN	AVG	MAX
400	Loaded Data	488	617	710	545	667	738
	Effectiveness	6.72	10.4	16.8	6.04	9.60	16.2
	Disk Access Time	1.27	2.75	13.7	1.52	3.00	14.2
	Tree Traversal	2.37	2.92	3.40	1.19	1.46	1.70
	Total Time	3.64	5.67	17.1	2.71	4.46	15.9
450	Loaded Data	465	671	791	533	723	818
	Effectiveness	7.52	13.2	20.4	6.60	12.3	19.8
	Disk Access Time	1.10	2.85	14.5	1.40	3.07	14.8
	Tree Traversal	2.25	3.14	3.75	1.16	1.59	1.84
	Total Time	3.35	5.99	18.2	2.56	4.66	16.6
500	Loaded Data	397	547	612	475	600	650
	Effectiveness	7.80	14.5	22.8	6.54	14.0	21.6
	Disk Access Time	0.938	2.63	15.0	1.28	2.95	15.3
	Tree Traversal	1.94	2.62	2.97	1.05	1.34	1.48
	Total Time	2.87	5.25	17.9	2.33	4.29	16.7
Overall	Loaded Data	397	611	791	475	663	818
	Effectiveness	6.72	12.7	22.8	6.04	11.9	21.6
	Disk Access Time	0.938	2.74	15.0	1.28	3.00	15.3
	Tree Traversal	1.94	2.89	3.75	1.05	1.46	1.84
	Total Time	2.87	5.63	18.2	2.33	4.46	16.7

Temporal-Cut Query		T-BON			IA-OCTREE		
		MIN	AVG	MAX	MIN	AVG	MAX
Overall	Loaded Data	127	181	227	167	205	233
	Effectiveness	140	234	356	108	207	348
	Disk Access Time	0.398	0.907	4.08	0.530	1.16	4.15
	Tree Traversal	0.151	0.217	0.274	0.204	0.251	0.292
	Total Time	0.549	1.12	4.35	0.734	1.41	4.44

TABLE II

QUERY PERFORMANCE FOR THE RICHTMYER-MESHKOV INSTABILITY DATASET. (LOADED DATA (MB), EFFECTIVENESS (RATIO OF THE NUMBER OF EXTRACTED TRIANGLES (K) TO LOADED DATA (MB)), DISK ACCESS TIME (SEC), TREE TRAVERSAL TIME (SEC), TOTAL TIME (SEC))

performance as compared to its performance at the given time steps. As seen in the table, the total time to render an isosurface at an intermediate time step is roughly 50% more than the time to render an isosurface at one of the given time steps.

Figures 8, 9 and 10 show the rendered images of temporal and spatial cut query results for the Richtmyer-Meshkov dataset. The spatial cuts reveal how the data evolves along the temporal axis, which is not easy to capture with traditional visualization techniques. Figures 11, 12 and 13 show the rendered images of temporal and spatial cut query results in the Five Jets dataset. Figure 14 shows the rendered image at an intermediate time step. This clearly illustrates the usefulness of rendering at intermediate time steps as it highlights how features are changing at a finer granularity.

We also illustrate the effect of the data layout on the performance. Table V shows the speed up in overall disk access time achieved by using our particular lexicographic order versus using the popular z-order. As can be seen from the table, our access time is faster approximately by a factor of 4.7 for the y and z hyperplane queries, while it is slower by a factor of 2.5 for the cuts along the X dimension. Clearly we can change the order in our lexicographic ordering to give preference to any dimension. In the absence of any preferences and under the assumption that the three types of cuts are equally likely,

