ADR Visualization: A Generalized Framework for Ranking Large-Scale Scientific Data using Analysis-Driven Refinement

Boonthanome Nouanesengsy*, Jonathan Woodring†, John Patchett‡, Kary Myers§, and James Ahrens¶
Los Alamos National Laboratory

ABSTRACT

Prioritization of data is necessary for managing large-scale scientific data, as the scale of the data implies that there are only enough resources available to process a limited subset of the data. For example, data prioritization is used during *in situ* triage to scale with bandwidth bottlenecks, and used during focus+context visualization to save time during analysis by guiding the user to important information. In this paper, we present *ADR* visualization, a generalized analysis framework for ranking large-scale data using Analysis-Driven Refinement (ADR), which is inspired by Adaptive Mesh Refinement (AMR). A large-scale data set is partitioned in space, time, and variable, using user-defined importance measurements for prioritization. This process creates a *prioritization tree* over the data set. Using this tree, selection methods can generate sparse data products for analysis, such as focus+context visualizations or sparse data sets.

Keywords: data triage, focus+context, large-scale data, big data, scientific data, prioritization, adaptive mesh refinement

Index Terms: I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing; I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms

1 INTRODUCTION

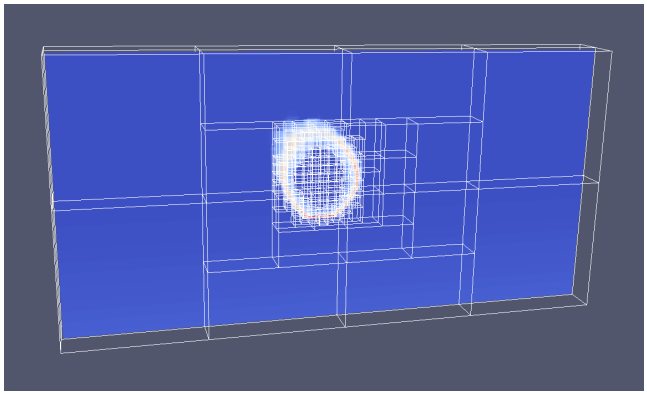


Figure 1: ADR, Analysis-Driven Refinement, using the maximum value of a partition as the importance measurement for a 3D asteroid simulation. A partition is refined if the maximum value meets an importance criteria. Data priority, or importance, is the refinement depth of the data in the priority tree.

*boonth@lanl.gov
†woodring@lanl.gov
‡patchett@lanl.gov
§kary@lanl.gov
¶ahrens@lanl.gov

Large-scale data implies that there is insufficient computational or human resources to fully process all of the data. If all of the data will not be processed, decisions must be made as to which data will be processed, which data will be looked at first, and which data will be culled or ignored. For example, in time-evolving, large-scale simulations, a common way to save I/O bandwidth is to store time step data at periodic intervals. This is an explicit culling of the time series, deciding *a priori* which parts of the data are important enough to use in later analysis. Likewise, a focus+context visualization, like automatic camera guidance, is able to focus the user's attention, saving them time. The viewpoint is positioned toward more important data, while less important data might never be looked at.

The common thread in both of these examples of data analysis workflows is that data elements are ranked with respect to each other, and some data is identified as more important and thus chosen for resource access while others are not. Analysis-Driven Refinement (*ADR*) visualization is a generalized framework for prioritization of large-scale data for applications in analysis in resource-constrained environments. Our method prioritizes a data set by creating a *prioritization tree*, which is a relative ranking of the data by recursive partitioning, as seen in Figure 1. The first key concept is that ADR creates an analysis-driven Adaptive Mesh Refinement-like (AMR) grid, using user-defined importance measurements for data partitioning. This analysis grid spans space, time, and variables, and is independent of the source data set's original grid.

The second key concept in ADR is the ability for end-users to define custom measurements for partitioning the tree. User-specified measurements define what is important in a data set and determines whether recursive partitioning should continue. Examples of measurements include minmax, standard deviation, entropy, or value range to determine if data is important enough to be partitioned further. Importance measurements can also be domain specific, such as counting the number of eddies in an ocean simulation partition, or a halo census in a cosmology data partition.

The third key concept in ADR is that the resulting prioritization aids selection algorithms in generating sparse data products for explicit data budgets, such as *in situ* data triage and focus+context visualizations. This prioritization system specifically improves upon previous streaming architecture research [1], by describing a generalized framework for prioritizing data. Thus, our contribution is a framework for prioritizing large-scale scientific data by: 1) an AMR-like partitioning over a data set, independent of its grid, which 2) utilizes user-defined measurements during partitioning, and 3) is applicable to many different triage, analysis, and focus+context visualizations that have data budget constraints.

2 RELATED WORK

In ADR visualization, the priority tree creates a high-dimensional refinement grid, which is very similar in appearance to an Adaptive Mesh Refinement (AMR) grid. AMR is a technique for improving the runtime of scientific simulations by changing the mesh cell size and computation requirements for parts of the computational domain. AMR has been shown to be essential in a wide variety of scientific fields, including fluid dynamics [16], astrophysics [8][14], cosmology [20][21], computer graphics [13] and

many others. Berger *et al.* [2][3] introduced the approach of block-structured AMR, which covers the computation domain with a hierarchical data structure of Cartesian grids and subgrids. Another popular approach for AMR is using finite-element models on unstructured meshes, an approached used by Lohner [12] for computational fluid dynamics. For scientific simulations which employ AMR, one possibility is to use the AMR grid generated by simulation for visualization and analysis purposes. One problem with that approach, for ADR, is the AMR grid represents the finest resolution required over all variables.

