Volume Graphics (2005)
E. Gröller, I. Fujishiro (Editors)

Visualization of Time-Varying Volumetric Data using Differential Time-Histogram Table

Hamid Younesy[†], Torsten Möller[‡], Hamish Carr[§]

Simon Fraser University , Simon Fraser University , University College, Dublin

Abstract

We introduce a novel data structure called Differential Time-Histogram Table (DTHT) for visualization of time-varying scalar data. This data structure only stores voxels that are changing between time-steps or during transfer function updates. It allows efficient updates of data necessary for rendering during a sequence of queries common during data exploration and visualization. The table is used to update the values held in memory so that efficient visualization is supported while guaranteeing that the scalar field visualized is within a given error tolerance of the scalar field sampled. Our data structure allows updates of time-steps in the order of tens of frames per second for volumes of sizes of 4.5GB, enabling real-time time-sliders.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Graphics data structures and data types - E.1 [Data]: Data Structures

Keywords: time-varying data, volume rendering

1. Introduction

As computing power and scanning precision increase, scientific applications generate time-varying volumetric data with thousands of time steps and billions of voxels, challenging our ability to explore and visualize interactively. Although each time step can generally be rendered efficiently, rendering multiple time steps requires efficient data transfer from disk storage to main memory to graphics hardware.

We use the temporal coherence of sequential time frames, the spatial distribution of data values, and the histogram distribution of the data for efficient visualization. Temporal coherence [SH99] is used to prevent loading unchanged sample values, spatial distribution [WG92] to preserve rendering locality between images, and histogram distribution to update for incremental updates during data classification.

Of these, temporal coherence and histogram distribution are encoded in a differential table computed in a preprocessing step. This differential table is used during visualization to minimize the amount of data that needs to be loaded from disk between any two successive images. Spatial distribution is then used to accelerate rendering by retaining unchanged geometric, topological and combinatorial information between successive images rendered, based on the differential information loaded from disk.

The contributions of this paper are to identify and analyze statistical characteristics of large scientific data sets that permit efficient error-bounded visualization. We unify the statistical coherence of the data with the known spatial and temporal coherence of the data in a single differential table used for updating a memory-resident version of the data.

We define coherence and distribution more formally in Section 2, review related work in Section 3 and support our definition of coherence and distribution in Section 4 with a case study. In Section 5 we show how to exploit data characteristics to build a binned data structure representing the temporal coherence and spatial and histogram distribution of the data, and how to use this structure to accelerate interactive exploration of such data. We then present some results

[†] hyounesy@cs.sfu.ca
[‡] torsten@cs.sfu.ca
[§] hamish.carr@ucd.ie

in Section 6 and some conclusions in Section 7, followed by comments on possible future work in Section 7.

2. Data Characteristics

In order to discuss both previous work and our own contributions, we start by defining our assumed input and the statistical properties of the data that we will exploit for accelerated visualization: we will support these assertions empirically in our case study in Section 4.

Formally, a time-varying scalar field is a function $f : \Re^4 \rightarrow \Re$. Such data is often sampled f at fixed locations $V = \{v_1, \ldots, v_n\}$ and fixed times $T = \{t_1, \ldots, t_\tau\}$. Since we exploit this sampling regularity to accelerate visualization, we assume that our data is of the form $\mathscr{F} = \{(v,t,f(v,t)) \; \forall \; v \in V, t \in T\}$, where each triple $(v,t,f(v,t))$ is a *sample*.

Previous researchers report [OM01] that only a small fraction of $\mathscr{F}$ is needed for any given image generated during visualization. Thus, any image can be thought of as the result of visualizing a subset of $\mathscr{F}$ defined by a query q.

More formally, given $\mathscr{F}$ and a condition q at time t (e.g. a transfer function), find the set $\mathscr{F}|q$ of all samples $(v,t,f(v,t))$ for which q is true (e.g. the opacity of $f(v,t)$ is non-zero). Thus, $\mathscr{F}|q$ is the set of samples needed to render q, called the *active set* of q.

To construct a query q for isosurface extraction at isovalue h, $\mathscr{F}|q$ consists of all samples belonging to cells whose isovalues span h. For volume rendering, we base transfer functions on the isovalue, as shown in Figure 4: the active set therefore has a range of isovalues.

We now restate our observation: we expect $\|\mathscr{F}|q\| \ll \|\mathscr{F}\|$. Moreover, $\mathscr{F}|q$ is rarely random, nor is it evenly distributed throughout the data set. Instead, $\mathscr{F}|q$ tends to consist of large contiguous blocks of samples, due to the continuity of the physical system underlying the data, and the inherent organization of most physical systems being studied. We exploit both characteristics to accelerate visualization.

We also exploit the fact that human exploration of data usually involves continuous (i.e. gradual) variation of queries. Formally, we are interested, not in a single query q, but in a sequence $q_1, \ldots, q_k$ of closely-related queries, with occasional abrupt changes to a new query sequence $q'_1, \ldots, q'_m$. We expect that the active sets for any two sequential queries will be nearly identical, i.e. that $\mathscr{F}|q_{i+1} \approx \mathscr{F}|q_i$.

In particular, we exploit *coherence* and *distribution*. Let us denote two samples $\sigma_1 = (v_1,t_1,f(v_1,t_1))$ and $\sigma_2 = (v_2,t_2,f(v_2,t_2))$, and choose small values $\delta, \lambda > 0$.