Spatial-Cut Query at Z		T-BON			IA-OCTREE		
		MIN	AVG	MAX	MIN	AVG	MAX
60	Loaded Data	73.6	241	472	80.7	265	525
	Effectiveness	1.68	3.36	5.50	1.54	3.06	4.95
	Disk Access Time	0.394	7.89	18.6	0.394	8.80	19.9
	Tree Traversal	0.641	1.83	3.63	0.188	0.624	1.27
	Total Time	1.03	9.72	22.2	0.580	9.42	21.1
80	Loaded Data	190	331	406	199	358	436
	Effectiveness	1.93	3.86	6.27	1.84	3.57	5.84
	Disk Access Time	0.237	9.42	17.2	0.618	10.7	18.5
	Tree Traversal	1.61	2.55	3.18	0.485	0.834	1.05
	Total Time	1.85	11.9	20.3	1.10	11.5	19.5
100	Loaded Data	214	367	472	222	390	502
	Effectiveness	3.36	4.74	6.07	3.24	4.47	5.71
	Disk Access Time	0.664	9.96	18.9	0.771	10.5	21.1
	Tree Traversal	2.04	2.82	3.34	0.564	0.902	1.14
	Total Time	2.70	12.7	22.2	1.33	11.4	22.2
Overall	Loaded Data	73.6	313	472	80.7	337	525
	Effectiveness	1.68	3.98	6.27	1.54	3.70	5.84
	Disk Access Time	0.237	9.09	18.9	0.394	10.0	21.1
	Tree Traversal	0.641	2.40	3.63	0.188	0.787	1.27
	Total Time	1.03	11.4	22.2	0.580	10.7	22.2

Temporal-Cut Query		T-BON			IA-OCTREE		
		MIN	AVG	MAX	MIN	AVG	MAX
Overall	Loaded Data	0.013	2.34	7.99	0.015	2.44	8.34
	Effectiveness	7.33	48.6	104	6.66	46.6	100
	Disk Access Time	0.000	0.027	0.247	0.000	0.028	0.270
	Tree Traversal	0.001	0.004	0.014	0.001	0.006	0.019
	Total Time	0.001	0.031	0.261	0.001	0.034	0.289

TABLE III
QUERY PERFORMANCE FOR THE FIVE JETS DATASET. (LOADED DATA (MB), EFFECTIVENESS (RATIO OF THE NUMBER OF EXTRACTED TRIANGLES (K) TO LOADED DATA (MB)), DISK ACCESS TIME (SEC), TREE TRAVERSAL TIME (SEC), TOTAL TIME (SEC))

IA-OCTREE	Given Time Steps			Intermediate Time Steps		
	MIN	AVG	MAX	MIN	AVG	MAX
Loaded Data	0.015	2.44	8.34	0.030	4.88	16.6
Effectiveness	6.66	46.6	100	3.33	23.3	50.0
Disk Access Time	0.000	0.028	0.270	0.000	0.080	0.326
Tree Traversal (+ Interpolation)	0.001	0.006	0.019	0.001	0.021	0.090
Triangle Extraction	0.001	0.088	0.369	0.001	0.087	0.358
Triangle Rendering	0.001	0.015	0.258	0.001	0.019	0.234
Total Time	0.003	0.137	0.916	0.003	0.207	1.00

TABLE IV
TEMPORAL-CUT QUERY PERFORMANCE COMPARISON FOR THE FIVE JETS DATASET. (LOADED DATA (MB), EFFECTIVENESS (RATIO OF THE NUMBER OF EXTRACTED TRIANGLES (K) TO LOADED DATA (MB)), DISK ACCESS TIME (SEC), TREE TRAVERSAL TIME (+ INTERPOLATION) (SEC), TRIANGLE EXTRACTION (SEC), TRIANGLE RENDERING (SEC), TOTAL TIME (SEC))

the average performance of our layout is obviously superior to that of the z-order layout.

	X	Y	Z
Speed up	0.4	3.9	5.5

TABLE V

SPEED UP IN DISK ACCESS TIME USING OUR LEXICOGRAPHICAL ORDER INSTEAD OF THE Z-ORDER. (AVERAGE OVER 42 $x[or\ y] = \alpha$ HYPERPLANE QUERIES AND 13 $z = \alpha$ HYPERPLANE QUERIES OVER 40 TIME STEPS AT ISOVALUE=70 IN THE RICHTMYER-MESHKOV INSTABILITY DATASET.)

V. CONCLUSION

We have presented an effective scheme for exploring 4-D isosurfaces of time varying data using a novel data structure and efficient techniques for out-of-core data movements. Our new techniques lead to effective visualization of large time-varying data sets and provide more insights than previous visualization techniques. In particular, we introduced a new compact and adaptive 4-D indexing structure called the *Information-Aware Octree* (IA-Octree) based on a new entropy-based technique for assessing the relative variability of the data along the spatial and temporal dimensions. Our experimental results showed that this approach can effectively capture spatial and temporal coherence leading to a structure that is very compact and yet quite effective in identifying active blocks for the hyperplane query types. We also proposed an out-of-core scheme using the controllable delayed fetching technique and optimized disk layout, which enabled efficient and scalable rendering.

