Query-Driven Visualization of Time-Varying Adaptive Mesh Refinement Data

Luke J. Gosink, *Student Member, IEEE*, John C. Anderson, *Student Member, IEEE*, E. Wes Bethel, *Member, IEEE*, and Kenneth I. Joy, *Member, IEEE*

Abstract—The visualization and analysis of AMR-based simulations is integral to the process of obtaining new insight in scientific research. We present a new method for performing query-driven visualization and analysis on AMR data, with specific emphasis on time-varying AMR data. Our work introduces a new method that directly addresses the dynamic spatial and temporal properties of AMR grids that challenge many existing visualization techniques. Further, we present the first implementation of query-driven visualization on the GPU that uses a GPU-based indexing structure to both answer queries and efficiently utilize GPU memory. We apply our method to two different science domains to demonstrate its broad applicability.

Index Terms—AMR, Query-Driven Visualization, Multitemporal Visualization

1 Introduction

Computational simulation has become an essential and powerful tool impacting a diverse group of scientific disciplines such as engineering, biology, and medicine. Detailed simulations that model time-dependent, continuous physical phenomena, along with analysis and visualization tools that address the temporal aspects of these simulations, are essential to generate new understanding and insight into many domain-specific problems. Approaches for visualizing time-varying data are generally based on either temporally sequential, or temporally concurrent analysis methods. In the former, renderings are first generated from individual timesteps by using traditional visualization approaches (e.g. isosurface extraction or volume rendering). These renderings are then viewed sequentially as an animation. In contrast, temporally concurrent visualization methods (i.e. multitemporal visualizations) present the important features from multiple timesteps in a *single* image.

In scientific simulations, the immense size and sheer complexity of data generated from highly-detailed numerical methods has popularized the use of adaptive mesh refinement (AMR) strategies. In numerical simulations, AMR-based techniques adaptively refine the domain space of a simulation, both spatially and temporally, into a hierarchy of nested, sequentially refined grids. Though these strategies are computationally efficient and provide significant storage benefits, the dynamic aspects of the grid hierarchies pose significant challenges for visualization methods. Specifically, each timestep in a simulation contains a unique grid hierarchy, consisting of multiple levels of grid cell refinement. When considering a fixed spatial location in the computational domain at two or more timesteps, the disparity of grid cell refinement that occurs between the grid hierarchies at this location prevents the simultaneous evaluation of data necessary for many visualization algorithms.

In this work, we address the challenges of using a query-driven visualization (QDV) approach to visualize time-varying AMR data. QDV methods allow users to process ad-hoc queries over large-scale datasets and visualize the spatial regions where data satisfies the queries. QDV methods are well-suited for analyzing and visualizing datasets that are both large and highly complex [26].

We present a two-step method for compositing and synchronizing AMR data from a series of timesteps. We first generate a composite template from the AMR grid hierarchies of these timesteps; the composite template preserves the finest level of grid cell refinement from each grid hierarchy. We then synchronize each timestep's grid hierarchy to the composite template. This approach enables our method to process queries on a common AMR grid hierarchy. Using this data structure, we move the work of query processing to the GPU to realize the benefit of greatly accelerated QDV analysis. On the GPU side, we integrate our new method with a GPU-based query engine, called the Bin-Hash index [10].

The main contributions of this work are the following.

- We develop a new framework for doing QDV processing and visualization of time-varying AMR data. The core of this method is based upon a synchronization strategy that addresses the disparities in spatial refinement that exist between any series of timesteps in an AMR-based simulation.
- We demonstrate the first GPU-based QDV approach that utilizes a GPU-based indexing strategy to accelerate query processing, efficiently utilize GPU memory, and accelerate QDV methods.

In the next section, we discuss work germane to our efforts. This is followed in Section 3 by an overview of AMR grid fundamentals, our composite template construction and timestep synchronization process, and an introduction to the Bin-Hash index. Finally, we present the results of our method from both a qualitative and quantitative analysis perspective.

2 Previous Work

To provide a new method for analyzing and visualizing time-varying adaptive mesh refinement data, our work builds upon three separate fields: AMR visualization, query-driven visualization (QDV), and time-dependent visualization methods.

2.1 Visualization of Adaptive Mesh Refinement Data

Adaptive mesh refinement (AMR) strategies are based on the observation that localized complexity in physical phenomena – i.e. the rate of change observed in physical quantities within small regions in the domain – often varies substantially over space and time. Utilizing this observation, AMR strategies proceed by simulating these physical phenomena adaptively. Rather than utilizing a costly uniform grid of high density for the *entire* domain space, AMR techniques begin with a relatively course (and thus cheaper) grid hierarchy and adaptively refine grids in this hierarchy *only* in regions of the domain requiring higher levels of accuracy.

Luke J. Gosink, John C. Anderson, and Kenneth I. Joy are with the Institute for Data Analysis and Visualization (IDAV) at the University of California, Davis. E-mail: ljgosink, janderson, kijoy@ucdavis.edu.

E. Wes Bethel is with the Scientific Visualization Group at Lawrence Berkeley National Laboratory, E-mail: ewbethel@lbl.gov.

Manuscript received 31 March 2008; accepted 1 August 2008; posted online 19 October 2008; mailed on 13 October 2008.
For information on obtaining reprints of this article, please send e-mailto:tvcg@computer.org.

Importantly, this adaptive refinement occurs not just spatially, but temporally as well. As the simulation evolves, a regridding algorithm tests and refines grid cells with a frequency directly related to their level of refinement. Thus, grid cells of fine refinement – indicating regions of complex or important behavior – undergo testing for regridding more frequently than grid cells of comparatively coarser refinement. This adaptive spatial and temporal refining of the domain space results in a hierarchy of nested, sequentially refined grids that are computationally cheaper to construct and are less expensive to store than a high-density uniform grid.

Though AMR was first presented in 1984 [5], and then extended in 1989 [6], the challenges of mapping common visualization techniques to AMR's spatially dynamic grid structure were not addressed until much later. One of the earliest examples of AMR visualization was given by Max [20] in his cell-sorting method for volume rendering. Norman et al. [21] convert AMR hierarchies into finite-element hexahedral cells with cell centered data, thus enabling the use of standard visualization tools.

More recent work focuses upon operating directly on AMR data. Work by Ma [19] describes a parallel rendering strategy for AMR data and presents two contrasting visualization approaches. Weber et al. [27, 29] present software and hardware-accelerated methods based on cell projection that facilitate direct volume rendering of AMR data. In their work, they render an AMR hierarchy by starting with its coarsest representation. The image is then refined by subsequently integrating the results obtained from renderings of finer grids. Weber et al. [28] also present crack-free isosurface extraction methods for AMR data. Park et al. [23] present a hierarchical multi-resolution splatting technique for AMR data that utilizes kd-trees and octrees. Their novel approach provides interactive performance for modest sized data.

