Query-Driven Visualization Strategies for the Analysis and Visualization of Large, Complex Datasets

By

LUKE JEREMY GOSINK
B.S. Chemistry (University of California, Davis) 1996

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Prof. Kenneth I. Joy

Prof. Bernd Hamann

Prof. John D. Owens

Committee in charge

2009

Acknowledgments

I take this opportunity to acknowledge all who have made this work possible through their sacrifice, determined efforts, and supportive guidance. In the list of many to whom I am indebted, I first thank my adviser Ken Joy who gave me the opportunity to learn and conduct research at two world-renowned research groups: the Institute for Data Analysis and Visualization at the University of California at Davis, and the Visualization Group at Lawrence Berkeley National Laboratory at Berkeley. Ken's enthusiasm for scientific visualization and his confidence in this research have been key to its successful, evolving progress. I also thank John Owens who greatly motivated my work with multi-core architectures (Part II), and who inspired much of my general purpose GPU implementation work (Part III, and Part IV Chapter 5). I also thank Bernd Hamann for generously serving on both my qualification exam committee, and my dissertation committee.

I thank my mentor and supervisor Wes Bethel at Lawrence Berkeley National Laboratories. Wes was an original architect for the strategy of Query-Driven Visualization (QDV) and so has been fundamental to this work since its beginning. Over the years, Wes has been key in helping me to develop, clarify, and expound on much of this research that has helped the area of QDV to evolve. I also thank Lawrence Berkeley National Laboratories for their generous support and funding during the course of my graduate career.

I thank John Bell and Marc Day at the Center for Computational Science and Engineering in Lawrence Berkeley National Laboratory. They kindly provided much of the data I use in this work: the methane V-flame data (Part IV, Chapter's 4 and 5) and the turbulent ultra-lean premixed hydrogen combustion data (Part II and Part IV Chapter 4).

I also thank my colleagues at the Institute for Data Analysis and Visualization (IDAV) at UC Davis for their support, time, and feedback over the years. I specifically thank John Anderson for untold hours of time spent sharing, debating, and constructing ideas that were vital to this work; those talks will be sorely missed. I thank Shubhabrata Sengupta for sharing his wisdom on all things related to the GPU; his insights motivated some of the performance benefits that are observed in Part II. I would also like to thank Christoph Garth for his valuable guidance and friendship over the last two years I spent in graduate school; his gentle leadership-through-example as a postdoc helped me to greatly improve the quality of my research.

I warmly thank George Roussas in the Statistics Department at the University of California at Davis for all of his time and wisdom, as well as his friendship; George you inspired in me a love for statistics. Conversations with George motivated the work in Part IV of this dissertation.

I thank my extended family—Stu (Papa), and Gail (Mimi)—for their unceasing support and love over the last several years. I also thank my Mom, Dad, Craig (Dad #3), and second family—Ganesha (Dad #2) and Saraswati (Mom #2)—with all of my heart; you are the dear ones who set me in motion upon on this wonderful journey so many years ago and for that I will be eternally grateful. Last, I thank my wife Laura and daughter Aurelia for loving me and shouldering the vast portion of burden incurred with this work; words can't express how honored and grateful I am to be your husband and father.

Contents

List of Figures

List of Tables

Luke Jeremy Gosink
December 2009
Computer Science

Query-Driven Visualization Strategies for the Analysis and
Visualization of Large, Complex Datasets

Abstract

There is an urgent need in scientific communities, driven by their ability to generate ever-larger, increasingly complex data, for scalable analysis methods that rapidly identify salient trends in scientific data. Query-Driven Visualization (QDV) methods are among the small subset of techniques that are able to address both large and highly complex datasets—e.g. multivariate, multitemporal, and multiresolution representations of scalar, vector, and function field data. This dissertation presents new methods that either directly extend the utility and accelerate the performance of QDV as a whole, or enable QDV's substantial and flexible analysis strengths to be applied to new areas of scientific research.

The first part of this dissertation presents a new data-parallel strategy that accelerates the most fundamental task performed by QDV: the evaluation of user defined, ad hoc queries. The second part of this dissertation extends QDV strategies to analyze and visualize time-varying adaptive mesh refinement (AMR) data. AMR techniques are used in many scientific communities to efficiently and accurately model complex, continuous physical phenomena. By extending QDV methods to address the dynamic spatiotemporal properties of time-varying AMR data, I provide scientists with a powerful tool for visually analyzing the data generated from these important simulations. The final part of this dissertation leverages statistical analysis methods to generate deeper insight into the regions that are selected by a user's query. In this effort I introduce two new methods that increase the utility of query-driven strategies. The first strategy uses correlation fields, created between pairs of variables, in conjunction with the cumulative distribution functions (CDF) of variables expressed in a user's query. This strategy identifies important variable interactions within query regions. The second strategy forms a statistical-based segmentation within the query-region to generate deeper insight into the "statistical structure" of a user's query. In this approach, seg-

ments indicate which variable contributes most to the underlying joint density distribution of the user's query. These segments, when used in conjunction with each variable's CDF, intuitively aid users in refining the constraints over the variables in their query.