VI. ACKNOWLEDGEMENT

We would like to thank Mark A. Duchaineau at the Lawrence Livermore National Lab for making the Richtmyer-Meshkov instability data set available to the University of Maryland, Kwan-Liu Ma at University of California at Davis for the Five Jets dataset and Amitabh Varshney for his constant advice throughout this research.

REFERENCES

[1] The ASCI Turbulence project, The Richtmyer-Meshkov dataset, http://www.llnl.gov/CASC/asciturb.

[2] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 163–169.

[3] J. Wilhelms and A. V. Gelder, “Octrees for faster isosurface generation,” *ACM Transactions on Graphics*, vol. 11, no. 3, pp. 201–227, Jul 1992.

[4] Y. Livnat, H.-W. Shen, and C. R. Johnson, “A near optimal isosurface extraction algorithm using the span space,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 1, pp. 73–84, Mar 1996.

[5] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno, “Speeding up isosurface extraction using interval trees,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 2, pp. 158–170, Apr 1997.

[6] C. L. Bajaj, V. Pascucci, and D. R. Schikore, “Fast isocontouring for improved interactivity,” in *Proceedings of the IEEE symposium on Volume visualization*, 1996, pp. 39–46.

[7] Y.-J. Chiang and C. T. Silva, “I/o optimal isosurface extraction,” in *Proceedings of IEEE Visualization*, 1997, pp. 293–300.

[8] Y.-J. Chiang, C. T. Silva, and W. J. Schroeder, “Interactive out-of-core isosurface extraction,” in *Proceedings of IEEE Visualization*, 1998, pp. 167–174.

[9] C. L. Bajaj, V. Pascucci, D. Thompson, and X. Y. Zhang, “Parallel accelerated isocontouring for out-of-core visualization,” in *Proceedings of the IEEE symposium on Parallel visualization and graphics*, 1999, pp. 97–104.

[10] Y.-J. Chiang, R. Farias, C. T. Silva, and B. Wei, “A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids,” in *Proceedings of the IEEE symposium on parallel and large-data visualization and graphics*, 2001, pp. 59–66.

[11] X. Zhang, C. Bajaj, and V. Ramachandran, “Parallel and out-of-core view-dependent isocontour visualization using random data distribution,” in *Proceedings of the IEEE symposium on Parallel visualization and graphics*, 2002, pp. 9–17.

[12] C. Weigle and D. C. Banks, “Extracting iso-valued features in 4-dimensional scalar fields,” in *Proceedings of the IEEE symposium on Volume visualization*, 1998, pp. 103–110.

[13] P. Bhaniramka, R. Wenger, and R. Crawfis, “Isosurface construction in any dimension using convex hullsn,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 2, pp. 130–141, Mar 2004.

[14] Y.-J. Chiang, "Out-of-core isosurface extraction of time-varying fields over irregular grids," in *Proceedings of IEEE Visualization*, 2003, pp. 29–36.

[15] H.-W. Shen, "Isosurface extraction in time-varying fields using a temporal hierarchical index tree," in *Proceedings of IEEE Visualization*, 1998, pp. 159–166.

[16] P. M. Sutton and C. D. Hansen, "Accelerated isosurface extraction in time-varying fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 2, pp. 98–107, Apr 2000.

[17] Q. Shi and J. JaJa, "Efficient isosurface extraction for large scale time-varying data using the persistent hyperoctree (phot)," in *UMIACS-TR-2006-01*, 2006.

[18] H.-W. Shen, L.-J. Chiang, and K.-L. Ma, "A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree," in *Proceedings of IEEE Visualization*, 1999, pp. 371–377.

[19] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley, 1991.

[20] A. K. Jain, *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[21] H. Samet, *The design and analysis of spatial data structures*. Addison-Wesley, 1990.

[22] Time-Varying Volume Data Repository, The Five Jets dataset, http://www.cs.ucdavis.edu/∼ma/ITR/tvdr.html.