Spatial coherence is coherence with respect to the spatial dimensions x,y,z of the domain of f, and is based on the spatial continuity of the physical system. Small spatial changes imply small functional changes, i.e. if $|v_1 - v_2| < \delta$ and $t_1 = t_2$, then $|f_1 - f_2| < \lambda$ for small λ.

Temporal coherence is coherence with respect to the time dimension t of the domain of f and is based on the temporal continuity of the physical system. Hence, if $|t_1 - t_2| < \delta$ and $v_1 = v_2$, then $|f_1 - f_2| < \lambda$ for small λ.

Given two sequential queries q_i, q_{i+1} which differ only slightly in space or time, $\mathscr{F}|q_i$ and $\mathscr{F}|q_{i+1}$ will therefore overlap significantly. We exploit this to accelerate the task of updating the active set for rendering purposes, focusing principally on temporal exploration. We also exploit statistical properties of the data sets in question, which we describe in terms of *data distribution*:

Histogram distribution is a property of the functional dimension (i.e. the range of f) and depends largely on the data being studied. However, it is usually true that extracted features are chosen at values where the range of f varies greatly: thus we expect f to be fairly well distributed, although sampling may affect this.

Spatial distribution is a large scale form of spatial coherence, and is based (again) on the physical phenomena scientists and engineers study. Although data is commonly generated over large domains, it is often true that nothing interesting happens in most of the domain. As a result, the active sets for queries made by the user are often clustered in a subset of the space.

Histogram distribution is crucial where two sequential queries q_i and q_{i+1} differ only slightly in the functional dimension. For well-distributed data values, we expect $\mathscr{F}|q_i$ and $\mathscr{F}|q_{i+1}$ to be substantially identical: i.e. that we can change transfer functions more rapidly by loading only the new data values required. Moreover, the spatial distribution of the data allows us to have spatially sparse data structures.

3. Related Work

The difficulty of rendering large data sets is well-recognized in the literature, with successive solutions addressing progressively larger data sets by exploiting various data characteristics. Broadly speaking, however, the solutions proposed have dealt with isosurface extraction and volume rendering as separate topics, whereas our solution is applicable to either with suitable modifications.

Modern isosurface extraction and volume rendering methods are based on the recognition that the problem is *decomposable* into smaller sub-problems. Lorenson & Cline [LC87] recognized this and decomposed the extraction of an isosurface over an entire data set by extracting the surface for each cell in a cubic mesh independently.

Almost simultaneously, Wyvill, McPheeters & Wyvill [WMW86] exploited spatial coherence for efficient isosurface extraction in their *continuation method*. Spatial coherence has also been exploited for volume rendering [WTTL96].

Other researchers [CMM*97, Gal91, LSJ96, WG92] exploit spatial distribution for efficient extraction of isosurface active sets. In particular, Wilhelms & van Gelder [WG92] introduced the *Branch-On-Need Octree (BONO)*, a space efficient variation of the traditional octree. Octrees, however do not always represent the spatial distribution of the data, and are inapplicable to irregular or unstructured grids.

Livnat et al. [LSJ96] introduced the notion of *span space* - plotting the isovalues spanned by each cell and building a search structure to find only cells that span a desired isovalue. Since this represents the range of isovalues in a cell by a single entry in a data structure, it exploits histogram distribution. Histogram distribution has also been exploited to accelerate rendering for isosurfaces [CVCK03] by computing active set changes with respect to isovalue.

Chiang et al. [CSS98] clustered cells in irregular meshes into *meta-cells* of roughly equal numbers of cells, then built an I/O- efficient span space search structure on these meta cells, again exploiting histogram distribution.

For time-varying data, it is impossible to load an entire data set plus search structures into main memory, so researchers have reduced the impact of slow disk access by building memory-resident search structures and loading data from disk as needed.

Sutton & Hansen [SH99] extended branch-on-need octrees to time-varying data with *Temporal Branch-on-Need Octrees (T-BON)*, using spatial and temporal coherence to access only those portions of search structure and data which are necessary to update the active set.

Similarly, Shen [She98] noted that separate span space structures for each time-step are inefficient, and proposed a *Temporal Hierarchical Index (THI) Tree* to suppress search structure redundancy between time-steps, exploiting both histogram distribution and temporal coherence.

Reinhard et al. [RHP02] binned samples by isovalue for each time step, loading only those bins required for a given query: this form of binning exploits histogram distribution but not temporal coherence.

For volume rendering, Shen & Johnson [SJ94] introduced *Differential Volume Rendering*, in which they precomputed the voxels that changed between adjacent time-steps, and loaded only those voxels that changed, principally exploiting temporal coherence.

Shen [SCM99] extended Differential Volume Rendering with *Time-Space Partitioning (TSP) Trees*, which exploit both spatial distribution and temporal coherence with an octree representation of the volume. Binary trees store spatial and temporal variations for each sub-block, and this information is used for early termination of a ray-tracing algorithm, based on a visual error metric. Shen also cached images for each sub-block in case little or no change occurred between frames. Ma & Shen [MS00] further extended this by quantizing the isovalues of the input data and storing them in octrees before computing differentials.

Finally, Sohn and Bajaj [SBS02] use wavelets to compress time-varying data to a desired error bound, exploiting temporal coherence as well as spatial and histogram distribution.

Our contribution in this paper is to unify temporal coherence in the form of differential visualization with statistical distributions in the form of isovalue binning.

Before introducing our data structures, however, we demonstrate the validity of coherence and distribution with some case studies.