Kähler and Hege [14] present a hardware-accelerated volume-rendering approach to visualize AMR data. Their work, based on 3D textures, directly utilizes the hierarchical grid structure of the AMR data to rapidly render high-resolution datasets – including AMR data consisting of over nine levels of refinement. Kähler et al. also present a novel strategy for remotely visualizing AMR data at intermediate time steps [15]. Their method utilizes so-called "keyframe" timesteps to generate intermediate grid hierarchies. Data for these grid structures is then acquired through interpolation strategies (via Hermite or linear methods) by using the existing data in the keyframe timesteps.

Kähler et al. have more recently extended their earlier work by presenting a GPU-assisted raycasting strategy for accelerating the visualization of AMR data [16]. This work utilizes a kd-tree, resident on the GPU, to provide a view-consistent ordering of data, and accelerate the task of volume rendering. They contrast their method's results with a hardware accelerated slice-based volume rendering approach. Their method generates superior images to the slice-based approach, with no observable artifacts.

2.2 Query-Driven Visualization

Query-Driven Visualization (QDV) is an important and effective way to combine database and visualization technologies. QDV strategies are based on the observation that smaller subsets of data are usually the genesis of insight or breakthroughs to new trends [3, 11]. In QDV, users begin analysis by forming definitions for data that are "important" to them. This characterization consists of constructing range constraints for variables of interest. As an example, a user analyzing a combustion dataset may set constraints over specific variables such as: ($1100 < \textit{temperature} < 1800$) AND ($\textit{pressure} < 780$). QDV methods use these range constraints to filter data records passed to visualization and analysis software. This query filtration process focuses visualization and analytical resources exclusively on data that is meaningful to the user.

Stockinger et al. [25] were first to present the notion of coupling visualization with high performance query technology. Their work demonstrates that the computational complexity of visualization processing can be constrained to the number of items returned by a query. Their approach introduces a software system (DEX) that utilizes a highly efficient indexing and query infrastructure, called FastBit, to rapidly identify records of interest [32, 33].

Gosink et al. [9] extend the utility of QDV methods by using correlation fields to explore variable interactions within the domain space of query-regions. Their work focuses on characterizing flame-front regions in combustion simulations.

Bethel et al. apply QDV principles to network traffic analysis [7]. They use compressed bitmap indices to visualize and characterize over 2.5 billion records of network connection data. Stockinger et al. [24] extend the QDV approach for traffic analysis by presenting a family of new parallel algorithms that generate queryable two-dimensional conditional histograms. These conditional histograms are used to detect and characterize distributed scans.

2.3 Multitemporal Visualization

Though researchers have proposed various methods to track time-varying features across multiple timesteps in sequential fashion (i.e. animations), less attention has been paid to the direct visualization of 4D data. In direct visualization of 4D data, or multitemporal visualization, important features from selected timesteps are conveyed in a *single* meaningful visualization. Projecting four-dimensional information meaningfully onto a two-dimensional image is difficult to do without saturating the visualization with too much information. Finding ways to extend traditional, three-dimensional visualization methods to four dimensions is one intuitive way to approach visualizing time-varying data.

Hansen et al. [12,13] use 3D scalar fields as elevation maps in 4D. In these works, 4D lighting, shading, and plane-tracing (i.e. 4D ray tracing) are used to visualize higher dimensional data. Bajaj et al. [2] extend object splatting techniques to present a generalized hyper-volume splatting approach. Their method presents a multi-resolution algorithm for providing insightful visualizations of scalar fields in any dimension; the focus of their work, however, is not *explicit* temporal feature tracking. Bhaniramka et al. [8] extend and generalize marching methods to higher dimensions, specifically generating 4D isosurfaces that can be sliced to enable the study of time-evolving features. Woodring et al. [31] also extend slicing techniques for the direct rendering of 4D space-time volumes (as apposed to the 4D isosurfaces generated by Bhaniramka's work). Their hyper-projection method displays unique spatiotemporal features. Woodring and Shen [30] present a way to directly volume render time-varying data in a single multitemporal image. In their work, they orthographically project four-dimensional data (i.e. volume data accumulated over time) onto a three dimensional image plane. They use traditional rendering methods over the image plane, using opacity values that are spatially and temporally based, to realize multitemporal images.

Extending AMR and QDV Work

To date, no techniques exist that facilitate the rendering of time-varying AMR data in a multitemporal fashion. We address this important issue in this work by combining GPU-based QDV methods with a new approach for temporally synchronizing the time-varying data from a series of AMR timesteps. The results of our method allow for the interactive visualization of multivariate, spatiotemporal AMR data. We also extend previous QDV work in two aspects: we present the first application of QDV techniques on AMR data, and we also present the first GPU-based QDV approach that utilizes a GPU-based indexing strategy to accelerate query processing, efficiently utilize GPU memory, and accelerate QDV methods.

3 Method

Query-driven analysis and multitemporal visualization of AMR data are hindered by the dynamic temporal and spatial properties of AMR-based simulations. Specifically, in AMR-based simulations, any fixed spatial location in the domain can be covered by a grid cell of varying refinement based upon the timestep analyzed. For query-driven visualization (QDV), this disparity in refinement between timesteps prevents the evaluation of multitemporal queries. Similarly, coherent multitemporal renderings of AMR data from any visualization approach – extracting and simultaneously rendering isosurfaces from multiple timesteps, or volume rendering with time-dependent transfer functions – also require addressing these temporal-based disparities of spatial refinement.

Our new method addresses these challenges by synchronizing all AMR grid hierarchies (from any subset of timesteps) with a composite template. This synchronization process facilitates query-driven analysis and multitemporal visualization in two aspects: temporally sequential visualizations, where features from these timesteps are analyzed in sequential frames as a movie, and temporally concurrent visualizations where a single multitemporal image conveys the important features from all synchronized timesteps.

We base our method on the AMR grid hierarchy outlined by Berger and Colella [5,6]. We outline this hierarchy and its properties next. For more details regarding AMR-based simulations, we refer the reader to [1,5,6].

3.1 AMR Grid Structure

Adaptive mesh refinement (AMR) implementations are traditionally based on a nested hierarchy of successively refined axis-aligned grids. These grids are identified using the notation $G_{l,k}$ where l indicates the level of cell refinement for the grid, and k is the unique number for the grid given this refinement level [6]. We use the notation $G_{l,k\rightarrow k+n}$ to refer to a continuous set of grids at a single level of refinement. The increase in resolution from one grid refinement level to the next is specified by a refinement ratio ”r”, which indicates how many grid cells of level $l+1$ fit into a single grid cell of level l (considering a single axis-direction).

$$\Delta x^{l+1} = \tfrac{1}{r}\Delta x^{l}, \quad \Delta y^{l+1} = \tfrac{1}{r}\Delta y^{l}, \quad \Delta z^{l+1} = \tfrac{1}{r}\Delta z^{l} \qquad (1)$$

Note that the refinement ratio may change between successive grid levels. For example, the refinement ratio may be two between levels l and $l+1$, and four between levels $l+1$ and $l+2$. Though not required, refinement ratios are usually based on powers of two [4]. This convention seems to reflect a good balance between coding simplicity and an effective realization of the benefits of refinement.