Part I

Query-Driven Visualization

Chapter 1

An Introduction to Query-Driven Visualization Strategies

Query-Driven Visualization (QDV) is a visualization-based discovery strategy that supports rapid data analysis, fosters dataset exploration, and helps scientists to interactively test and build hypothesis with their data, thereby accelerating the scientific discovery process. The term "Query-Driven Visualization" refers to the strategy of restricting computational and cognitive workloads, by either limiting or prioritizing processing, visualization and interpretation, exclusively to records defined to be scientifically important by the scientist. This strategy is based upon the observation that smaller subsets of data are usually the genesis of insight or breakthroughs to new trends [13, 45]; QDV's goal is to accelerate the task of finding these important subsets of data.

At a high level, QDV proceeds as a simple two-step process: query evaluation followed by the analysis and visualization of the query's results. Figure 1.1 illustrates this two-step process in the context of a QDV-based scientific workflow (bottom workflow) and contrasts it with a more traditional workflow (top workflow). The QDV user begins by characterizing, through a set of range conditions defined on selected variables (e.g. $1000 < temperature < 1200$), properties for data considered to be scientifically important. These constraints form the criteria necessary to construct a query, referred to as a *boolean range query*, whose solution separates records that are *not* important to the user's research from those records that are.

From a lower level, accelerated query processing techniques—the beige region of "Scien-

tific Data Management" in the bottom workflow of Figure 1.1—are first used to rapidly locate and isolate the records that meet the user's definition for important. After locating this data, analysis and visualization efforts are focused exclusively to these records. In QDV, the results of a query are traditionally visualized by rendering each record that passes the query as a single hexahedral cell; cell locations correspond to regions in the spatial domain where variable values meet the user's query constraints. Figure 1.2 illustrates applied Query-Driven Visualization using this type of cell-based rendering. Without querying to filter the important records, significant computational and cognitive resources are wasted in processing large amounts of data that are not particularly relevant to the user's needs.

As an illustrative example, assume a user is analyzing a dataset that simulates methane combustion. Further suppose that the user is interested in determining how concentrations of methane are distributed spatially throughout a fixed range of pressure (i.e. $5000 < pressure <$