4. Case Studies: Data Characteristics of Isabel & Turbulence

As noted in the previous sections, our approach to accelerating visualization depends on mathematical, statistical and physical properties of the data being studied. Before progressing further, it is therefore necessary to demonstrate that these properties are the case. We do so by means of case studies of three different data sets.

The first and largest data set is the Hurricane Isabel data set provided by the National Center for Atmospheric Research (NCAR) for the IEEE Visualization Contest 2004 and has 48 time steps of dimension 500x500x100. It contains 13 different floating point scalar variables of which we study the temperature and pressure. We have also studied the Turbulent Jets (jets) and Turbulent Vortex Flow (vortex) data sets provided by

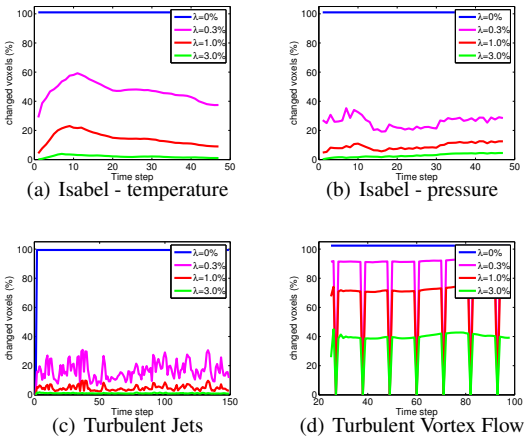
D. Silver at Rutgers University and R. Wilson at University of Iowa to

Kwan-Liu Ma at University of California, Davis. The jets data contains 150 time steps of 104x129x129 floats, while the vortex data contains 100 time steps of 128^3 floating point scalars.

4.1. Temporal Coherence

We demonstrate temporal coherence by computing the difference between adjacent time steps, and graphing the percentage of differences that exceed an error bound expressed as a percentage of the overall data range. As we see in Figure 1(a), at an error bound of 0%, all of the voxels differ from one time step to the next. This may be due to numerical inaccuracies in the floating point data. It therefore follows that 100% of the samples will need to be replaced for the memory-resident version of the data to be accurate.

However, as we see from the second line on the graph, between 40% and 60% of the samples differ by more than 0.3%. Thus, for a 0.3% error in the isovalues displayed, we can save as much as 60% of the load time between queries. As the error bound is increased further, the number of samples to be loaded decreases further: at 1% error, between



(a) Isabel - temperature (b) Isabel - pressure

(c) Turbulent Jets (d) Turbulent Vortex Flow

Figure 1: *Temporal Coherence: Number of samples that change by more than an error bound of* $\lambda = 0\%, 0.3\%, 1\%, 3\%$ *for the temperature and pressure scalars of the Isabel data set as well as the jets and vortex data sets.*

80% and 90% of the samples need not be reloaded, while at 3% error, over 95% of the samples need not be reloaded.

Similar effects can be seen for pressure in Isabel (Figure 1(b)) and for the jets and vortex data sets (Figure 1(c) & (d)). In Figure 1(d), we believe that the sharp periodic dips indicate that certain time steps were inadvertently duplicated. This aside, the overall conclusion is that relatively small error bounds permit us to avoid loading large numbers of samples as time changes.

4.2. Histogram Distribution

In the last section, we saw that the statistics of temporal coherence permit efficient differential visualization for a given error bound. It is natural to ask whether similar statistics are true for small changes in isovalues defining a query. We show in Figure 2 that this is true by examining the histogram distribution for each data set.

For this figure, we set our error bound to 1% and computed the size of the active set for individual data ranges (bins) and for different times, and also the number of samples whose values changed in the temporal dimension. For example, we see in Figure 2(a) that temperature isosurfaces in the Isabel data set never have active sets of more than 5% of the size of a single slice, and that less than half of the active set typically requires replacement between one time step and the next. For pressure, although nearly 25% of the samples are in the active set in the worst case, these samples change little between timesteps, indicating that the physical phenomenon marked by this pressure feature does not move much in space between time steps.

In the jets data set (Figure 2(c)), the worst-case behaviour is exhibited: at an isovalue of roughly 80% of the range, nearly 100% of the samples are in the active set. Although this has implications for the actual rendering step, we see that this set changes little with respect to time. However, as it turns out, these data values constitute most of the "air" surrounding the flow under study and are likely to be ignored during data exploration.

In the vortex data set (Figure 2(d)), we see again that data bins involve relatively few samples, and that these samples have a large degree of temporal coherence.

4.3. Spatial Distribution

Unlike histogram distribution, spatial distribution is harder to test, as it is best measured in terms of the spatial data structure used to store the data. We therefore defer consideration of spatial distribution to the discussion of Table 1 in Section 5.

5. Differential Time Histogram Table

In the previous section, we demonstrated that temporal coherence and histogram distribution offer the opportunity to reduce load times drastically when visualizing large time-varying data sets. To achieve the hypothesized reduction in load-time we introduce a new data structure called the *Differential Time Histogram Table (DTHT)* along with an efficient algorithm for performing regular and differential queries in order to visualize the data set.

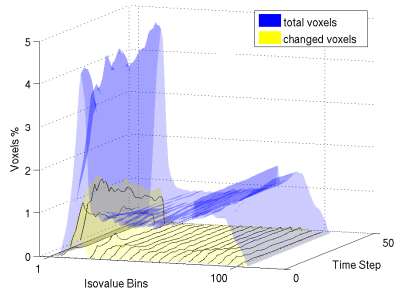
To exploit temporal coherence and histogram distribution, we apply differential visualization in the temporal direction and binning in the isovalue direction. Since we expect user exploration of the data to consist mostly of gradual variations in queries, we precompute the differences between successive queries. We modify temporal coherence by including in the differential set only those samples for which the error exceeds a chosen bound, further reducing the number of samples to be loaded at each time step.