An additional property required of all grids is the notion of *proper nesting*. This nesting property is strictly defined in the sense that grid cells of refinement level l are prohibited from abutting any grid cell other those of refinement level l, $l+1$, or $l-1$. A simple 2D example demonstrating an AMR hierarchy is illustrated in Figure 1.

At the start of a simulation, $t = 0$, the initial AMR grid hierarchy contains a single grid composed of cells of the coarsest level of refinement. Before the simulation begins, the grid cells of this initial hierarchy are refined based upon a convergence/stability criteria specified by the user. This refinement criteria, utilized both at the start and during the simulation process, may be based on the behavior of flow features (e.g. vorticity or density gradients) [1], or on factors that are more complex [6]. This initial refinement process is iterative – testing and refining are repeated until for all grid cells at all levels of refinement either the convergence criteria is met, or the finest allowed level of refinement is reached (maximum refinement levels are user set).

Given this refinement procedure, note that regions can be covered by multiple grids: e.g. a spatial location covered by $G_{2,a}$, will also

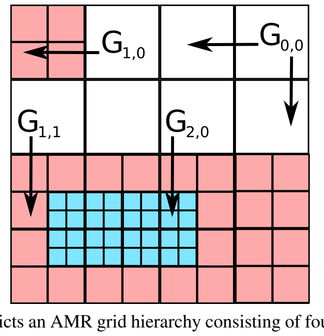


Fig. 1: This image depicts an AMR grid hierarchy consisting of four grids and three levels of refinement: $G_{0,0}$, $G_{1,0\rightarrow 1}$, and $G_{2,0}$. Grid cells are refined with a refinement ratio of r=2 and are properly nested: grid cells at level 2 do not abut grid cells at level 0.

be covered by some $G_{1,b}$ as well as $G_{0,c}$. With each refinement level possessing a different set of data for the specific region, a visualization method can take one of several approaches to utilize this data [18]:

- Treat all the grids (and their values) independently;
- Combine the data together in some way that is physically meaningful and use the result for visualization; or
- Use the data value(s) from the finest grid available and ignore data value(s) from coarser grids.

In our method we adopt the last approach to acquire data values from AMR grid hierarchies; by using the finest resolution source available at any given location in the domain, we are sure to be using the more accurate and detailed information produced by the computational model.

3.1.1 Advancing Grid Cells in Time

As the simulation advances beyond $t = 0$, a time-stepping algorithm evaluates and regrids grid cells according to their level of refinement: $G_{l,0\rightarrow n}$ are evaluated and regridded independently of $G_{l+1,0\rightarrow m}$ etc. The frequency of these regriddings is directed by the refinement ratio such that r defines both the spatial *and* temporal refinement properties that guide AMR-based simulations:

$$\Delta t^{l+1} = \frac{1}{r}\Delta t^{l} \qquad (2)$$

The time-stepping algorithm can be thought of as a recursive approach that advances grids cells in time according to their level of refinement [1]. To advance level l, $l_0 \leq l \leq l_{max}$, the following steps are performed:

1. Advance grid cells at level l in time by one timestep. Calculate data values for these grid cells at this new time. Additionally, if $l+1 \leq l_{max}$, assess all grid cells at this new time for the need of additional refinement (through the convergence criteria). For all cells that require further refinement, generate new grid cells at refinement level $l+1$ in these cell locations.
2. Advance level $l+1$ grid cells r times using Equation 2 to determine the length of the timestep. At each of the r timesteps, calculate data values for these grid cells. Additionally, if $l+2 \leq l_{max}$, assess all grid cells for the need of additional refinement (through the convergence criteria). For all cells that require further refinement, generate new grid cells at refinement level $l+2$ in these cell locations.
3. Synchronize data from grid cells at level $l+1$ back to level l.

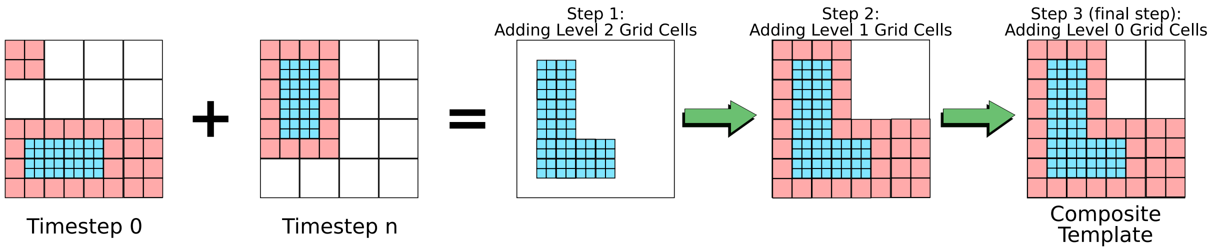
The synchronization of data in the last step involves several steps that effectively serve to propagate accuracy back to the coarser refined grid levels. In this way the accuracy of the data at coarser levels of refinement is corrected/adjusted with finer resolution data.

3.2 Composition and Synchronization of AMR Grids

To perform QDV on a series AMR timesteps, e.g. timestep 0 and timestep n, every spatial grid cell associated with a data record at timestep 0 must have a corresponding spatial grid cell of equivalent refinement at timestep n. We achieve this consistency of refinement by first constructing a composite template from the series of AMR timestep’s grid hierarchies. We then use this template to direct the synchronization of these grid hierarchies for purposes of QDV.

3.2.1 Composite Template Construction

The composite template construction process consists of a refinement-level ordered compositing of AMR grid cells. From the AMR grid hierarchies contained in a series of timesteps, the construction process begins by adding to the composite template only those grid cells whose refinement level is equal to l_{max}. Next, the construction process *conditionally* adds to the template those grid cells whose refinement level is equal to l_{max-1}. With conditional additions, the process adds a grid cell to the template ***only if*** a grid cell of finer refinement is not already



Timestep 0
+
Timestep n
=
Step 1:
Adding Level 2 Grid Cells
Step 2:
Adding Level 1 Grid Cells
Step 3 (final step):
Adding Level 0 Grid Cells
Composite Template

Fig. 2: This figure illustrates the sequential process of compositing the AMR grid hierarchies of two selected timesteps. The process begins by filling the composite template with all grid cells, from both timesteps, of the finest level of refinement. In each subsequent pass, our procedure adds grid cells of the next level of lesser refinement to the template - conditioned on the basis that a more finely refined grid cell has not already been placed at that position. Finally, we add grid cells of the coarsest level of refinement to the template.

resident in the template at this given location. This strategy continues until the process conditionally adds grid cells of refinement level equal to l_0. Figure 2 illustrates this construction process.

This refinement-level ordered compositing guarantees two fundamental properties in the final composite template:

- The template maintains the finest level of refinement from each timestep utilized in its construction – the template thus preserves the high fidelity data created by the numerical simulation.
- The composite template provides a basis for resolving the differences in grid cell refinement that exist when a given point in space is covered by different grid cell refinement levels at different points in time. Specifically, every grid cell from any AMR grid hierarchy used to construct the composite template can be mapped to a grid cell of equivalent refinement, or a group of nested grid cells of greater refinement, in the composite template.