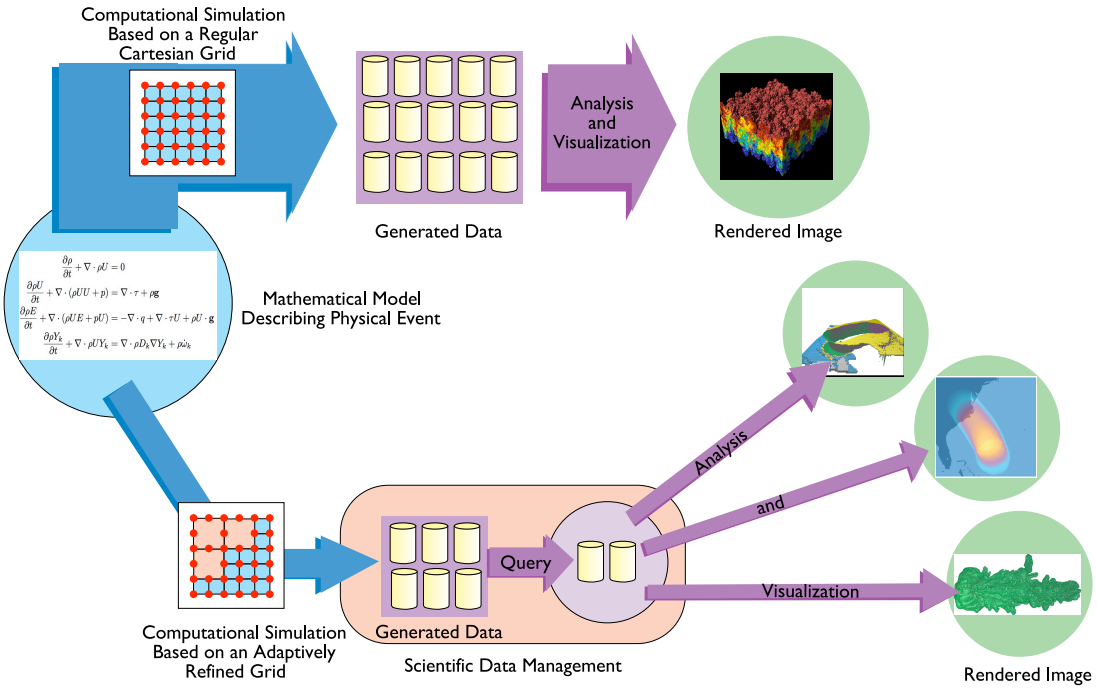
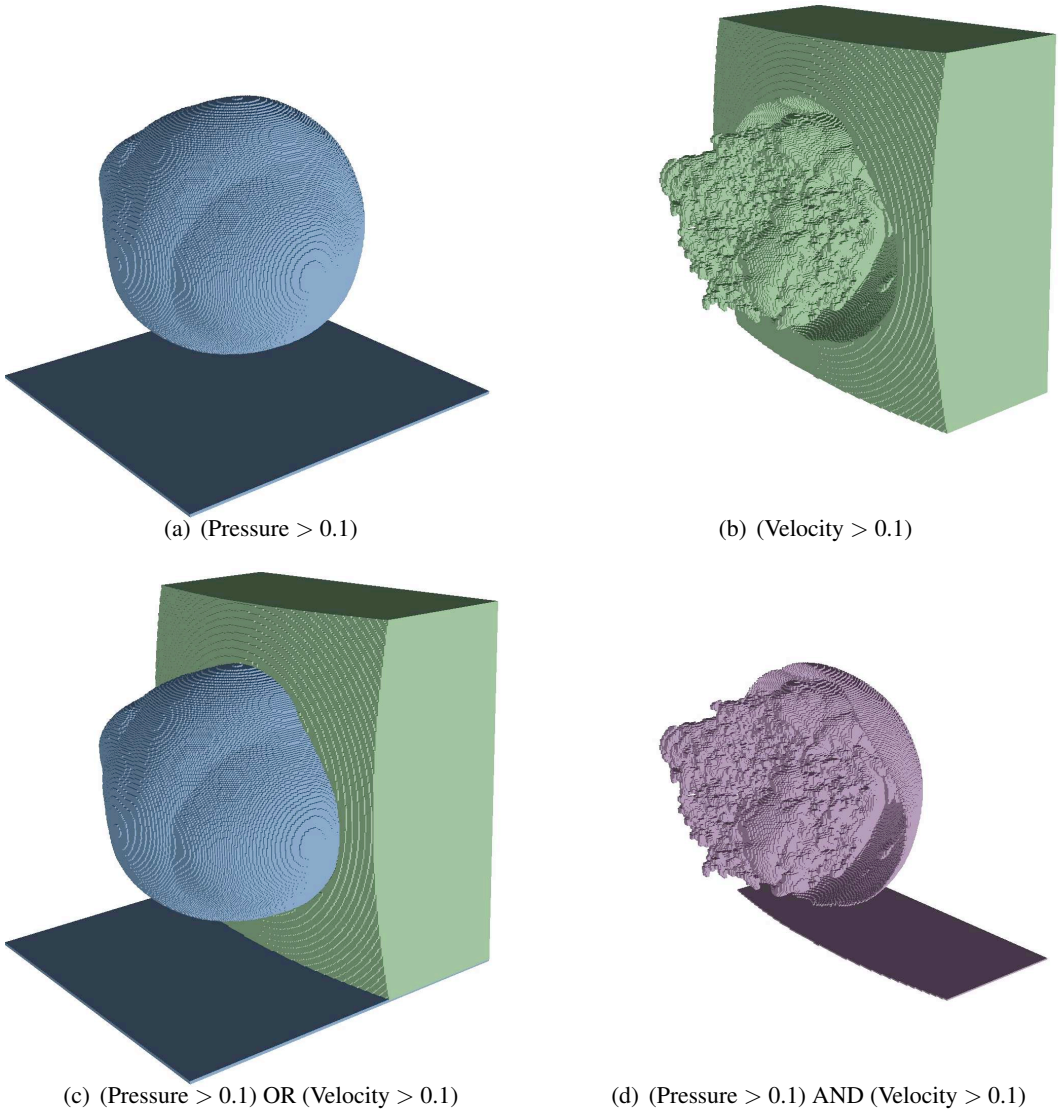


Figure 1.1: This figure depicts two differing scientific workflow models: a traditional workflow (top), and a QDV-accelerated workflow (bottom). The traditional workflow does not use any methods to efficiently generate data, or strategies to accelerate analysis and visualization tasks. The QDV workflow depicts a significantly more efficient scientific workflow. In this workflow a simulation based on adaptive mesh refinement techniques is coupled with QDV-based analysis and visualization strategies to accelerate the scientific discovery process.