ADR narrows down subsets of data based on importance measurements, and the result is similar to feature detection and extraction. Silver and Zabusky [19] use a flood fill algorithm to extract features from a 3D volume. Post *et al.* [17] use a similar technique for feature extraction and then represents those features as icons. More recently, Tzeng and Ma [23] use machine learning techniques on the problem of feature extraction in order to reduce the need for manual intervention. The system learns to extract features with certain properties through limited user interactions. Ji and Shen [10] use earth mover's distance to find the globally best match for tracking features over time. A survey of feature extraction and tracking techniques for flow fields is given by Post *et al.* [18].

Data selection and triage are important techniques for large-scale data, which can drastically reduce the amount of data written to disk or transmitted over a network. Indeed, one primary goal of ADR is to provide a flexible prioritization framework for data triage. Tong *et al.* [22] developed a method to select salient timesteps of a time-varying dataset using dynamic time warping. Their method, though, finds a globally optimal solution and thus requires all timesteps to be available at once. Because of this, their method is not suitable for *in situ* uses. Woodring *et al.* [29] utilize statistical sampling techniques to create a level-of-detail representation of a cosmology simulation. Modeling a time-varying data set as a 3D array with Time Activity Curves (TAC) is the approach used by Fang *et al.* [7] to locate user specified regions and timesteps of interest. Biswas *et al.* [4] used mutual information to find groups of variables in a multivariate data set which had high information overlap.

Similar to our work, Wang *et al.* [25][26] partitioned volume data in image space based on entropy to guide level-of-detail selection. Our framework is more general, as it is applied in data space, which allows many other types of measurements, and selection algorithms to be used. Another work by Wang *et al.* [27] calculated an importance curve for each block of a time-varying data set by using conditional entropy, and clustered the data using these importance curves.

Another goal of ADR visualization is focus+context visualization, to save scientist and analyst time. Viewpoint selection has been studied extensively for several different types of data. In 2005, Bordoloi and Shen [5] generated informative views for volume rendering by using a viewpoint goodness measure, which included using entropy and taking into account the transfer function. Viewpoints for vector fields were evaluated by Lee *et al.* [11] by calculating the entropy of a vector field, and using a maximal entropy projection framebuffer to obtain a view dependent measure. ADR can utilize several different measurements for automatic camera placement. Muehler *et al.* [15] produced viewpoints of anatomical structures from medical images. Several parameters, including occlusion and viewpoint stability were used in their method. Vazquez *et al.* [24] calculated viewpoint entropy by measuring the distribution of projected mesh polygons to find good cameras views for image-based rendering.

3 ADR: Analysis-Driven Refinement

Figure 2 displays an overview of the ADR visualization framework. The first step, and primary focus of this paper, is the *organization* step, where input data is ranked by partitioning them into a prioritization tree. An example of a spatial partitioning can be seen in Figure 1. A large-scale data set is recursively refined, or split, into partitions. For each partition, an importance measurement is calculated, and the measurement is tested against *importance criteria*. If the measurement passes the importance criteria, the partition is split and each resulting child partition is recursed on. An example would be refining a data set based on maximum temperature as the importance measurement, and setting the importance criteria as a user-defined temperature threshold. For each partition, the maximum temperature is calculated, and then the partition is refined if the maximum is greater than the user threshold. The relative importance between the resulting data partitions is determined by the depth at which the partitions occur in the tree. Construction of the prioritization tree is described in Section 3.2.

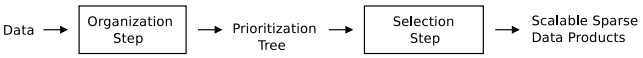


Figure 2: An overview of the ADR visualization framework.

The second step in ADR visualization, the *selection* step, uses the resulting prioritization information to generate data products, such as sparse data sets or focus+context visualizations. The selection step is discussed in Section 3.7. Examples of the selection step include using the prioritization tree to determine automatic camera placement, or choosing only the most important data partitions to write to disk.

3.1 Data Model

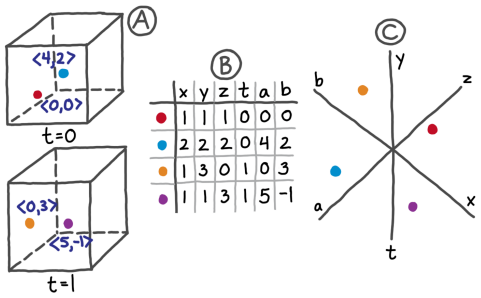


Figure 3: Different views for representing the same time-varying data set, where $a, b \in v$ are a set of variables at a position in space, x, y, and z, over time t: (A) "traditional" spatio-temporal view, (B) table (matrix) view, and (C) high-dimensional projection using star coordinates.

The primary focus for ADR visualization is to prioritize data generated by large-scale scientific simulations that are run on supercomputers. Therefore, the input data set to ADR visualization is assumed to be a time-varying, multivariate data set with spatial information. In particular, ADR is designed to be run *in situ*, so that data may be prioritized for triage operations that occur while the data is still in supercomputer memory. For sake of clarity, we will describe the abstract data model in several different ways, which are shown graphically in Figure 3. The most familiar data model for a time-series simulation (with a set of discrete time steps, t) is spatial blocks over time (t_i in t). The set of spatial data points, per time step t_i, have position (x, y, z coordinates) and field values (a set of variables, v, per point).

Alternatively, we can describe this spatio-temporal data set as a table or matrix, with c columns and r rows. The c columns describe the x, y, z, t, positions of all the data in space-time, along with their v values per point (row), for a total of $|v|+4$ columns. The

total number of rows (points) r is equal to $|t| * n$, where n is the average number of data points per discrete time step in t. Finally, the data in the table can also be described by a set of $|t| * n$ points in $|v| + 4$ dimensional space, where the coordinates of a point p is $(x_p, y_p, z_p, t_p, v_p^0 ... v_p^{|v|-1})$.