5.1. Computing the DTHT

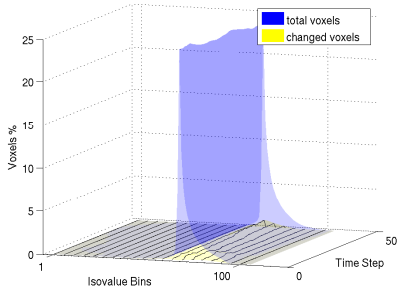
Our DTHT, shown in Figure 3 stores samples in a two-dimensional array of bins defined by isovalue range and time step. In each bin, we store the active set (a), and a set of differences (arrows) between the active sets of adjacent bins.

For a given data set with τ time steps, we create a DTHT with τ bins in the temporal direction and b bins in the isovalue direction. Large values of b will increase the size of the table, but decrease the range of values in each bin, making active set queries more efficient for one bin, but not necessarily faster for a query covering a large range of bins. Also, large numbers of bins will result in more storage overhead.

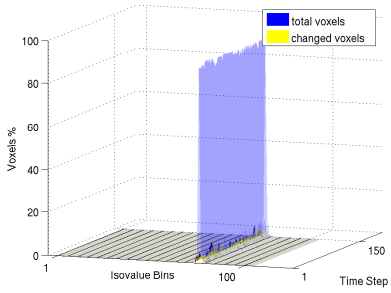
We choose a number b and divide the isovalue range into b



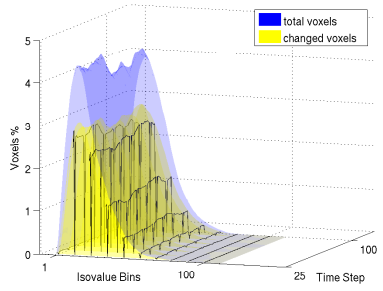
(a) Isabel - temperature



(b) Isabel - pressure



(c) Turbulent Jets



(d) Turbulent Vortex Flow

Figure 2: *Temporal Coherence + Histogram distribution: Number of samples that change by more than an error bound of $\lambda = 1\%$ for the temperature and pressure scalars of the Isabel data set as well as the jets and vortex data sets. Each data set is divided up into 100 iso-value bins.*

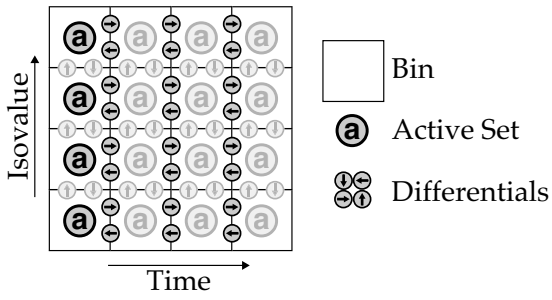


Figure 3: *The Differential Time Histogram Table (DTHT). Each bin holds the active set (a) and differentials between the bin and adjacent bins (arrows). Components not used for our experiments are shown in a lighter shade.*

bins. We also choose λ, the amount of error we will tolerate in isovalues for a single sample over time.

We compute the DTHT by loading the first time step into memory and binning the samples. We then load the second time step into memory alongside the first and compute the difference between the two time steps. Any sample whose difference exceeds the error bound is added to the differential set in the time direction. Any sample whose difference does *not* exceed the error bound is modified so that the value in the second time step is identical to the value in the first time step: this ensures that, over time, errors do not accumulate beyond our chosen bound.

At the same time, differentials are computed in the isovalue direction if these are desired. In our experiments, we have chosen to render using splatting: in this case each sample belongs to only one bin and the active sets in adjacent bins are disjoint. It follows that the isovalue differentials are equal to the active sets, and can be omitted accordingly, we show them in a lighter shade in Figure 3. In case we want to adapt the data structure to support extraction of isosurfaces, each voxel may belong to a range of isovalue bins depending on the values of its neighbors; therefore the bins will have overlap and it would be reasonable to have differentials between neighbor bins to avoid redundancy.

The first time step is then discarded, and the next time step is loaded and compared to the second time step. This process continues until the entire data set has been processed.

For each sample, we store location, isovalue, and normalized gradient vector at a leaf node of a branch-on-need octree. At all times, the current active set is stored in a branch-on-need octree, as are the active sets and the differentials for each bin. Samples are added or removed from the current active octree by tandem traversal with the octrees for the bins.

Instead of single samples, however, we store entire bricks of data (typically $32 \times 32 \times 32$) at octree leaves, either in linked lists (if no more than 60% is active) or in arrays (if

Structure	List		Octree All Active Sets		Octree 1st Active Set	
λ	GB	%age	GB	%age	GB	%age
Isabel - temperature						
0.0%	34.6	737%	27.7	591%	18.5	394%
0.3%	22.4	477%	17.9	382%	8.7	186%
1.0%	15.0	320%	12.0	256%	2.8	60%
3.0%	12.2	260%	9.8	209%	0.59	12%
Isabel - pressure						
0.0%	34.6	737%	27.7	590%	18.5	394%
0.3%	17.6	371%	14.1	301%	4.9	104%
1.0%	13.8	294%	11.1	237%	1.9	40%
3.0%	12.4	264%	9.9	211%	0.72	15%
Turbulent Jets						
0.0%	7.4	673%	5.0	455%	3.4	330%
0.3%	3.4	310%	2.4	216%	0.71	70%
1.0%	2.8	255%	1.9	173%	0.20	20%
3.0%	2.6	236%	1.8	164%	0.06	6%
Turbulent Vortex Flow						
0.0%	4.6	731%	3.7	588%	2.5	394%
0.3%	4.0	636%	3.2	509%	2.0	320%
1.0%	3.5	556%	2.8	445%	1.6	252%
3.0%	2.6	413%	2.1	334%	0.93	150%

Table 1: *Size (in GB and percent of original data set) of DTHT using lists and octrees with all or only the first active set stored.*

more than 60% is active). We save space by storing sample locations relative to the octree cell instead of absolutely, requiring fewer bits of precision.