3.2.2 Grid Synchronization

The composite template provides the common grid hierarchy necessary for performing query-driven analysis and multitemporal visualization of AMR data. The variable attributes (e.g. *pressure*, *density*, etc.) contained in each timestep's grid hierarchy must now be synchronized with this template. The second fundamental property of the composite template formulates this synchronization process.

Those grid cells (from all grid hierarchies used to generate the composite template) that map to regions of greater refinement in the composite template are synchronized through a regridding process. This regridding process iteratively divides the grid cell in question into a nested group of grid cells of increased refinement. This refinement continues iteratively until grid cells are identical in refinement and hierarchical ordering to the group of nested grid cells in the composite template. To complete the synchronization, we propagate the cell centered value of the original grid cell to the centers of the newly created grid cells. Figure 3 illustrates this process. With each timestep's grid hierarchy synchronized, multitemporal query-driven visualization of AMR data is now possible.

3.3 Query-Driven Visualization of Temporal AMR Data

The goal of query-driven analysis is to provide scientists with interactive and resource-efficient methods for visually exploring large multidimensional data. To meet these needs, it is important to process user's queries, and render the results generated from these queries, as fast as possible. We meet these needs by employing a GPU-based query indexing structure, called the Bin-Hash index [10]. By utilizing a GPU-based query engine, we can implement the entire QDV process on the GPU and accelerate QDV performance as a whole with the GPU's parallel processing power. In our implementation, the CPU serves as a host to the GPU, only streaming the minimal data necessary to perform full-resolution queries (Section 3.3.2). All queries are evaluated (and rendered) on the GPU by executing kernels written in NVIDIA's data-parallel programming language CUDA [22]. QDV in literature typically evaluates scalar data. However, the Bin-Hash index can also evaluate vector data, as well as evaluate an arbitrary number of timesteps or variables.

The integration of the Bin-Hash index into QDV is similar to previous integrations that utilized a CPU-based index [26]. Both strategies use index building, index searching, record processing, and rendering procedures. The difference between our work and previous work is that we query and render adaptively refined spatiotemporal data. This difference requires, in addition to the use of the composite template and synchronized grid hierarchies (Section 3.2), mapping query results to direct the rendering of grid cells during the rendering stage. We discuss Bin-Hash index building, searching, and rendering next.

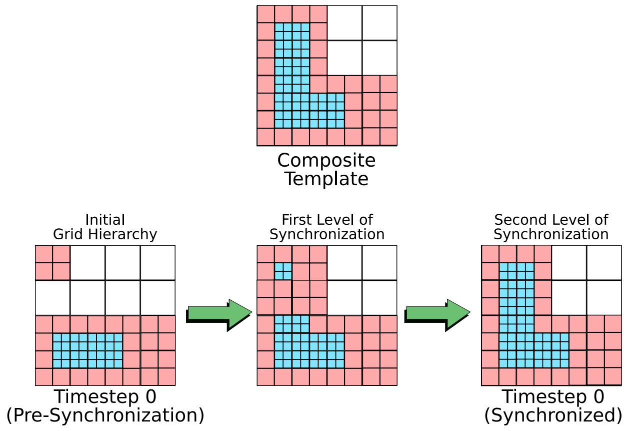
3.3.1 Bin-Hash Index Construction

The strategy of the Bin-Hash method is based upon the observation that query performance can be accelerated through the utilization of multi-resolution information. Supporting this approach requires two levels of informational representation for the AMR data records: full-resolution (the 32-bit base data) and low-resolution information (8-bit encoded data).

The Bin-hash index construction algorithm takes as input the full-resolution AMR data from a single timestep and generates both an encoded *and* spatially compacted version of this input. The index construction algorithm performs this operation in two stages. In the first stage, it utilizes a binning strategy to generate a binned (ı.e. encoded) version of the data. In the second stage it utilizes a combination of data partitioning with a technique referred to as spatial hashing [17] to compactly represent the full-resolution data contained in each bin previously created by the first stage's binning procedure.

The first stage in the index construction process begins by examining and binning – independently – the data from each selected timestep's hierarchy. For example, given a set of bin boundaries on a variable A, such as $(b_0, b_1, \ldots, b_n)$, each bin is defined to be the interval $(b_0 \leq A < b_1)$, $(b_1 \leq A < b_2)$, and so on. Bin-Hash binning always utilizes 256 bins, where each bin contains approximately the same number of records. The encoded version of the dataset, referred to as low-resoution data, is created by replacing each 32-bit full-resolution data value with its associated 8-bit bin number (0-255).

The second stage in the process of index construction requires the partitioning and spatial compaction of the original full-resolution data.



Composite Template
Initial Grid Hierarchy
First Level of Synchronization
Second Level of Synchronization
Timestep 0 (Pre-Synchronization)
Timestep 0 (Synchronized)

Fig. 3: This figure depicts the sequential process of synchronizing the grid hierarchy of a given timestep with a composite template. At each level of synchronization, grid cells conditionally refine themselves by one additional level according to whether or not they are synchronized with the composite template. In this example, synchronization is complete for the grid hierarchy in the second level of synchronization.

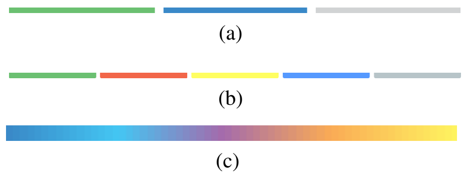


Fig. 4: These images depict the three transfer functions we employ in our work. In (a) and (b) the colors correspond to levels of AMR grid refinement for the respective Argon Bubble and Hurricane datasets: green colors indicate grid cells of finest refinement, and gray colors indicate grid cells of coarsest refinement. In (c) the colors are used to convey summary statistic information in multitemporal visualizations: blue colors indicate regions where few queries have selected a cell during QDV analysis, and yellow colors indicate regions where the most queries have selected the cell.

To perform this, records are first partitioned according to their bin numbers. Next, these subsets of data are spatially compacted through a technique called perfect spatial hashing [17]. Perfect spatial hashing takes all the full-resolution data associated with the records of a given bin, and stores it separately as two small tables: a hash table and an offset table. This operation is performed for all 256 bins. Once the second stage is completed, the total full-resolution dataset is now represented as 256 pairs of hash and offset tables. The total storage overhead for the indices is approximately 1.5 - 2.0 times the size of the original AMR data. This partitioning and spatial hashing of the data is essential to the search process as the next section details.

3.3.2 Bin-Hash Index Searching

Before query processing begins, low-resolution (i.e. encoded) data for all selected timesteps is first uploaded onto the GPU. Resolving a query then consists of two stages, both performed on the GPU. In the first stage the GPU-resident low-resolution data is evaluated in a low-resolution query. In certain cases this low-resolution information is insufficient and full-resolution data must be utilized. In the second stage, up to two pairs of hash and offset tables (per variable) are sent to the GPU to assist in evaluating a full-resolution query. The result of this two-stage index searching approach is a single bit-vector – a boolean array, with one entry per AMR data record, that indicates which records (grid cells) have passed and which have failed the query.