3.2 Prioritization Tree

A data set is recursively split into subpartitions, and Figure 4 shows an illustration of this applied to the three data models, which will result in a hierarchical tree representation. The core algorithm that we use is a high-dimensional partitioning, similar to a kd-tree, where high-dimensional data points are recursively partitioned into hypercubes, inspired by stratification and latin hypercubes [29]. Our particular axis partitioning order is time groups (t dimension) first, followed by variable groups (v dimensions), and ending with space (x, y, and z dimensions) groups. Divisions (partitions) are decided by user-defined, analysis-driven importance criteria, which decide whether partitioning should continue. This is similar to how a traditional spatial kd-tree will continue to spatially subdivide data until it reaches stopping criteria.

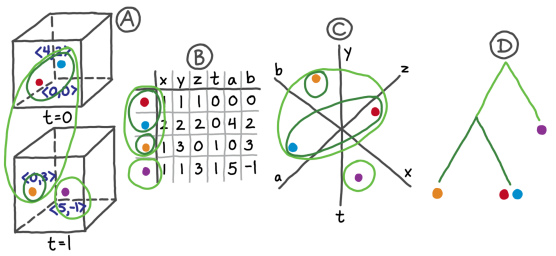


Figure 4: One hierarchical partitioning of the data set, shown in (A), (B), and (C), across the three data models. (D), the priority tree, is equivalent to this partitioning.

Our partitioning algorithm is different from a simulation AMR-octree or a visualization kd-tree in two primary ways: 1) we use analysis-driven criteria to control partition refinement, and 2) we partition the time and variable "dimensions" in addition to the spatial dimensions. Considering a recursive partitioning of a spatio-temporal data set, the root of the tree represents a cluster containing all of the data in a data set. Nodes further down the tree are data points contained within a space-time-variable partition. This new grid is created independently of the source data grid, which is based on refinement using the importance measurements. Different example measurements are discussed in Section 3.6. In general, our ADR construction forms a high-dimensional tree of data points, similar to kd-trees for k-NN search.

3.3 Top-Down Partitioning

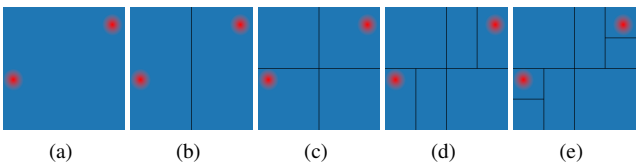


Figure 5: An illustration of the top-down algorithm for the spatial partitioning of a single time-variable partition. Low values are blue, and high values are red. The importance measurement is the maximum value of the partition. The importance criteria is whether the maximum value is close to the global maximum, within a user-defined percentage.

In constructing the prioritization tree, we have developed two methods, a top-down and bottom-up approach. The top-down approach is easier to explain, while the bottom-up approach is more practical from an implementation point-of-view. For now, let us assume that we have all of the data from a time-series, spatial, multivariate data set. In a top-down fashion, the priority tree is created by applying *time importance measures* to the data set, and partitioning it by time values, if the data passes the criteria measurements. Variables, and then space, are partitioned in a similar manner, using *variable importance measures* and *space importance measures*. Partitioning stops if the importance measurement does not pass the importance criteria. Figure 6 describes the steps involved in the top-down approach, while Figure 5 shows a concrete example of the spatial partitioning process.

1: **given:** a data set of one partition
2: **constraint:** apply partitioning in sets of dimensions: time, variable, and space order
3: **if** partition is larger than smallest permissible for a dimension **then**
4: calculate the importance measurement of the current partition
5: **if** the measurement passes the importance criteria **then**
6: split the partition into child partitions
7: recurse on step 3 with each child partition
8: **end if**
9: **end if**

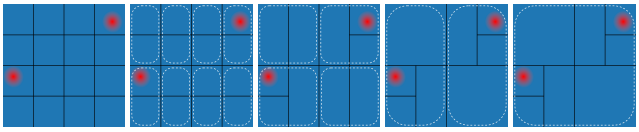
Figure 6: The top-down approach for partitioning

The top-down algorithm will recursively partition a data set by the time dimension, until recursion cannot continue (i.e., either we have reached the smallest time partition size or the time importance criteria has failed). Then, it will partition the data set by variables, and then space, until partitioning cannot continue in those dimensions, either. The importance measurements and criteria, used in steps 4 and 5, need to be supplied by the user. They define how the data set is partitioned according to analysis-driven criteria, such that we are able to use several time, variable, and space importance measurements and importance criteria. For now, we restrict ourselves to axis aligned splitting, such that we form hypercube partitions in $|v| + 4$ dimensional space (i.e., we split and create ranges of values on a dimension). Note that any general type of split is acceptable in the ADR framework.

The top-down approach is easy to understand, but it has multiple drawbacks when trying to implement it in parallel. Typically for a large-scale simulation, the data for a particular discrete time step will be distributed over several processes. Therefore, to calculate the importance measurements *in situ*, importance values or the data itself need to be gathered from all of the processes in a partition. For example, calculating the importance value for a time partition requires gathering all data from all of the processes that the time steps are distributed over. Secondly in the top-down approach, the data set will be read and communicated multiple times (in the worst case, as many times as the maximum depth of the priority tree), as the importance measure has to be recalculated after every splitting step. Due to these drawbacks, we developed an alternative, bottom-up construction.