(a) (Pressure > 0.1)

(b) (Velocity > 0.1)

(c) (Pressure > 0.1) OR (Velocity > 0.1)

(d) (Pressure > 0.1) AND (Velocity > 0.1)

Figure 1.2: Illustration of applied query-driven visualization (QDV) using cell-based rendering. (a) shows the regions corresponding to high pressure in a simulated supernova. (b) indicates the regions where high velocity, along the z-axis, occur in the same dataset. In the lower row of images, regions selected in (a) and (b) are combined using logical join, (c), and intersect, (d), operations. In (c) the colors correspond to regions of *either* high pressure (blue) OR high velocity (green). In contrast, the solution region rendered in (d) is rendered as a single color; this region contains *both* high pressure AND high velocity.

10000) from this combustion data. The user begins by constructing an initial query for records where ($methane <$ 1e-6) AND (5000 $< pressure <$ 10000). In such a query, the user is essentially requesting: "Show me the (spatial) regions where low concentrations of methane exist within a specific range of pressure."

The first QDV stage processes this query and identifies all records that satisfy the user's constraints. The second stage renders these regions to depict spatially where in the dataset the query's constraints were met. This process is, in many cases, very rapid and allows for interactive exploration of large, complex data. Thus after the initial query is processed and rendered, the user in my example could steadily increment the range constraints for *methane* concentration, while keeping the constraints for pressure fixed, and observe the changing records rendered in the visualization. The total process, sequentially querying and rendering regions of increasing *methane* concentration in a fixed range of *pressure*, interactively provides the information the user seeks.

This interactive and explorative approach taken by QDV is consistent with the needs of many scientific users who need methods that provide support, in the capacity of hypothesis testing and interactive exploration, to help them find and focus on critical features hidden in large, complex datasets—e.g. multivariate, multitemporal, and multiresolution representations of scalar, vector and function field data. This dissertation presents new methods that either directly extend the utility and accelerate the performance of QDV as a whole, or enable QDV's substantial and flexible analysis strengths to be applied to new areas of scientific research.

Accelerating Query-Driven Analysis and Visualization

I begin this dissertation (Part II) by presenting work that accelerates one of the most fundamental tasks performed in Query-Driven Visualization (QDV): the evaluation of user-defined ad hoc boolean range queries. One of the primary methods for accelerating the task of answering a query is to employ means for answering a query with parallel support, i.e. divide the work of the query over multiple processors and evaluate these individual portions of work simultaneously, in parallel. While high-end performance solutions for such a strategy would enlist the aid of networked CPU clusters or supercomputers, the path that I explore is to utilize a resource that is more accessible to researchers, the graphics processing unit (GPU). Current GPUs offer substantial performance

benefits over modern CPUs: accelerated processing capabilities afforded through massive data parallelism and rapid on-board data transfer rates. Additionally, they offer one of the cheapest forms of raw computational power available.

There are, however, challenges to utilizing this powerful resource for answering a query: GPU utility is limited by a reduced memory store and there are significant I/O performance limitations associated with accessing and transferring data from disk, or across networks, to the GPU. I address these challenges with a new Data-Parallel Bin-based Indexing Strategy (DP-BIS) that effectively utilizes the parallel processing power of the GPU. My approach concentrates on reducing both the amount of bandwidth and memory required to answer a query. I achieve this goal by integrating two key strategies: bin-based encoding and data partitioning facilitated through a modified Order-preserving Bin-based Cluster (OrBiC) [114]. Combined, the encoded data and modified OrBiC tables help overcome the limitations imposed by limited GPU memory and reduce the amount of data that must be accessed from disk when answering a query. In timing measurements the DP-BIS index can be 18X faster than existing indexing technologies such as the projection index. I employ the performance benefits of the DP-BIS index to create a GPU-based QDV engine in the next area of my research. In Part III of this dissertation, I address the challenges of applying QDV strategies to a specific type of multitemporal, multiresolution data referred to as adaptive mesh refinement data.