In the first stage of the index searching algorithm we first determine the boundaries of the query. Consider an example. Given a user's range constraints on a given variable and timestep, such as (*pressure* > 100), we determine the bin(s) (Section 3.3.1) that these constraints fall into. In this example, assume that the value "100" for pressure is contained in the value range captured by bin 17. Bin 17 is then defined to be a "boundary bin" for the query. Next, the search process evaluates a low-resolution query by accessing the low-resolution data on the GPU. These low-resolution data records are then characterized as passing (the given record's value is *greater than* 17), failing (the given record's value is *less than* 17), or in need of full-resolution data (the given record's value is *equal* to 17).

In the second stage of the index searching algorithm, those low-resolution records that were characterized in stage one as needing full-resolution data, now undergo a full-resolution query. In this full-resolution query, the hash and offset tables corresponding to the boundary bin(s) of the query are streamed to the GPU from the CPU. This data transfer constitutes a trivial impact on PCI-E bandwidth [10], as a maximum of $\frac{2}{256}$ the size of the original dataset is transfered over the bus (at most two boundary bins may exist per variable in a query out of the possible 256 bins). Each record whose low-resolution value corresponds to a boundary bin utilizes the hash and offset tables of this bin, via a perfect spatial hash, to access its original full-resolution value. Once this data has been retrieved, the searching algorithm performs a full-resolution query and all records are classified as passing or failing the query. In the case of a query that constrains more than one variable, the bit-vector solutions from each single variable are logically combined to form the multi-dimensional query's final solution.

3.3.3 Mapping AMR-Based Bit-Vectors to Three-Space

Our rendering algorithm takes the bit-vector solution created in the index search stage, and generates (for cells passing the query) renderable coordinates for dynamically-sized hexahedral cells in three-space. For uniform datasets, bit-vectors are rendered by mapping each record's unique index to a three-space position (through modular indexing), and rendering a constant sized hexahedral cell at this location. However, these rendering techniques won't work on query solutions generated from AMR data. AMR grids have dynamic cell sizes, and with arbitrary cell counts per dimension, modular indexing will not work.

Our approach to solving this problem is to generate a single record-ID list for each composite template that stores the spatial location and level of refinement of each grid cell in the template's hierarchy. The ordering of this list coincides with the ordering of the records being queried by the Bin-Hash method. Thus, the rendering stage in our implementation first accesses the bit-vector solution of the user's query. For each record that passes the query, the algorithm uses the record's index value to lookup the spatial location and level of refinement of that record. The appropriately sized hexahedral cell parameters are then written to a buffer in the GPU's global memory. The rendering algorithm additionally uses the cell's refinement level, and a color lookup table, to determine the color to render each grid cell; grid cells of a common refinement level share a common color. The memory is then mapped as a Vertex Buffer Object and rendered to the screen.

4 Visualization Applications and Analysis

We apply our new query-driven visualization (QDV) method to two datasets. We demonstrate our method's ability to generate multitemporal visualizations from time-varying AMR data, by visualizing summary statistic information generated from a single query that has been evaluated over multiple timesteps. In our analysis the term, "synchronized AMR data", refers to AMR data that has been synchronized with a composite template, "non-synchronized AMR data" refers to AMR data that has *not* been synchronized with a composite template.

All tests were performed on a desktop machine running the Windows XP operating system with SP2. All GPU kernels were run utilizing NVIDIA's CUDA software: drivers version 1.6.2, SDK version 1.1 and toolkit version 1.1. We additionally used the following hardware:

- *Motherboard*: EVGA 680i - 1066MHz FSB; 16X PCI-Express
- *Processor*: Intel QX6700 - 2.66GHz; 2 x 128KB L1; 2 x 4MB L2
- *Co-processor (Graphics Card)*: NVIDIA 8800GTX - 768MB GDDR3

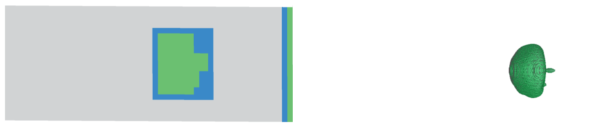
4.1 Dataset 1: Argon Bubble with Shock Wave

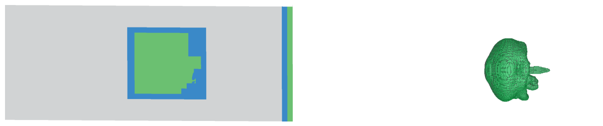
This dataset models a simulated shock wave passing through an argon bubble surrounded by atmospheric gases (i.e. air). One of the important characteristics of this dataset is the dispersion of the argon gas over time. There are over 1000 simulated timesteps in this dataset where the physical property we analyze is gas density; i.e. mass of gas per unit volume (ranging in values from 1.3 to 5.1). We analyze 18 AMR timesteps from these 1000: the grid hierarchies associated with times 100, 150, ..., and 950. Each timestep's (synchronized) hierarchy consists of three refinement levels and a total of 14 million cells.

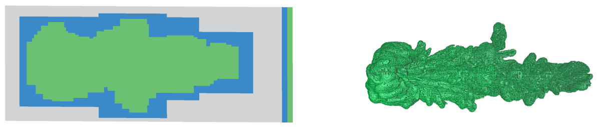
Multitemporal Visualization

The left column of images in Figures 5(a) through 5(c) depict two-dimensional slices of selected AMR grid hierarchies; these are the grid hierarchies we use to construct our composite template. The right column of images shows cells from these hierarchies that have been rendered through a process of query-driven analysis; all rendered grid cells in images from this column are selected by querying for density values: $(density \geq 1.5)$. In both columns, colors depict levels of grid refinement and are based on the transfer function shown in Figure 4(a). Specifically, gray regions indicate grid cells of coarsest refinement in the Argon Bubble simulation; and green regions indicate areas of finest refinement. Compositing the AMR grids from the 18 timesteps results in the composite template shown in Figure 5(d).

We use this composite template to generate a multitemporal visualization based on summary statistic information accumulated from


(a) Timestep 100


(b) Timestep 200


(c) Timestep 900

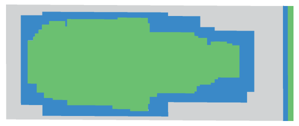

(d) Composite template

Fig. 5: This figure depicts images from select timesteps of the Argon Bubble dataset. In this dataset there are three grid hierarchy levels, shown in these images as colored gray (coarsest cell refinement), blue (medium cell refinement), and green (finest cell refinement). The left column of images, (a) - (c), show two-dimensional slices through the AMR grid hierarchies of these select timesteps. The image in figure (d) depicts the composite template we construct from all (18) timesteps we use in our analysis. The right column of images show the cells selected from query-analysis for the individual timesteps; in these images regions where gas density is greater than 1.5 are rendered.

queries that select regions of high gas density. We construct this visualization by first generating an integer-based solution array. This array contains one entry for each grid cell in the composite template that tracks how many times its respective grid cell passes a series of queries. We then query the synchronized AMR data of the first timestep (Timestep 100) for grid cells where $(density \geq 1.5)$ – this query delineates regions of higher gas density. Cells from this timestep that have passed the query increment the value corresponding to their position in the solution array by one. We apply the *same* query to the next timestep's synchronized AMR data (Timestep 150); results of this query too are added to the array. We repeat this process for all 18 timesteps. The final solution array contains summary statistic data that reveals how higher-levels of gas density disperse spatially over time.