3.4 Bottom-Up Merging

An alternative approach is to build the prioritization tree from the bottom-up through merging partitions together, which amounts to a result equivalent to the top-down approach. One requirement for



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

Figure 7: An illustration of the bottom-up merging approach for the smallest spatial partitions of a single time-variable partition that is analogous to Figure 5. White dotted borders show parent partitions being created from merging child partitions. Child partitions only remain in the tree if the parent meets the importance criteria.

this bottom-up algorithm is that all importance measurements must be order-independent reducible (i.e., associative). This is not a strict constraint for the top-down construction. Figure 8 describes the steps for the bottom-up approach and Figure 7 shows an example of creating the spatial partitioning of the tree.

1: **given:** a data set split into time-space-variable partitions of the smallest permissible
2: **for each** partition, calculate importance measurements
3: **constraint:** apply merging in sets of dimensions: space, variable, and time order
4: **while** the tree is not complete **do**
5: create a parent partition, merging valid child partitions
6: reduce the child importance to create parent importance
7: **if** parent importance meets the importance criteria **then**
8: keep the children of the parent in the tree
9: **else**
10: prune the children, the parent is a now a leaf
11: **end if**
12: **end while**

Figure 8: The bottom-up approach for partitioning

In this method, rather than recursively splitting, we pre-split all of the data into the smallest partitions permissible, and gather them into a priority tree. This is quite similar to a hierarchical, bottom-up clustering approach, where a data set is split into the smallest elements and then continually merged, until the tree is complete. To make our algorithm equivalent to the top-down splitting approach, we evaluate whether the merging of two children would result in a parent that would have split in the top down case (step 7). If not, we prune the children from the tree.

The computational advantage, primarily for parallelism, is that the points in a data set are only read once to generate subsequent importance values. The importance value of a parent partition is calculated from the children through associative reduction of importance. For example, if the importance measurement is maximum temperature, then the measurement for a parent partition can be found by finding the maximum among the all importance measurements of children partitions. A top-down, splitting approach will have to constantly rebalance and communicate data importance values to processors that share data within an ADR partition, which significantly slows down the computation. Also, the bottom-up parallel communication happens in pairs, while the communication pattern in the top-down parallel approach will have scatter-to-many. As mentioned earlier, this does require that the importance measurements are associative for bottom-up merging, but this is rarely a serious constraint, particularly for our test cases.

3.5 Incremental Time Handling

Strictly speaking, time partitioning cannot happen as described in prior sections for the majority of time-varying simulations. This is because generally only one discrete time step is available *in situ*, due to time-evolving simulations and supercomputers having limited memory. Therefore, evaluating the importance of a discrete time step needs to be done in an out-of-core, best-effort, and/or heuristic manner. The splitting/merging step for the time dimension has to be done with this expectation in mind, such that the data from an entire time step may never be saved, due to the selection (triage) step.

Approaches we have taken use streaming, out-of-core, statistical calculations to determine time partitions. For example, we have experimented with time-series data models (not described in this paper) using the current time step and past time steps. This is used to predict if we need to create a new time partition, or if the current time step belongs to a previous partition. Another problem is that future time-series data (as well as past data, which may not be saved) is also not known at a particular instance in time. Therefore, we must use a best-effort approach to predict if the current time step will be important, relative to future data, using projections of past statistical time-series data. Time partitioning is difficult, and likely an unsolvable problem in the context of time-series simulations. Despite this, there are many different strategies for time partitioning that could be developed in the future.

3.6 Importance Measurements and Importance Criteria

The prioritization tree construction relies on importance measurements and importance criteria to determine if a partition is significant enough to subdivide. This framework is flexible enough to allow many types of importance measurements to perform arbitrary data refinement. For computational efficiency, spatial importance measurements, in particular, ought to be associative to be able to apply our bottom-up parallel implementation. Importance measurements are categorized into three types, based on the level of the tree currently being evaluated: time axis, variable axes, or space axes.

One special case is that an explicit stopping criteria should be used in any ADR implementation: stop refinement if the priority tree data structure is growing too large. The priority tree is an additional data structure that will temporarily increase the memory footprint, but this is a tradeoff to intelligently lower the cost of what a simulation actually saves to disk. Note that the priority tree is a temporary data structure, because after it is created, it is immediately used to triage and select data, and then discarded afterwards.

The temporary memory cost can be controlled by a variety of factors: maximum recursion depth, size of the priority tree to memory allocation ratio, maximum number of leaves, maximum subtree size, etc. For most use cases, we do not actually need the inner structure of the priority tree, and only care about the leaves. Thus, the tree data structure size can be reduced through depth-first, tail recursion in the top-down implementation. In the bottom-up construction, we merge bounding hypercubes, which means if left uncontrolled we will roughly double the size of the memory footprint in the worst case. That is the same memory overhead footprint as in the top-down method in a worst-case, tail-call optimized implementation that only keeps leaf hypercube bounds.

3.6.1 Time Importance Examples

Modulo Time Index: This is the typical time selection algorithm employed by most scientific simulations, where time steps are saved at periodic intervals. The importance measurement for this calculates the time index, and the criteria is whether the index is modulo m. The drawback for this is that it does not inspect the time-series data, and just marks each mth time step as important.

Entropy Change: We calculate the entire entropy of all of the data in a time step. The importance criteria is whether the difference

between the entropy of the current time step and a previous important time step exceeds an absolute delta in bits. This allows us to detect large-scale changes in the distribution of values in a time-evolving simulation and use those time steps as partition markers, as shown in Figure 9.

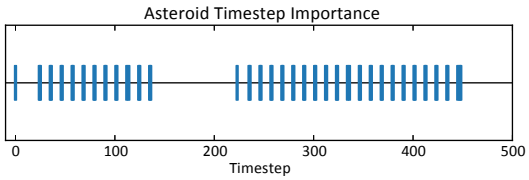


Figure 9: Incremental time partitioning for an asteroid impact simulation by measuring change in entropy. Out of the 500 time steps, they were grouped into 65 time partitions.