Moreover, octrees are memory coherent, so updates are more efficient than linear lists, although at the expense of a larger memory footprint, as shown in Table 1. We note, however, that the memory footprint is principally determined by the size of the octrees for the active sets.

Further reductions in storage are possible depending on the nature of the queries to be executed. We assume that abrupt queries are few and far between, and dispense with storing active sets explicitly except for the first time step. If an abrupt query is made at time t, we can construct the active set by starting with the corresponding active set at time 0 and applying all differential sets up to time t. Doing so reduces the amount of storage required even further, as shown in the third column of Table 1. As with the isovalue differentials, we indicate this in Figure 3 by displaying the unused active sets in a lighter shade. We can also store active sets in a subset of a priorily chosen keyframes so that we always start from the nearest keyframe instead of starting from the first frame. Another way of reducing the storage is to remove the active sets and differentials for the non-interesting isovalue ranges (e.g. the empty spaces) which can have a huge impact depending on the nature of the dataset.

5.2. Queries in the DTHT

For any query, we first classify the query as gradual (a small change) or abrupt (a large change): as noted in Section 2, we expect the nature of user interaction with the data will cause most queries to be gradual in nature rather than abrupt.

We chose to implement volume rendering using point-based splatting, in which the active set consists only of those samples for which the opacity is non-zero. This active set may span multiple DTHT cells, as shown in Figure 4, in which case we use the union of the active sets for each DTHT cell. It is also possible to use other volume rendering methods as long as they alow runtime update of the data. For instance, hardware texture slicing would be suitable only if the hardware allows texture updates without requiring to load the whole texture for each update.

Abrupt queries are handled by discarding the existing active set and reloading the active set from the corresponding bin or bins on disk. Because the data was binned, the active set is over-estimated for any given transfer function. This is a conservative over-estimate which includes the desired active set, but is sensitive to the size of the bin. Too few bins leads to large numbers of discards for a particular query, while too many bins leads to overhead in data structures and overhead in merging octrees.

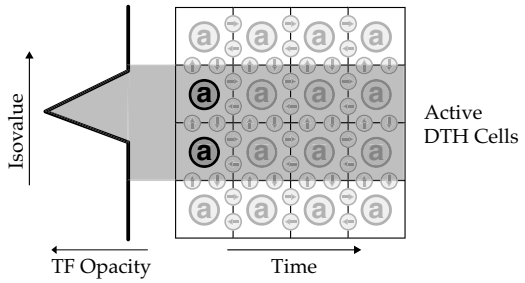


Figure 4: *Active DTHT Cells For a Given Transfer Function.*

Gradual queries in the temporal direction edit the active set using differentials for each bin spanned by the transfer function. Since forwards and backwards differentials are inverse, we only store on each arrow the set of samples that needs to be *added* in that direction. The set of samples that needs to be *removed* is then the set of samples stored on the opposite-direction arrow between the same two bins.

Gradual queries in the isovalue direction discard all samples in a bin when the transfer function no longer spans that bin, and add all samples in a bin when the transfer function first spans that bin. It should be noted that random access to voxels in our octree based data structure is not efficient since it requires top down traversing of all octrees for to access the voxel. Hence, visualizing multimodal data would be generally difficult. We have addressed that in our future work.

For faster rendering, partial images for each base level

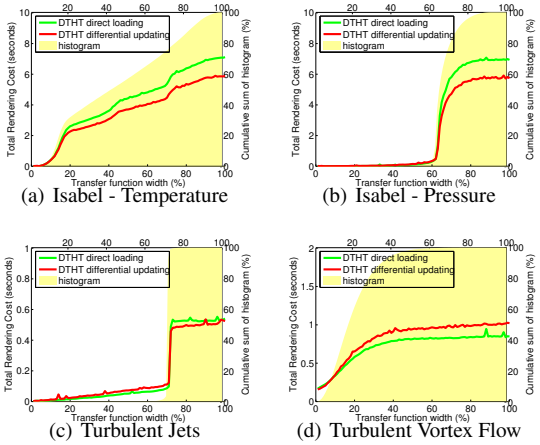
block in the current active octree are stored, as in the TSP tree [SCM99]. Since we update base level blocks in the octree only when changes exceed the λ-bound, these partial images often survive two or more time steps.

6. Results and Discussion

We implemented our differential temporal histogram table algorithm and visualized different scalar time-varying datasets introduced in Section 4, using a 3.06GHz Intel Xeon processor, 2GB of main memory and an nVidia AGP 8x GeForce 6800 graphics card, with data stored on a fileserver on a 100Mb/s Local Area Network.

In our experiments we used $b = 100$ isovalue bins, set the base-level brick size in the octrees to 32x32x32, and tested four different values for error bounds ($\lambda = 0\%, 0.3\%, 1.0\%, 3.0\%$).