We visualize these summary statistics in Figure 6. This figure is colored according to the transfer function shown in Figure 4(c). Cells that contain low gas density over the entire length of the simulation (i.e. no query from any timestep indicated the cell passed our query for high density) are shown in blue; in contrast, cells that contain higher gas density for the *entire* length of the simulation (i.e. every query over the timesteps indicated the cell passed our query for density) are indicated in yellow. From this multitemporal visualization we can see how higher-levels of gas density are distributed over space with respect to time. This type of visualization allows scientists to assess how effective various types of shock-waves are at dispersing chemical gases.

4.2 Dataset 2: Hurricane Isabel

This dataset was generated by a climate simulation that models a hurricane event over 48 timesteps. This dataset represents a common class of uniform resolution data consisting of a uniform grid of hexahedral cells (500x500x100). Such time dependent datasets can be costly to store and analyze. To ameliorate these costs, we recast this uniform data in a multi-resolution framework to ease storage and visualization demands. We recast the data by adaptively coarsening the flattened grid – in regions where detail is *not* required – into a multi-resolution grid framework that abides by the properties of an AMR hierarchy.

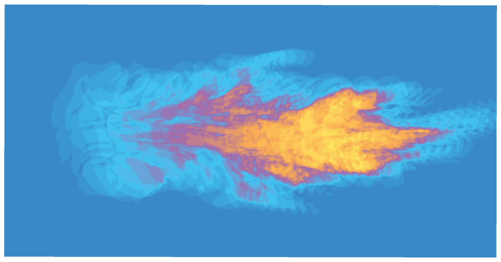


Fig. 6: This multitemporal image depicts summary statistic information gathered from queries processed over 18 select timesteps from the Argon Bubble dataset. In this image, yellow regions indicate areas where high gas density predominantly resides over the course of the Argon Bubble simulation. Light blue regions show areas where only a few timesteps indicate the presence of high gas density; these regions denote where argon is dispersing.

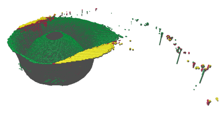
We define "important" regions in this process – that is, regions where coarsening should *not* take place – as areas where observed physical properties vary greatly. Those regions where observed physical properties do *not* vary are subjected to coarsening.

The results of this adaptive coarsening are a series of multi-resolution, time-dependent datasets that follow the structural properties of an AMR grid hierarchy. Each timestep in the original source data contains 25 million cells. After coarsening, each timestep contains 5 levels of refinement and a total of 8.6 million cells. Thus the criteria for adaptive coarsening results in a storage savings of about 65%, while still preserving the dataset's important features. We apply our method for generating multitemporal visualizations to these datasets; we generate a composite template from all timesteps, and then synchronize the timesteps with this template. One of the important characteristics in this dataset is the low-pressure regions that depict the location of the hurricane event. With our new multitemporal visualization method, we effectively characterize these regions. In this section, all queries for pressure are in Pascal units

Multitemporal Visualization

Figure 8 depicts select non-synchronized (top row) and synchronized (bottom row) timesteps from the hurricane dataset. The top row illustrates the individual grid hierarchies generated from our adaptive coarsening approach; the bottom row depicts the same grid hierarchies after synchronization with a composite template. The cells rendered in these images (both top and bottom rows) have passed a double-constraint query for pressure that selects cells from a given timestep that either contain low pressure OR high pressure: $(-200 \leq pressure \leq 20)$ OR $(500 \leq pressure \leq 1000)$. Note that we also show, to assist in interpreting the data in Figure 8's images, the regions of isolated low pressure (Figure 7(a)) and isolated high pressure (Figure 7(b)) .

The regions in Figure 8, as well as those in Figure 7(a) and Figure 7(b), are colored according to the transfer function in Figure 4(b); green regions indicate grid cells of finest refinement, and gray regions indicate grid cells of coarsest refinement. In both rows of images in Figure 8 observe that the low pressure regions, which characterize the important hurricane event, are preserved at the finest level of cell refinement. In contrast, regions of high pressure, which characterize areas where little observable variation in physical properties occur, are predominately coarsened by our adaptive coarsening method. In the bottom row of images in Figure 8, which show results for querying timesteps of composited and synchronized AMR grids, the path of the


(a) Cells selected from a query selecting regions of low pressure: $(-200 \leq pressure \leq 20)$

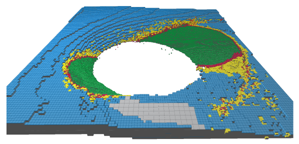
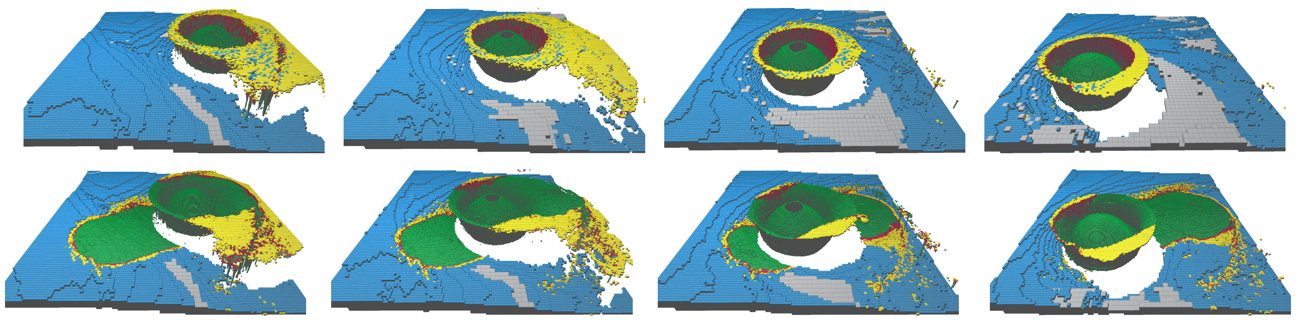

(b) Cells selected from a query selecting regions of high pressure: $(500 \leq pressure \leq 1000)$

Fig. 7: This figure depicts grid cells in the Hurricane dataset that contain relatively low (a) and high pressure (b).