Extending the Utility of Query-Driven Analysis and Visualization

Query-Driven Analysis and Visualization of Adaptive Mesh Refinement (AMR) Data

In scientific research, the ability to generate detailed simulations that model time-dependent complex phenomena is essential to obtaining new insight and understanding. Unfortunately, these simulations can often place severe demands on computational and storage resources. AMR-based simulations ease these demands by extending the dynamic range of grid-based numerical methods beyond the limits feasibly supportable by hardware (i.e. when using a single-resolution grid of equivalent numerical accuracy). Emphasizing user-controlled spatiotemporal adaptivity throughout a given simulation, AMR methods provide substantial computational and storage savings over static grid approaches. AMR's resource-saving attributes have popularized the utilization of AMR techniques extensively in the fields of computational physics [14], engineering, combustion chem-

istry [15, 27], and now more recently, astrophysics [29] and cosmology [21, 81].

In this work, Part III of this dissertation, I introduce a new method that directly addresses the challenges associated with visualizing the dynamic spatial *and* dynamic temporal properties inherent in time-varying AMR data. I present a two-step method for compositing and synchronizing AMR data from a series of timesteps. I 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. I then synchronize each timestep's grid hierarchy to the composite template. This approach enables my method to process queries on a common AMR grid hierarchy. Using this data structure, I move the work of query processing to the GPU to realize the benefit of greatly accelerated QDV analysis. This work represents the first GPU-based QDV engine.

Statistical Analysis for Query-Driven Visualization

In the remaining portion of this dissertation, Part IV, I present new methods to extend the amount of information and insight that user's are able to obtain from QDV strategies. In Chapter 4, I present a new method for visually conveying statistical information about the trends that exist *between* variables in a user's query. In this method, correlation fields, created between pairs of variables, are used with the cumulative distribution functions of variables expressed in a user's query. This integrated use of cumulative distribution functions and correlation fields visually reveals, with respect to the solution space of the query, statistically important interactions between any three variables, and allows for trends between these variables to be readily identified. I demonstrate this method with two chemical combustion simulations by analyzing interactions between differing chemical species.

Following the theme of this statistics-oriented work, I present (Chapter 5) a statistics-based framework that models the underlying joint distribution within a query's solution. Using this model greatly enhances the information a user obtains from a query. For example, by exploring the limits of this distribution, users can visualize meaningful query boundaries. Exploring the distribution range inside these boundaries provides the user with critical information that can serve as a guide for refining the constraints over specific variables in the query. Finally, when constructing this joint distribution, I can formulate a segmentation between variables based on the distribution of all

variables constrained in the query. This segmentation facilitates both a direct visualization of how a user's query is composed and the importance of each variable to the joint distribution of the query.

I conclude this dissertation in Part V by outlining key areas for new QDV-related research.

Part II

Accelerating Query-Driven Visualization Through Data Parallelism

Chapter 2

Data Parallel Bin-Based Indexing for Answering Queries on Multi-Core Architectures

2.1 Introduction

Growth in dataset size significantly outpaces the growth of CPU speed and disk throughput. As a result, the efficiency of existing query processing techniques is greatly challenged [12, 28, 45]. The need for accelerated I/O and processing performance forces many researchers to seek alternative techniques for query evaluation. One general trend is to develop highly parallel methods for the emerging parallel processors, such as multi-core processors, cell processor, and the general-purpose graphics processing units (GPU) [7]. In this work, I propose a new parallel indexing data structure that utilizes a Data Parallel Bin-based Index Strategy (DP-BIS). I show that the available concurrency in DP-BIS can be fully exploited on commodity multi-core CPU and GPU architectures.

The majority of existing parallel database systems work focuses on making use of multiple loosely-coupled clusters, typified as shared-nothing systems [10, 30, 64, 76, 86, 87]. Recently, a new parallel computing trend has emerged. These type of parallel machines consist of multiple tightly-coupled processing units, such as multi-core CPUs, cell processors, and general-purpose

GPUs. The evolution of such machines in the coming decade is to support a tremendous number of concurrent threads working from a shared memory. For example, NVIDIA's 8800 GTX GPU—the GPU used in this work—has 16 multiprocessors, each of which supports 768 concurrent execution threads. Combined, these multiprocessors allow the GPU to manage over 12,000 concurrent execution threads. Fully utilizing such data parallelism on shared-memory systems requires developing new query processing algorithms that can support high levels of concurrency with respect to computation and data access.