3.6.2 Variable Importance Examples

Kolmogorov-Smirnov: We generate the histograms for a variable and keep the last histogram that met the importance criteria. The importance is the Kolmogorov-Smirnov (K-S) distance between the two histograms. The criteria is met if the distance exceeds a threshold. In this case, the current histogram for the variable is significantly different from last histogram of the variable. Figure 10 shows examples of variables marked as important over time by using the K-S distance.

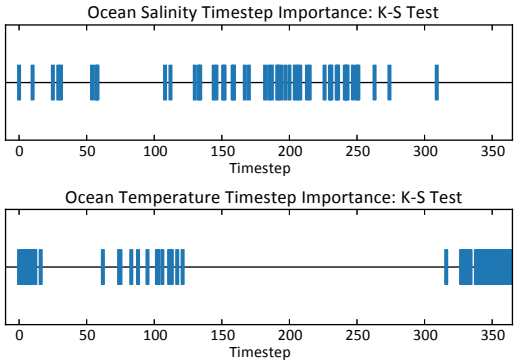


Figure 10: An ocean climate simulation, showing variable refinement over time, using the Kolmogorov-Smirnov distance. Out of 365 time steps, 48 salinity and 61 temperature variable partitions passed the importance criteria, and were refined further in space.

Top N Mutual Information: Using the work introduced by Biswas *et al.* [4], we are able to rank variables relative to each other based on clustering them by the amount of mutual information contained among the different variables. Using the mutual information graph in their method, we assign a rank to each of the variables depending on the total amount of information overlap in a cluster. The importance criteria picks the top ranked variable from each cluster, which is the most informative variable.

3.6.3 Spatial Importance Examples

Shannon Entropy: Information entropy measures the apparent randomness or predictability of a set of data. Figure 11 shows an example of using entropy as a spatial importance measurement, with partitioning occurring when the entropy is relatively high.

Value Range: Prioritizing user-specific values allows the data to be ranked based on values of interest. Figure 12 shows an example of spatial partitioning of data using maximum value and user-selected values of interest.

Feature Census: Any domain-specific feature measurement coupled with an importance criteria can be used in priority tree construction. For example, the Okubo-Weiss (or Lambda-2) parameter,

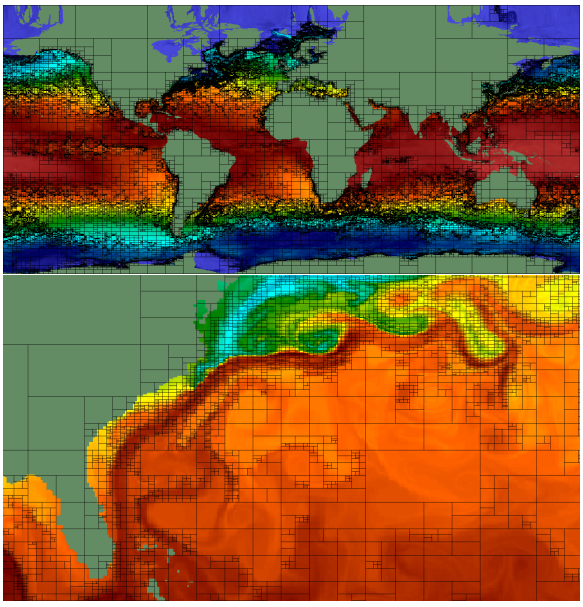


Figure 11: Spatial partitioning using entropy as the importance measurement. The importance criteria is whether entropy is relatively high. The top figure shows the refinement of the entire spatial domain of an ocean simulation, while the bottom figure shows a zoomed in region of the same refinement.

for eddy detection in ocean simulations [28], can be used to prioritize data based on the eddy census meeting a threshold. Similarly, other features, such as the number of halos in dark matter cosmology simulations or curvature surface count in seismic data, can be used to prioritize a data set.

3.7 Data Triage and Filtering: The Selection Step

Once the organization step has completed and the prioritization tree has been built, the selection step is performed. Selection algorithms walk through the prioritization tree to generate sparse data products. This is where the triage takes place, as the data has been ranked, and it is up to the selection step algorithm to determine which data to keep, based on the priority.

Depth in the priority tree describes the relative importance of data, i.e., the data within a partition deep in the prioritization tree has passed the importance criteria more often than data which is present at a shallower part of the tree. Therefore, the most important data, as determined the by the priority tree, is the data in the leaf partitions.

The basic tree-walking algorithm for data selection is to *walk the leaves of the priority tree with depth as the priority value of the partition.* The selection step method defines what data to save and what data products to generate from the leaves. Below, we provide several data product generation examples and describe how the data is selected and stored from the leaves of the priority tree.

Raw Data Storage: Typical scientific simulations will store "vis dumps," or raw data every few time steps for post-processing visualization and analysis (see *Modulo Time Index* in Section 3.6.1). With ADR, we can have data-introspective control over raw data storage by choosing to save time steps and/or variables which have passed the analysis-driven criteria. For example, Figures 9 and 10 show time steps and variables that have passed entropy and Kolmogorov-Smirnov importance criteria, respectively. In these cases, we can trigger raw data storage to disk, rather than in periodic intervals, to store only one time step or variable per partition.