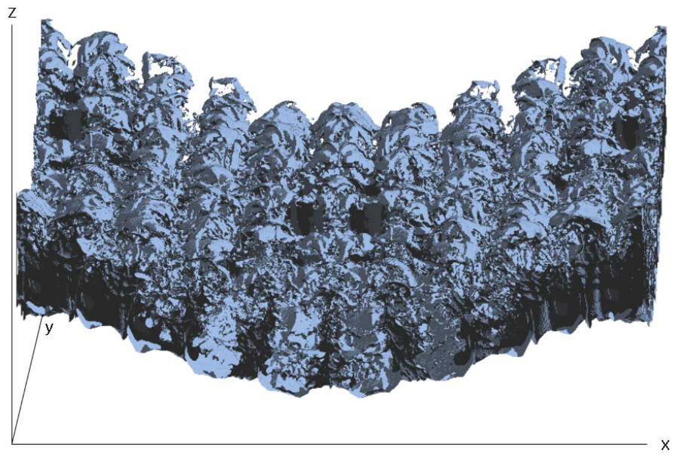


Fig. 8. Isosurface of the Richtmyer-Meshkov instability dataset rendered at isovalue=70 and time step=139.

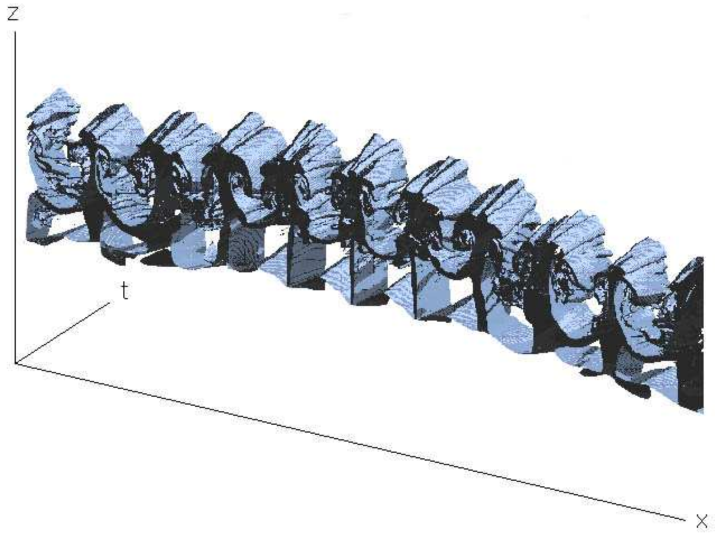


Fig. 9. Isosurface of the Richtmyer-Meshkov instability dataset cut by Y=300 over 100-139 time steps at isovalue=70.

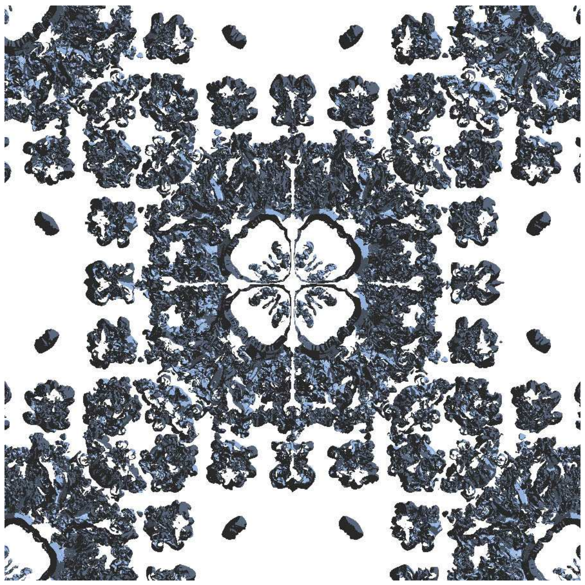


Fig. 10. Isosurface of the Richtmyer-Meshkov instability dataset cut by Z=500 over 100-139 time steps at isovalue=70. (Time axis is orthogonal to the paper.)