A number of researchers have successfully demonstrated the employment of GPUs for database operations [32, 42, 43, 50]. Among the database operations, one of the basic tasks is to select a number of records based on a set of user specified conditions, e.g. "SELECT: records FROM: combustion_simulation WHERE: pressure $>$ 100." Many GPU-based works that process such queries do so with a projection of the base data [43,99]. Following the terminology in literature, I use the term *projection index* to describe this method of sequentially and exhaustively scanning all base data records contained in a column to answer a query [80]. On CPUs, there are a number of indexing methods that can answer queries faster than the projection index [25, 35, 113], but most of these indexing methods do not offer high enough levels of concurrency to take full advantage of a GPU. DP-BIS fully utilizes the GPU's data parallelism when answering a selection query; each thread on the GPU is used to independently access and evaluate an individual record. This one-to-one mapping of threads-to-records lets the DP-BIS index process large amounts of data using over 12,000 concurrent parallel operations at any one time.

Though GPUs offer tremendous data parallelism, their utility for database tasks is limited by a small store of resident memory. For example, the largest amount of memory available on NVIDIA's Quadro FX GPU is currently 4.0 GB, which is much too small to hold projections of all columns from a dataset of interest [12, 28, 45]. DP-BIS presents one method for ameliorating the challenges imposed by limited GPU memory. The DP-BIS index uses a form of data encoding that is implemented through a multi-resolution representation of the base data information. This encoding effectively reduces the amount of data the DP-BIS index must access and transfer when answering a query. As a result of the encoding, the DP-BIS index can query dataset sizes that would otherwise not fit into the memory footprint of a GPU. Additionally, by transferring smaller amounts of data when answering a query, the DP-BIS index utilizes data bus bandwidth more efficiently.

In the DP-BIS approach, I separately bin each column's base data using an equal depth binning strategy. I augment each column's binned index by generating a corresponding Data Parallel Order-preserving Bin-based Cluster (OrBiC). To resolve a query condition on a column, I first determine the boundaries of the query. Consider an example. For range conditions such as "*pressure* $>$ 100", I determine the bin whose range captures the constraint "100". In this example, assume that the value "100" is contained in the value range captured by bin_{17}. I refer to bins that capture one of the query's constraints as "boundary bins". In this example, records contained in bins less than the boundary bin (i.e. $\text{bin}_0 \rightarrow \text{bin}_{16}$) fail the query. Correspondingly, records contained in bins greater than the boundary bin pass the query. Boundary bin records can't be characterized by their bin number alone; they must be evaluated by their base data value. I call the records in the boundary bin the candidates and the process of examining the candidate values the candidate check [97]. The DP-BIS index strategy for answering a selection query is similar to that of a binned bitmap indexing strategy [5, 6, 112]. A central difference is that bitmap-based strategies indicate the record contents of each bin with a single bitmap vector; to answer a query, bitmap vectors are logically combined to build the query's solution. In contrast, the DP-BIS index directly accesses the bin numbers of all records from an encoded data table.

The Data Parallel OrBiC structure I use during the candidate check procedure provides an efficient way to extract and send boundary bin data from the CPU to the GPU. Additionally, this structure facilitates a rapid, concurrent way for GPU threads to access this data. Altogether, to answer a query, the DP-BIS index accesses the bin numbers and the base data values of the records in boundary bins. The total data contained in both these data structures is 75% smaller than the column projections used by other strategies that employ the GPU to answer a query. Additionally, the procedure for examining the bin numbers and the process of performing the candidate checks offer the same high level of concurrency as the GPU projection index.

In this work I assume that the base data will not (or seldom) be subjected to modification. This assumption too is made by other research database management systems that operate on large data warehouses that contain read-only data: e.g. MonetDB [20], and C-Store [98]. In addition to such database management systems, many scientific applications also accumulate large amounts of data that is never modified or subjected to transactions [45].

Finally, I specifically utilize and emphasize the GPU in this work because it possesses

the highest degree of data parallelism available in existing multi-core architectures. To this extent I view the GPU as a representative case of where multi-core architectures are evolving with respect to data parallelism and processing performance. In summary, this research makes the following contributions.

- I introduce a data parallel bin-based indexing strategy (DP-BIS) for answering selection queries on multi-core architectures. The concurrency provided by DP-BIS fully utilizes the data parallelism emerging in these architectures in order to benefit from their increasing computational capabilities.

- I present the first strategy for answering selection queries on a GPU that utilizes encoded data. This encoding strategy facilitates significantly better utilization of data bus bandwidth and memory resources than GPU-based strategies that rely exclusively on base data.