Camera Selection: In focus+context visualization, important

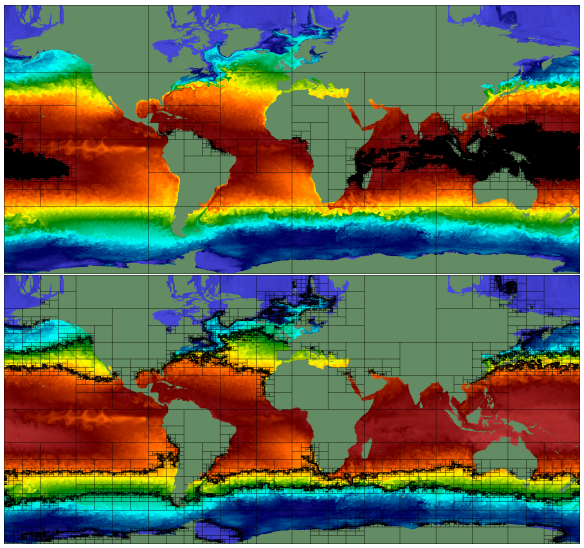


Figure 12: Spatial partitioning utilizing different temperature importance measurements for the ocean simulation data set. The top figure uses the global maximum value, while the bottom figure uses three user-selected values of interest. A partition passes the importance criteria if the values in the partition are within a threshold of the target importance values. Notice that the partitioning is different.

data points are highlighted for the user to save time or highlight data points that may be obscured. In ADR, one way to define focus areas is to choose the spatial partitions corresponding to the lowest leaves in the tree. To visually focus on these partitions, we can create camera bounds around clusters of the lowest leaves in the tree, as seen in Figure 13. In this example, we prune higher leaves from the tree and spatially cluster the remaining leaves to form groups for camera positioning.

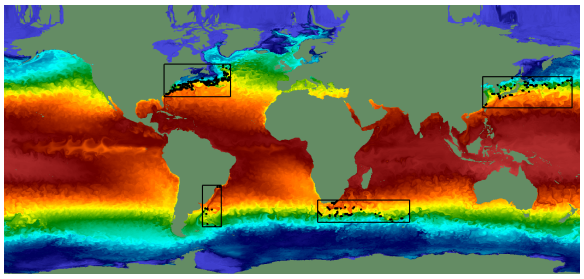


Figure 13: An example of producing camera placements using ADR. Tree partitions whose level are less than a threshold are filtered out, while the remaining partitions are clustered using K-means. The bounding boxes of the resulting spatial clusters form camera bounds, with four clusters shown.

Partition Database: The ADR priority tree algorithm essentially builds an index for the data, as the data is sorted by their position in the tree (depth, space, time, and variable). Each leaf has a bounding hypercube that defines the unique range of the data in the partition, and the priority tree is a search tree for that data. Using this indexing, we can store the data directly in a database [6, 9]. The search keys for those data points are the partition bounds from the priority tree.

Table 1 shows the size of a database through different storage schemes for ocean climate data in an SQLite database. The three schemes shown are: 1) store all of the data in a partition (just use the priority tree indexing), 2) compress the data in each partition, and 3) randomly sample data from the partitions, equivalent to stratified random sampling seen in [29].

Disk Usage for Database Schemes

Max Depth	Leaves	Full Data	Compress	Sample
6	58	1300 MB	583 MB	6.2 MB
10	423	1300 MB	582 MB	17 MB
14	1486	1300 MB	582 MB	46 MB
18	2753	1300 MB	612 MB	159 MB

Table 1: Disk usage for ADR partitioned ocean climate data that is stored in an SQLite database. The data in each partition is stored in the database, indexed by the depth in the tree and bounds of the partition (space, time, and variable). The Full Data column is the size of the database if all the data in a partition is stored. The Compress column is the size if the partition data is compressed with *gzip*, and the Sample column is the size if a random sample is taken per partition.

Streaming Visualization: Ahrens et al. [1] noted that many different priority schemes could be used for prioritization of streaming data. ADR visualization, by providing a framework for defining importance and priority, is a generalization of the prioritization methods used in that work. In their work, they primarily used spatial closeness to the viewing position for the selection of streaming data. We can provide the same camera position prioritization, through selection and ordering of partitions. We sort the leaves of the priority tree by the distance from a viewing position, and serve them in that order. This assumes the data has been partitioned into regions of interest. An alternative way of doing this is to directly incorporate distance as the priority measurement, and serve partitions based on depth ranking.

4 Computational Performance

We have written a parallelized prototype of ADR visualization, and tested it with two different data sets, shown in various figures throughout the paper. Since the intent is to work *in situ*, we performed scaling studies on the priority tree construction to show that it is efficient and can be run in conjunction with a simulation. One data set is ocean climate simulation data from the Parallel Ocean Program (POP), a tenth of a degree, high-resolution eddy resolving simulation. The native simulation grid is a structured resolution of 3600 x 2400 x 42 single floating-point data, which is roughly 1.4GB per variable. Four variables were used in our studies, with 365 simulated time steps.

In our implementation we use a hybrid top-down and bottom-up strategy, where partitioning of time and variables is performed top-down, to "trim" the priority tree. Then, we apply the bottom-up merging approach to create the spatial partitions. This hybrid method works well with how parallel simulations are set up to run, which increase incrementally over time and and are distributed in space over processes. In this case, we scan the data three times, once per time partitioning (globally once, but incrementally over time), once per variable partitioning, and once for spatial partitioning.

In Figure 14, strong scaling of the top-down and bottom-up priority tree construction for spatial partitioning is shown. The bottom-up construction runs very quickly, approximately in 1 to 2 seconds, even at low processor count, due to only needing to calculate measurements once. The bottom-up reduction allows us to save computational time, as the measurement is aggregated (reduced) from leaves in the priority tree. On the other hand, the top-down construction is slow in comparison, approximately 100 seconds, due to several drawbacks: 1) it has to repartition the data at every level of the tree 2) it has to communicate more often, performing much more data movement between processors, and 3) the importance measurement is recalculated, at every level of the tree.