(a) Timestep 10 (b) Timestep 25 (c) Timestep 35 (d) Timestep 45

Fig. 8: This series of images, selected from 48 timesteps, compares query results from non-sychronized (top row), and sychronized (bottom row) AMR grids of the Hurricane Isabel dataset. The query used on each timestep consists of two parts; we query for regions of low pressure $(-200 \le \textit{pressure} \le 20)$ OR regions of high pressure $(500 \le \textit{pressure} \le 1000)$. To assist in interpreting these images, the individual regions generated from these low and high pressure queries are shown in Figures 7(a) and 7(b).

hurricane is preserved at the finest level of refinement in the composite template as indicated by the green path.

We utilize this composite template to generate a multitemporal visualization based on summary statistics from queries – processed over each of the 48 timesteps – that select regions of low pressure. This process is analogous to the one performed in Section 4.1 for the Argon Bubble dataset. We begin by querying the synchronized AMR data of the first timestep (Timestep 1) for all cells with pressure values in the range of $(-200 \le \textit{pressure} \le 20)$ – note that this query characterizes regions of hurricane activity in the simulation. The results of this query are stored in an array. This process is repeated for all 48 timesteps; each query indicating in the array, those cells that have passed its query. The final results of the array contain summary statistics that indicate how the hurricane event, as characterized by our queries for low-pressure, evolves spatiotemporally.

We visualize these summary statistics in Figure 9. This figure is colored according to the transfer function shown in Figure 4(c). From this multitemporal visualization we can see how cells that predominantly contain low pressure over time – as indicated by regions of yellow – define the path of the hurricane.

Performance

There are two factors that contribute to the response time of a QDV application: the time it takes to process a query (Section 3.3.2), and the time it takes to render the query's solution (Section 3.3.3). To the user, these two times appear as a unified sum that we define as "query-driven response time". We analyze the performance of our work by presenting our query-driven response times in terms of the number of queries we can process and render in a single second. We additionally show this metric in terms of its two principle components: the time it takes to answer a query and the time to render the query's results.

Each timestep from the coarsened, synchronized Hurricane dataset (for the variable *pressure*) consists of 8.6 million cells. We evaluate our method's performance by independently analyzing two query characteristics important to QDV. We analyze QDV performance with respect to increasingly complex queries (i.e. the number of timesteps evaluated by the query), and QDV performance with respect to decreasingly selective queries (i.e. the percentage of cells that passed the query). The former query characteristic impacts the time it takes to process a query; the more timesteps in a query, the more time it takes to process the query. The latter impacts the time it takes to render the results of the query; queries with low selectivity select more cells that must be processed and rendered to the screen.

We begin our tests by querying a single timestep with queries that select 1%, 10%, and 20% of the cells as hits. As mentioned, we record for each of these queries the time it takes to answer a query, render the result of the query, as well as the total number queries we can process and render in a single second. We then consider an additional timestep in our queries, and repeat the same sequence of tests. We repeat this process until a total of five timesteps are simultaneously evaluated by all queries. The results of these tests are shown in Table 1.

The values shown in Table 1 indicate that performance for our QDV method is predominantly determined by the selectivity and not the complexity of a given query. Thus, users who analyze numerous variables or numerous timesteps in their queries, so long as the selectivity of these queries is high, will experience excellent performance with our method. Users whose queries are not very selective (i.e. select a large number of cells to render), even if they are only analyzing one variable, will experience slower performance. Note that even at the lowest level of performance (five timesteps at 20% selectivity in Table 1), our method is still operating above performance levels considered interactive (typically, any implementation that functions in excess of 6 Hz is considered interactive), and is providing excellent performance for QDV functionality.

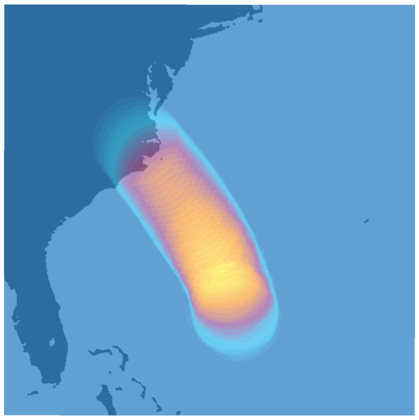


Fig. 9: This multitemporal image depicts summary statistic information gathered from queries processed over 48 timesteps from the Hurricane dataset. In this image, yellow regions indicate where low pressure predominantly exists across the 48 timesteps.

5 Conclusion and Future Work

We have presented a new method for performing query-driven visualization of time-varying AMR data. With our new analysis and visualization approach, we are able to construct multitemporal visualizations that convey in a single image how queries characterizing important interactions or properties evolve over time. We have demonstrated the extensible utility of our method by applying it to two different science

Timesteps Queried	*1% selectivity* (ms / ms / qps)	*10% selectivity* (ms / ms / qps)	*20% selectivity* (ms / ms / qps)
1 (8.6 million cells)	1.9 / 40.1 / 23.3	1.9 / 59.1 / 16.1	1.9 / 74.0 / 13.0
2 (17.2 million cells)	3.1 / 41.7 / 22.0	4.0 / 59.4 / 15.5	4.0 / 74.8 / 12.5
3 (25.8 million cells)	5.0 / 41.9 / 20.9	5.86 / 60.1 / 14.8	6.0 / 75.9 / 12.0
4 (34.4 million cells)	7.0 / 42.0 / 19.9	7.0 / 61.8 / 14.3	8.0 / 76.9 / 11.6
5 (43 million cells)	8.1 / 43.8 / 18.9	10.4 / 60.5 / 14.0	12.6 / 77.2 / 11.0

Table 1: This table depicts, for an increasing number of timesteps and records queried, performance times taken during the analysis of the Hurricane dataset (Section 4.2). The results are given according to three ranges for query selectivity: queries where 1%, 10%, and 20% of the available records are selected by the query. The first value in any given column and row entry is the time to answer the query, the second number is the time to render the query's solution; both of these values are given in milliseconds. The final number is the total number of queries processed and rendered per second; this measurement depicts the total performance experienced by the end user and incapsulates the two previous times.

domains, and showing how more traditional "flattened" time-varying datasets may be recast as AMR and evaluated by our approach.

One potential limitation of our new method is the possibility that a large number of compositing steps could result in a composite template that becomes a representation of the entire domain at the finest level of refinement. One observation to make in this worst-case scenario is that a fully refined composite template is not the expensive factor with respect to storage concerns. The concerns for storage arise from the synchronization of AMR timesteps *with* this template.

One approach we are pursuing to address this worst-case scenario is to develop methods that optimally select the timesteps synchronized in our analysis process, e.g., select the timesteps that minimize storage costs while maximizing information obtained. Such optimal selections need to be based upon the statistics of the AMR simulation itself: the temporal and spatial distribution pattern of fine-refinement cell counts, the rate at which these regions grow and diminish, etc. A second strategy we are pursuing is to utilize multiple composite templates, where each template is based upon a unique time interval. Timesteps are then synchronized to their local template; synchronizing timesteps to a small and local temporal range should reduce storage demands. Queries are evaluated over these individual templates, and the results are then combined using a template synchronization protocol. Both these strategies form the basis of our future work.