- I implement and demonstrate DP-BIS's performance on two commodity multi-core architectures: a multi-core CPU and a GPU. I show in performance tests that both implementations of DP-BIS are 3–4X faster than the GPU and CPU-based projection index with respect to total query response times. I additionally show that the GPU-based implementation of DP-BIS outperforms all index strategies with respect to computation-based times.

2.2 Background and Related Work

2.2.1 Related Bitmap Index Work

The data stored in large data warehouses and the data generated from scientific applications typically consists of tens to hundreds of attributes. When answering queries that evaluate such high-dimensional data, the performance of many indexing strategies diminishes due to *the curse of dimensionality* [117]. The bitmap index is immune to this curse and is therefore known to be the most efficient strategy for answering ad hoc queries over such data [79]. For this reason, major commercial database systems utilize various bitmap indexing strategies (e.g. ORACLE, IBM DB2, and Sybase IQ).

Another trait of the bitmap index is that storage concerns for indices are ameliorated through specialized compression strategies that both reduce the size of the data and that facilitate

the efficient execution of bitwise Boolean operations [3]. Antoshenkov [5] presents a compression strategy for bitmaps called the Byte-aligned Bitmap Code (BBC) and shows that it possess excellent overall performance characteristics with respect to compression and query performance. Wu et al. [112] introduce a new compression method for bitmaps called Word-Aligned Hybrid (WAH) and show that the time to answer a range query using this bitmap compression strategy is optimal; the worse case response time is proportional to the number of hits returned by the query. Recent work by Wu et al. [114] extends the utility of the bitmap index. This work introduces a new Order-preserving Bin-based Clustering structure (OrBiC), along with a new hybrid-binning strategy for single valued bins, that helps the bitmap index overcome *the curse of cardinality*; a trait where both index sizes and query response time increase in the bitmap index as the number of distinct values in an attribute increases.

Sinha and Winslet [93] successfully demonstrate parallelizable strategies for binning and encoding bitmap indexes, compressing bitmap vectors, and answering selection queries with compressed bitmap vectors. Their work supports bitmap use in a highly parallel environment of multiple loosely-coupled, shared-nothing systems. In contrast, my research addresses the challenges of supporting bin-based indexing on the newly emerging, tightly-coupled architectures that provide tremendous data parallelism at the node level in loosely-coupled, clustered systems; for example the graphics processor unit (GPU).

The basic attributes of the binned bitmap index (bin-based indexing, the use of simple boolean operators to answer selection queries, etc.) can be implemented in a highly parallel environment. For this reason, my new Data Parallel Bin-based Indexing Strategy (DP-BIS) follows the general structure of a binned bitmap index. Unfortunately, bitmap compression strategies, even the parallelizable strategies of Sinha and Winslet [93], do not support enough task-level parallelism to take advantage of the data parallelism offered by tightly-coupled architectures like GPUs. Thus one of the first objectives in my research is to develop a compression strategy, based upon the binning techniques of the binned bitmap index, that supports high levels of concurrency and reduces the amount of data required to answer a query.

2.2.2 Related GPU-Database Work

GPUs have been used to help support and accelerate a number of database functions [32, 38,42,43,50,70], as well as numerous general purpose tasks [49,83]. Sun et al. [99] present a method for utilizing graphics hardware to facilitate spatial selections and intersections. In their work, they utilize the GPU's hardware-accelerated color blending facilities to test for the intersection between two polygons in screen space.

Working within the constraints of the graphics API for fragment shaders, Govindaraju et al. [43] present a collection of powerful algorithms on commodity graphics processors for performing the fast computation of several common database operations: conjunctive selections, aggregations, and semi-linear queries. This work also demonstrates the use of the projection index to answer a selection query. Additionally, Govindaraju et al. [42] present a novel GPU-based sorting algorithm to sort billion-record wide databases. They demonstrate that their "GPUTeraSort" outperforms the Indy PennySort1 record, achieving the best reported price-for-performance as of 2006 on large databases.

More recent GPU-database work utilizes powerful, new general purpose GPU hardware that is supported by new data parallel programming languages (see Section 2.3). These hardware and software advances allow for more complex database primitives to be implemented on the GPU. Fang et al. [32] implement the CSS-Tree in the software GPUQP. This work characterizes how to utilize the GPU for query co-processing; unfortunately there is no performance data published about the implementation.

Lieberman et al. [61] implement an efficient similarity join operation in CUDA. Their experimental results demonstrate that their implementation is suitable for similarity joins in high-dimensional datasets. Additionally, their method performs well when compared against two existing similarity join methods.