In our first experiment we controlled the number of active voxels by changing the transfer function width (the range of non-transparent data values). Starting with a 0% width transfer function covering no bins, we increased the width until all data values were classified as non-transparent and all bins were loaded. The top three rows of the figures in the color plate show representative images under this scenario. In Figure 5, we compare the performance of DTHT active sets alone with the performance of DTHT active and differential sets, rendering each data set with an error bound of $\lambda = 1.0\%$ and transfer functions of varying widths. The performance is principally driven by the width of the transfer function, and it is clear that a significant speed gain in loading the data can be achieved with the differential sets.

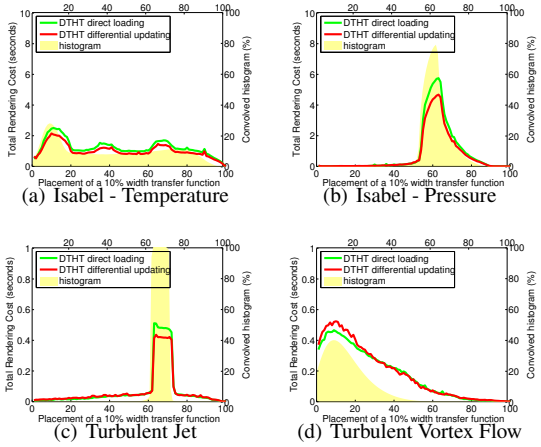


(a) Isabel - Temperature (b) Isabel - Pressure

(c) Turbulent Jets (d) Turbulent Vortex Flow

Figure 5: $\lambda = 1.0\%$ *Rendering and data loading performance for DTHT using active and differential files. The reported times are averaged over multiple time steps.*

We next investigated the performance of our algorithm for different isovalue ranges, collecting the same measurements as before for constant width (10%) transfer functions with different placements. The bottom three rows of the figures in the color plate show representative images under this scenario. The results for $\lambda = 1.0\%$ and different data sets are shown in Figure 6.



(a) Isabel - Temperature (b) Isabel - Pressure

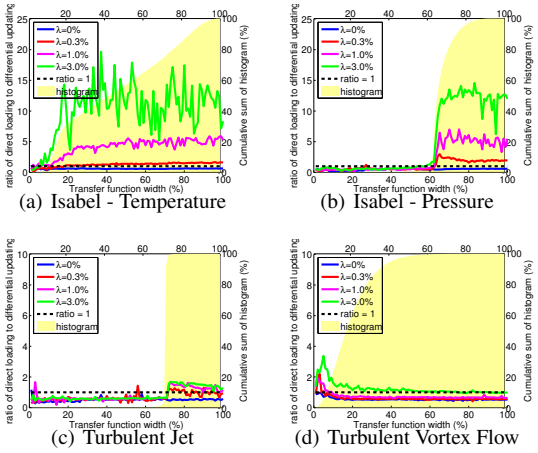
(c) Turbulent Jet (d) Turbulent Vortex Flow

Figure 6: $\lambda = 1.0\%$ *Performance for DTHT using active and differential files, showing also the ratio of active voxels to all voxels (in yellow). A transfer function spanning 10% of the data range was used with different centre points. Results are averaged over several time steps.*

To simplify comparison, we calculated the ratio between the differential DTHT updates and direct DTHT updates for both sets of experiments. Since the goal of this paper was to improve the loading times, we excluded rendering time from our figures. The goal was to investigate the cases in which updating the octree using differential files were faster than straightforward loading of the active files. Figure 7 and Figure 8 show the ratio between the performance of the two methods, respectively for different width transfer function and constant width transfer function.

In a third experiment we investigated performance for a fixed time but varying transfer function. Compared to the brute-force method, where all of the data is loaded, our method takes advantage of the rendering parameters to load only the voxels participating in the final rendering. Changing the transfer function caused new bins to be added to or removed from the data set. The results are shown in Figure 9.

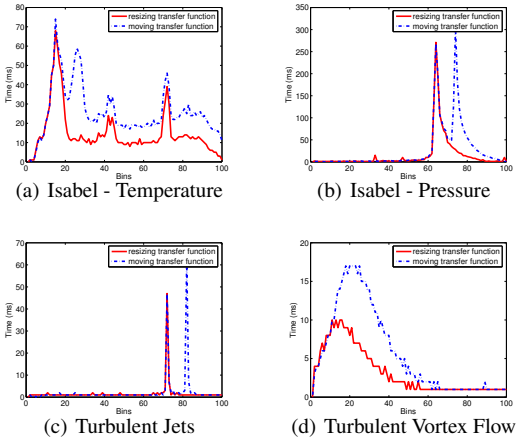
Loading time of DTHT methods depends greatly on the amount of active voxels. In both experiments, when a small number of voxels are active, loading the octree directly from the active files is comparatively faster than using the differential files. It is mainly because of the fact that loading the octree directly from the active_voxels files, requires going through the octree only once per each bin, while in the differential method, it will be twice per each bin; once to remove the invalid voxels, using voxels_to_remove files and then to