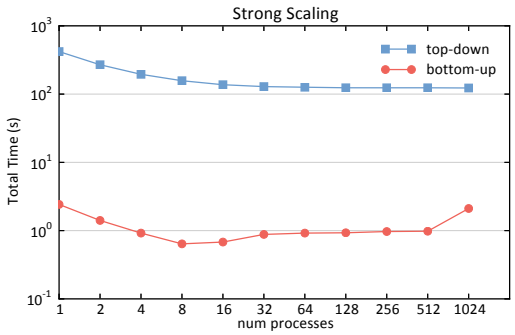


Figure 14: Strong scaling of the spatial partitioning with one time step of the ocean climate simulation. Both top-down and bottom-up methods are shown. The importance measurement and criteria are information entropy.

In Figure 15, we show the weak scaling of constructing the priority tree using top-down and bottom-up methods. Here, the effects of communication and repartitioning is shown between multiple processors in the top-down construction. In the top-down construction, as the number of processors double, the amount of data which needs to be moved between processors also doubles. This is because the data set needs to be repartitioned between processors at every level of the tree. On the other hand, the bottom-up construction has a small amount of data communicated at each step, and only between pairs, due to pair-wise tree reduction. Top-down in the worst case will move $nlogn$ data per time step while the bottom-up will only move $plogp$ data per time step, where $p \ll n$, p is the number of processors, and n is the number of data points per time step.

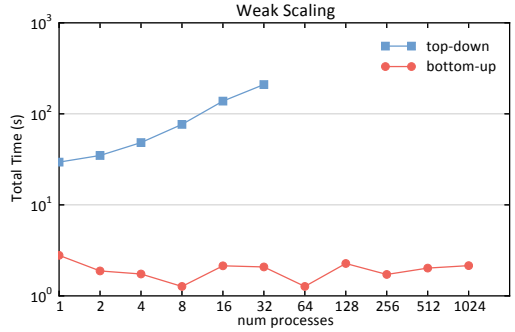


Figure 15: Weak scaling of the spatial partitioning with one time step of the ocean climate simulation. Both top-down and bottom-up methods are shown. The importance measurement used is information entropy. Note that top-down weak scaling cannot be run after a certain processor count due to lack of memory.

5 Conclusion and Future Work

We have described our prioritization framework for large-scale data triage and focus+context visualization. It provides an analysis-imposed structure on top of scientific data, independent of the source grid. The prioritization tree can then be used for selection of data in resource constrained situations. Embedded movies of time selection and camera selection on an exploding asteroid simulation can be seen in Figures 16, 17, 18, and 19.

The primary future work required for this research is that the precise amount of data resulting from the prioritization can not be known *a priori*. The amount of triage is controlled by importance criteria, but it does not directly control the amount of data that will pass the priority test. This is due to the time-series nature of simulations, and it is impossible to predict the future amount of data that may pass the importance criteria. One possible way to circumvent this problem is that it may be viable to estimate the amount of data triaged based on past importance criteria and simulation parameters.

Acknowledgments

This work was supported by the Department of Energy (DOE) Office of Science (SC) Advanced Simulation and Computing Research (ASCR).

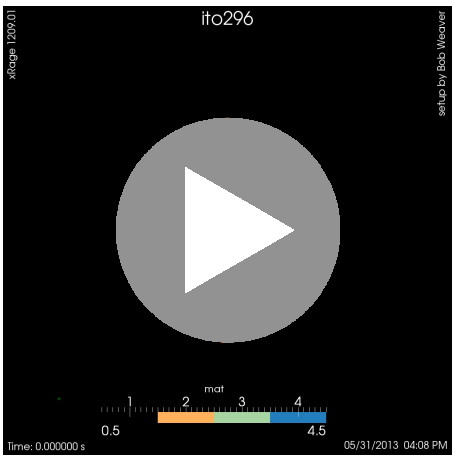


Figure 16: Movie of exploding asteroid simulation with no selection methods.

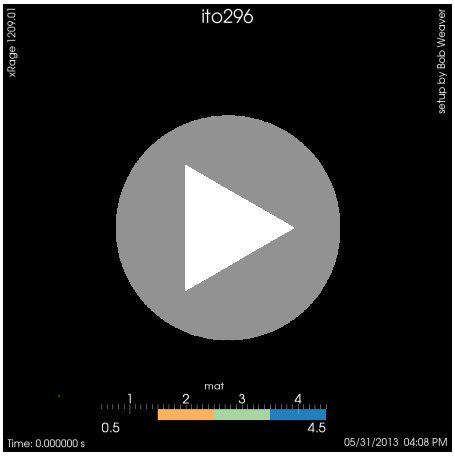


Figure 17: Movie of exploding asteroid simulation with time selection.

References

[1] J. P. Ahrens, N. Desai, P. S. McCormick, K. Martin, and J. Woodring. A modular extensible visualization system architecture for culled prioritized data streaming. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6495 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, Jan. 2007.

[2] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989.

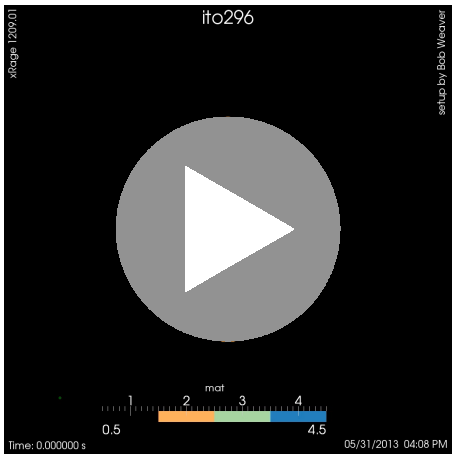


Figure 18: Movie of exploding asteroid simulation with camera selection.