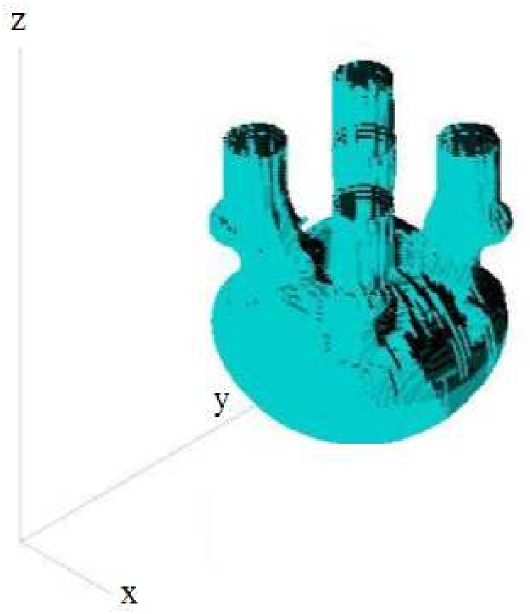


Fig. 11. Isosurface of the Five Jets dataset rendered at isovalue=254200 and time step=1200.

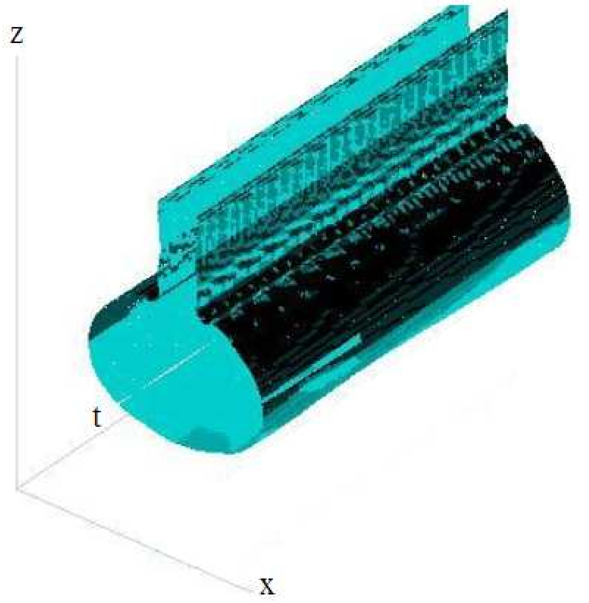


Fig. 12. Isosurface of the Five Jets dataset cut by Y=60 over 1200-1300 time steps at isovalue=254200.

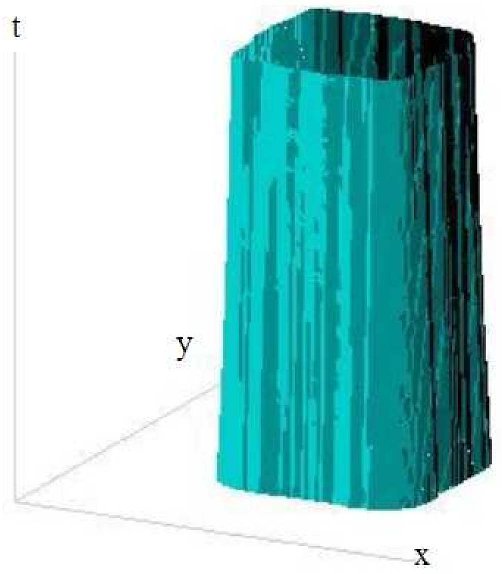


Fig. 13. Isosurface of the Five Jets dataset cut by Z=80 over 1200-1300 time steps at isovalue=254200.

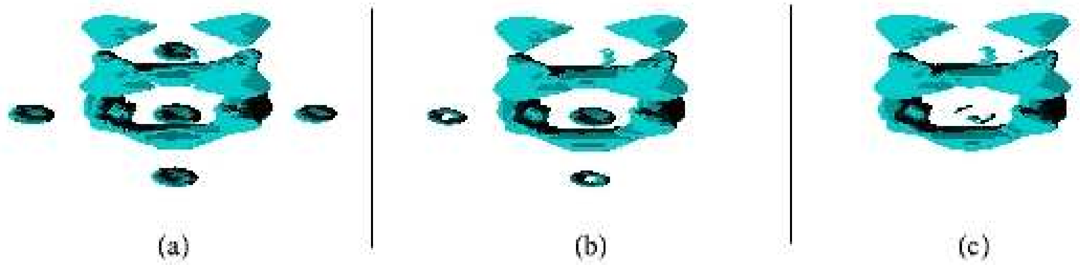


Fig. 14. Isosurfaces of the Five Jets dataset rendered (a) at Time 58, (b) at Time 58.5, (c) at Time 59 (isovalue=274200).