(a) Isabel - Temperature (b) Isabel - Pressure

(c) Turbulent Jet (d) Turbulent Vortex Flow

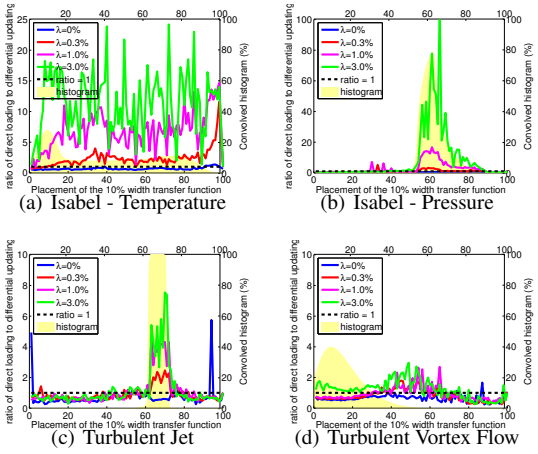
Figure 7: *Ratio between DTHT differential updating time and DTHT direct loading time for different error bounds, for different transfer function widths.*



(a) Isabel - Temperature (b) Isabel - Pressure

(c) Turbulent Jets (d) Turbulent Vortex Flow

Figure 9: *Loading time of incremental transfer function changes.*



(a) Isabel - Temperature (b) Isabel - Pressure

(c) Turbulent Jet (d) Turbulent Vortex Flow

Figure 8: *Ratio between DTHT differential updating time and DTHT direct loading time for different error bounds and different placements of a transfer function of constant width (10%).*

add the new voxels using voxels_to_add files. Hence, memory access time and octree processing time overcomes the files transaction time when small amounts of data are read from the hard drive.

As the number of active voxels increases, more data need to be loaded/updated from the external storage. Hence, the memory-storage transactions become the bottleneck. Since there are relatively few data in the differential files, the performance of updating the octree using the differential files becomes better than the direct loading. However, the ratio of this performance highly depends on the temporal coherence of the data set. For example since the Turbulent Vortex Flow data set isn't as temporally coherent as the other three data sets, even with a 1.0% error bound, the differential updating is slower than the direct loading and that's simply because the number of voxels_to_add + voxels_to_remove is larger than the actual amount of active_voxels.

Noise in the diagrams are mainly due to loading the data through a local area network connection which is fast, but does not always provide a constant data rate. There are also large variations in some parts of Figure 7 and Figure 8. It can be seen that the noise is mostly in the parts in which not many voxels are active. This is because of the fact that, as the loading times become smaller, they become more error prone to be affected by slightest activities in the hardware or the operating system.

7. Conclusion and future work

Efficient exploration and visualization of large time-varying data sets is still one of the challenging problems in the area of scientific visualization. In this paper we studied the temporal coherence of sequential times and histogram and spatial distribution of data in time-varying data sets and proposed a new data structure called differential time histogram table (DTHT) to take advantage of these characteristics to efficiently load and render time-varying data sets.

In a preprocessing step, we encode the temporal coherence and histogram distribution of the data in the DTHT data structure. Later, during the visualization step, we use this differential information to accelerate updating data by minimizing the amount of data that needs to be transfered from disk into main memory. To store the data in external storage and to keep the active data in main memory, we use

an octree data structure which accelerates loading and processing data by exploiting spatial distribution.

The work presented in this paper was mainly focused on analyzing the statistical characteristics of large datasets to reducing the data transfer between memory and external storage during the visualization process. We are planning to further improve performance by applying compression schemes for the data in leaf nodes and adding multi-level information to the sub-levels of the data structure to enable multi-resolution renderings.

Currently the rendering time is dominating the overall time to produce an image. This is due to the fact, that the rendering algorithm has not been optimized at this point. We plan to take advantage of the octree data structure for hierarchical rendering of our data in our next implementation. Image caching, occlusion culling and parallel rendering will further drastically improve the rendering performance.

Our future work includes applying proper modifications to DTHT in order to enable the extraction of isosurfaces and to handle multi-modal datasets, investigating methods to change the error bounds (λ) during run time based on the user control and the transfer funtion and methods to deal with multi-dimensional transfer functions.

References

[CMM*97] CIGNONI P., MARINO P., MONTANI C., PUPPO E., SCOPIGNO R.: Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (1997), 158–169. 3

[CSS98] CHIANG Y.-J., SILVA C. T., SCHROEDER W. J.: Interactive out-of-core isosurface extraction. In *Proceedings of IEEE Visualization '98* (1998), pp. 167–174. 3

[CVCK03] CHHUGANI J., VISHWAMANTH S., COHEN J., KUMAR S.: Isoslider: A system for interactive exploration of isosurfaces. In *Proceedings of Eurographics Visualization Symposium 2003* (2003), pp. 259–266, 304. 3

[Gal91] GALLAGHER R. S.: Span Filtering: An Optimization Scheme For Volume Visualization of Large Finite Element Models. In *Proceedings of Visualization 1991* (1991), IEEE, pp. 68–75. 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics 21*, 4 (July 1987), 163–169. 2

[LSJ96] LIVNAT Y., SHEN H.-W., JOHNSON C. R.: A Near Optimal Isosurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics 2*, 1 (1996), 73–84. 3

[MS00] MA K.-L., SHEN H.-W.: Compression and accelerated rendering of time-varying volume data. In *Workshop on Computer Graphics and Virtual Reality, 2000 Intl. Computer Symposium* (2000). 3

[OM01] ORCHARD J., MÖLLER T.: Accelerated Splatting Using a 3D Adjacency Data Structure. In *Proceedings of Graphics Interface 2001* (2001), pp. 191–200. 2

[RHP02] REINHARD E., HANSEN C. D., PARKER S.: Interactive ray tracing of time varying data. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization* (2002), pp. 77–82. 3