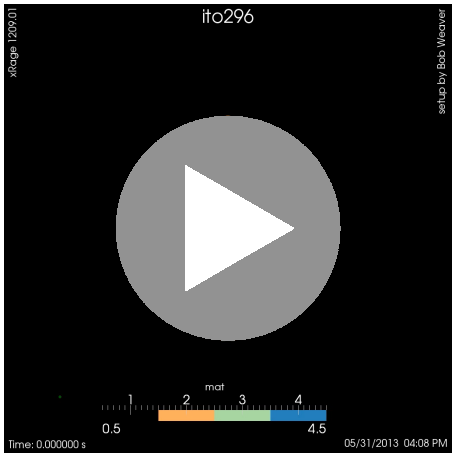


Figure 19: Movie of exploding asteroid simulation with time and camera selection methods.

[3] M. J. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53:484–512, Mar. 1984.

[4] A. Biswas, S. Dutta, H.-W. Shen, and J. Woodring. An information-aware framework for exploring multivariate data sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2683–2692, Dec. 2013.

[5] U. Bordoloi and H.-W. Shen. View selection for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 487–494, Oct 2005.

[6] R. J. Brunner, I. Csabai, A. Szalay, A. J. Connolly, G. P. Szokoly, and K. Ramaiyer. The Science Archive for the Sloan Digital Sky Survey. In G. H. Jacoby and J. Barnes, editors, *Astronomical Data Analysis Software and Systems V*, volume 101 of *Astronomical Society of the Pacific Conference Series*, page 493, 1996.

[7] Z. Fang, T. Moeller, G. Hamarneh, and A. Celler. Visualization and exploration of time-varying medical image data sets. In *Proceedings of Graphics Interface 2007*, pages 281–288. ACM, 2007.

[8] B. Fryxell, K. Olson, P. Ricker, F. Timmes, M. Zingale, D. Lamb, P. MacNeice, R. Rosner, J. Truran, and H. Tufo. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131(1):273, 2000.

[9] J. Graham, E. Givelberg, and K. Kanov. Run-time creation of the turbulent channel flow database by an hpc simulation using mpi-db. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 151–156, New York, NY, USA, 2013. ACM.

[10] G. Ji and H.-W. Shen. Feature tracking using earth movers distance and global optimization. In *Pacific Graphics*. Citeseer, 2006.

[11] T.-Y. Lee, O. Mishchenko, H.-W. Shen, and R. Crawfis. View point evaluation and streamline filtering for flow visualization. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 83–90, March 2011.

[12] R. Löhner. An adaptive finite element scheme for transient problems in cfd. *Comput. Methods Appl. Mech. Eng.*, 61(3):323–338, Apr. 1987.

[13] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 457–462, New York, NY, USA, 2004. ACM.

[14] A. Mignone, G. Bodo, S. Massaglia, T. Matsakos, O. Tesileanu, C. Zanni, and A. Ferrari. Pluto: A numerical code for computational astrophysics. *The Astrophysical Journal Supplement Series*, 170(1):228, 2007.

[15] K. Muehler, M. Neugebauer, C. Tietjen, and B. Preim. Viewpoint Selection for Intervention Planning . pages 267–274, Norrköping, Sweden, 2007. Eurographics Association.

[16] S. Popinet. Gerris: A tree-based adaptive solver for the incompressible euler equations in complex geometries. *J. Comp. Phys*, 190:572–600, 2003.

[17] F. Post, T. van Walsum, F. Post, and D. Silver. Iconic techniques for feature visualization. In *Visualization, 1995. Visualization '95. Proceedings., IEEE Conference on*, pages 288–295, 464, Oct 1995.

[18] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. In *Computer Graphics Forum*, volume 22, pages 775–792. Wiley Online Library, 2003.

[19] D. Silver and N. J. Zabusky. Quantifying visualizations for reduced modeling in nonlinear science: Extracting structures from data sets. *Journal of visual communication and image representation*, 4(1):46–61, 1993.

[20] M. Steinmetz. Grapesph: cosmological smoothed particle hydrodynamics simulations with the special-purpose hardware grape. *Monthly Notices of the Royal Astronomical Society*, 278(4):1005–1017, 1996.

[21] R. Teyssier. Cosmological hydrodynamics with adaptive mesh refinement: a new high resolution code called ramses. *Astron.Astrophys.*, 385:337–364, 2002.

[22] X. Tong, T.-Y. Lee, and H.-W. Shen. Salient time steps selection from large scale time-varying data sets with dynamic time warping. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 49–56, Oct 2012.

[23] F.-Y. Tzeng and K.-L. Ma. Intelligent feature extraction and tracking for visualizing large-scale 4d flow simulations. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 6. IEEE Computer Society, 2005.

[24] P.-P. Vzquez, M. Feixas, M. Sbert, and W. Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modelling. *Computer Graphics Forum*, 22(4):689–700, 2003.

[25] C. Wang, A. Garcia, and H.-W. Shen. Interactive level-of-detail selection using image-based quality metric for large volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):122–134, 2007.

[26] C. Wang and H.-W. Shen. Lod map - a visual interface for navigating multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1029–1036, 2006.

[27] C. Wang, H. Yu, and K.-L. Ma. Importance-driven time-varying data visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1547–1554, Nov 2008.

[28] S. Williams, M. Hecht, M. Petersen, R. Strelitz, M. Maltrud, J. P. Ahrens, M. Hlawitschka, and B. Hamann. Visualization and analysis of eddies in a global ocean simulation. *Comput. Graph. Forum*, 30(3):991–1000, 2011.

[29] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, and K. Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. EuroVis '11, May 2011.