Another potential limitation for this work comes from memory constraints, imposed by the GPU, that limit the amount of AMR data able to be processed by our method. Our current work is focused on developing an application, based on the Bin-Hash index, that utilizes a grid of GPUs that will alleviate this limitation. In this application, large datasets will be partitioned and analyzed in parallel across the GPU grid, and independent solutions will be composited for final viewing.

Acknowledgements

This work was supported by the Lawrence Berkeley and Lawrence Livermore National Laboratories, and by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET). We would like to thank colleagues in the Institute for Data Analysis and Visualization (IDAV) at UC Davis for their support during the course of this work. Additionally, the authors thank Bill Kuo, Wei Wang, Cindy Bruyere, Tim Scheitlin, and Don Middleton of the U.S. National Center for Atmospheric Research (NCAR), and the U.S. National Science Foundation (NSF) for providing the Weather Research and Forecasting (WRF) Model simulation data of Hurricane Isabel. Data for the Argon simulation was provided by John Bell and Marc Day, Center for Computational Science and Engineering, Lawrence Berkeley National Laboratory. We also thank our reviewers for helping to improve this paper.

References

[1] A. Almgren, J. Bell, P. Colella, L. Howell, and M. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *J. Comput. Phys.*, 142(1):1–46, 1998.

[2] C. L. Bajaj, V. Pascucci, G. Rabbiolo, and D. Schikorc. Hypervolume visualization: a challenge in simplicity. In *Proc. of IEEE Symposium on Volume Visualization*, pages 95–102, 1998.

[3] J. Becla and D. L. Wang. Lessons learned from managing a petabyte. In *Conference on Innovative Data Systems Research*, pages 70–83, 2005.

[4] J. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. A. Zingale. Adaptive low Mach number simulations of nuclear flame microphysics. *J. Comput. Phys.*, 195(2):677–694, 2004.

[5] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53(3):484–512, Mar. 1984.

[6] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82(1):64–84, May 1989.

[7] E. W. Bethel, S. Campbell, E. Dart, K. Stockinger, and K. Wu. Accelerating Network Traffic Analysis Using Query-Driven Visualization. In *Proc. of IEEE Symposium on Visual Analytics Science and Technology*, pages 115–122, Oct. 2006.

[8] P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in higher dimensions. In *Proc. of IEEE Visualization*, pages 267–273, 2000.

[9] L. Gosink, J. C. Anderson, E. W. Bethel, and K. I. Joy. Variable interactions in query driven visualization. In *IEEE Trans. on Visualization and Computer Graphics*, volume 13, pages 1400–1407, Nov. 2007.

[10] L. Gosink, K. Wu, E. W. Bethel, J. Owens, and K. Joy. Bin-Hash Indexing: A Parallel Method for Fast Query Processing. Technical Report LBNL-729E, Lawrence Berkeley National Laboratory, 2008. http://www.vis.lbl.gov/Publications/2008/LBNL-729E.pdf.

[11] J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. S. Szalay, D. J. DeWitt, and G. Heber. Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41, 2005.

[12] A. J. Hanson and R. A. Cross. Interactive visualization methods for four dimensions. In *Proc. of IEEE Visualization*, pages 196–203, 1993.

[13] A. J. Hanson and P. A. Heng. Illuminating the fourth dimension. *IEEE Comput. Graph. Appl.*, 12(4):54–62, 1992.

[14] R. Kähler and H.-C. Hege. Texture-based volume rendering of adaptive mesh refinement data. *The Visual Computer*, 18(8):481–492, 2002.

[15] R. Kähler, S. Prohaska, A. Hutanu, and H.-C. Hege. Visualization of time-dependent remote adaptive mesh refinement data. In *IEEE Visualization*, page 23, 2005.

[16] R. Kähler, J. Wise, T. Abel, and H.-C. Hege. GPU-assisted raycasting for cosmological adaptive mesh refinement simulations. In *Proc. of Volume Graphics: Eurographics Assoc.*, 2006.

[17] S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Trans. on Graphics*, 25(3):579–588, 2006.

[18] T. J. Ligocki, B. V. Straalen, J. M. Shalf, G. H. Weber, and B. Hamann. *A Framework for Visualizing Hierarchical Computations*, pages 197–204. Jan. 2003.

[19] K.-L. Ma. Parallel Rendering of 3D AMR Data on the SGI/Cray T3E. In *Proc. of the Symposium on the Frontiers of Massively Parallel Computation*, pages 138–145, Feb. 1999.

[20] N. L. Max. Sorting for polyhedron compositing. In *Focus on Scientific Visualization*, pages 259–268, 1993.

[21] M. L. Norman, J. Shalf, S. Levy, and G. Daues. Diving deep: Data-management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science and Eng.*, 1(4):36–47, 1999.

[22] NVIDIA Corporation. NVIDIA CUDA compute unified device architecture programming guide. `http://developer.nvidia.com/cuda`, Jan. 2007.

[23] S. Park, C. L. Bajaj, and V. Siddavanahalli. Case study: interactive rendering of adaptive mesh refinement data. In *Proc. IEEE Visualization*, pages 521–524, 2002.

[24] K. Stockinger, E. W. Bethel, S. Campbell, E. Dart, and K. Wu. Imaging and visual analysis - detecting distributed scans using high-performance query-driven visualization. In *Supercomputing*, page 82, 2006.

[25] K. Stockinger, J. Shalf, E. W. Bethel, and K. Wu. Dex: Increasing the capability of scientific data analysis pipelines by using efficient bitmap indices to accelerate scientific visualization. In *Scientific and Statistical Database Management*, pages 35–44, 2005.

[26] K. Stockinger, J. Shalf, K. Wu, and E. W. Bethel. Query-driven visualization of large data sets. In *Proc. of IEEE Visualization*, pages 167–174, 2005.

[27] G. H. Weber, H. Hagen, B. Hamann, K. I. Joy, T. J. Ligocki, K.-L. Ma, and J. M. Shalf. Visualization of adaptive mesh refinement data. In *Proc. of Visual Data Exploration and Analysis VIII*, volume 4302, pages 121–132, Jan. 2001.

[28] G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, and K. I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In *Proc. of Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*, pages 25–34, 335, May 2001.

[29] G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, K. I. Joy, and K.-L. Ma. High-quality volume rendering of adaptive mesh refinement data. In *Vision, Modeling, and Visualization 2001*, pages 121–128, 522, Nov. 2001.

[30] J. Woodring and H.-W. Shen. Chronovolumes: a direct rendering technique for visualizing time-varying data. In *Proc. of Eurographics/IEEE TVCG Workshop on Volume Graphics*, pages 27–34, 2003.

[31] J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumetric data. In *Proc. of IEEE Visualization*, page 55, 2003.

[32] K. Wu, W. S. Koegler, J. Chen, and A. Shoshani. Using bitmap index for interactive exploration of large datasets. In *Scientific and Statistical Database Management*, pages 65–74, 2003.

[33] K. Wu, E. J. Otoo, and A. Shoshani. On the performance of bitmap indices for high cardinality attributes. In *Proc. of Very Large Data Bases*, pages 24–35, 2004.