[SBS02] SOHN B.-S., BAJAJ C., SIDDAVANAHALLI V.: Feature based volumetric video compression for interactive playback. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics 2002* (2002), pp. 89–96. 3

[SCM99] SHEN H.-W., CHIANG L.-J., MA K.-L.: A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of IEEE Visualization '99* (1999), pp. 371–377. 3, 7

[SH99] SUTTON P., HANSEN C. D.: Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon). In *Proceedings of IEEE Visualization '99* (1999), pp. 147–153. 1, 3

[She98] SHEN H.-W.: Isosurface extraction in time-varying fields using a temporal hirarchical index tree. In *Proceedings of IEEE Visualization '98* (1998), pp. 159–166. 3

[SJ94] SHEN H.-W., JOHNSON C. R.: Differential volume rendering: a fast volume visualization technnique for flow animation. In *Proceedings of IEEE Visualization '94* (1994), IEEE Computer Society Press, pp. 188–195. 3

[WG92] WILHELMS J., GELDER A. V.: Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (July 1992), 201–227. 1, 3

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data structure for soft objects. *Visual Computer 2* (1986), 227–234. 2

[WTTL96] WAN M., TANG L., TANG Z., LI X.: Pc-based quick algorithm for rendering semi-transparent multi-isosurfaces of volumetric data. In *Proceedings of Visualization 1996* (1996), pp. 54–61. 2

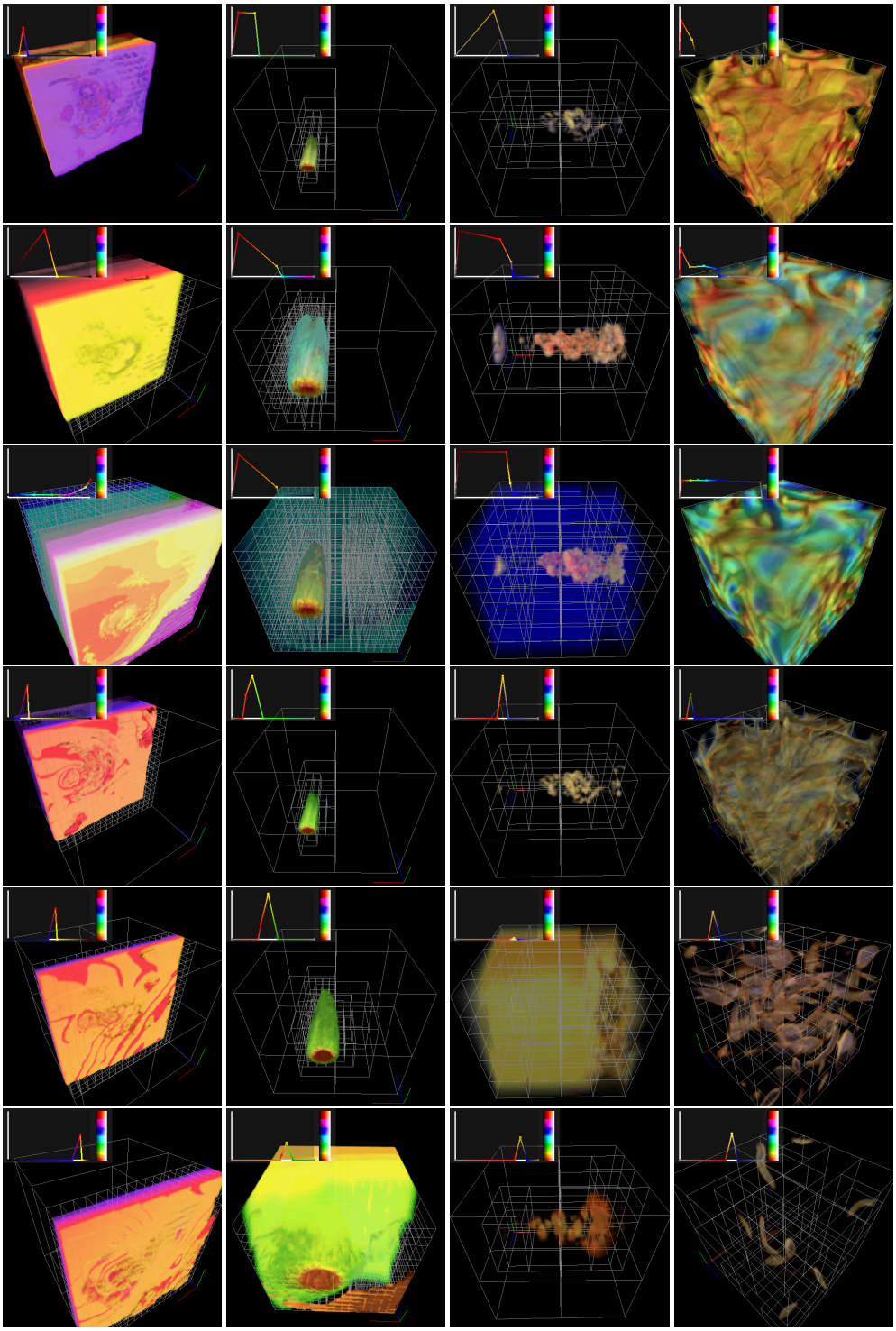


Figure 10: *Rendered Images with representative transfer functions. Displayed are six rows of each data set - (left to right) Isabel temperature, Isabel pressure, Turbulent Jets and Turbulent Vortex Flow. The top three rows show growing transfer functions, and the bottom three rows show shifting transfer functions, corresponding to our experiments in this section.*