He et al. [49] improve the data access locality of multi-pass, GPU-based gather and scatter operations. They develop a performance model to optimize and evaluate these two operations in the context of sorting, hashing, and sparse matrix-vector multiplication tasks. Their optimizations yield a 2–4X improvement on GPU bandwidth utilization and 30–50% improvement on performance times. Additionally, their optimized GPU-based implementations are 2–7X faster than optimized

CPU counterparts. He et al. [50] present a novel design and implementation of relational join algorithms: non-indexed and indexed nested loops, sort-merge, and hash joins. This work utilizes their bandwidth optimizations [49], and extends the work of Fang et al. [32]. They support their algorithms with new data-parallel primitives for performing map, prefix-scan and split tasks. Their work achieves marked performance improvements over CPU-based counterparts; GPU-based join algorithms are 2–7X faster than CPU-based approaches.

GPU-based strategies that address how to answer a selection query have yet to address the significant limitations imposed by the GPU's small memory, and those imposed by the data buses that transfer data to the GPU. To the best of my knowledge, all relevant literature utilizes algorithms that operate on a column's base data (i.e. non-compressed data). Utilizing base data severely restricts the amount of data the GPU can process. Further, streaming large amounts of base data to the GPU can impede the processing performance of many GPU-based applications. More specifically, GPU processing performance can rapidly become bottlenecked by data transfer rates if these transfer rates are not fast enough to keep the GPU supplied with new data. This bottleneck event occurs on GPUs whenever the arithmetic intensity of a task is low; the process of answering a simple range query falls into this classification.

In the following sections, I introduce some basic GPU fundamentals, as well as the languages that support general purpose GPU programming. I then introduce my Data Parallel Bin-based Indexing Strategy (DP-BIS) and show how it directly addresses the challenges of limited GPU-memory and performance-limiting bus speeds with a fast, bin-based encoding technique. This work is the first GPU-based work to present such an approach for answering queries. I also show how the DP-BIS index's binning strategy enables DP-BIS to support a high level of concurrency. This concurrency facilitates a full utilization of the parallel processing capabilities emerging in multi-core architectures.

2.3 GPUs and Data Parallel Programming Languages

Recent GPU-database works utilize powerful new data parallel programming languages like NVIDIA's CUDA [78] and OpenCL [73]. These new programming languages eliminate the long-standing tie of general-purpose GPU work with restrictive graphics-based APIs (i.e. frag-

ment/shader programs). Further, the GPUs supporting these languages also facilitate random read and write operations in GPU memory—scatter I/O operations are essential for GPUs to operate as a general-purpose computational machine.

The functional paradigm of these programming languages views the GPU as a co-processor to the CPU. In this model, the programmer writes two separate kernels for a general purpose GPU (GPGPU) application: code for the GPU kernel and the code for the CPU kernel. Here the CPU kernel must proceed through three general stages.

1. Send a request to the GPU to allocate necessary input and output data space in GPU memory. The CPU then sends the input data (loaded from CPU memory or hard disk) to the GPU.

2. Call the GPU kernel. When the CPU kernel calls a GPU kernel, the CPU's kernel suspends and control transfers to the GPU. After processing its kernel, the GPU kernel terminates and control is transferred back to the CPU.

3. Retrieve the output data from the GPU's memory.

From a high level, the GPU kernel serves as a sequence of instructions that describes the logic that will direct each GPU thread to perform a specific set of operations on a unique data element. The kernel thus enables the GPU to direct the concurrent and simultaneous execution of all GPU threads in a SIMT (single-instruction, multiple-thread) workflow. The GPU executes its kernel (step two above) by first creating hundreds to thousands of threads—the number of threads is user specified and application dependent. During execution, small groups of threads are bundled together and dynamically dispatched to one of the GPU's numerous SIMD multiprocessors. These thread bundles are then delegated by the multiprocessor to one of its individual processors for evaluation. At any given clock cycle, each processor will execute the same kernel-specified instruction on a thread bundle, but each thread will operate on different data.

With respect to memory resources, each GPU multiprocessor contains a set of dedicated registers, a store of read-only constant and texture cache, and a small amount of shared memory. These memory types are shared between the individual processors of a multiprocessor. In addition to these memory types, threads evaluated by a processor may also access the GPU's larger, and comparatively slower, global memory.

There are two important distinctions to make between GPU threads and CPU threads. First, there is no cost to create and destroy threads on the GPU. Additionally, GPU multiprocessors perform context switches between thread bundles (analogous to process switching between processes on a CPU) with zero latency. Both of these factors enable the GPU to provide its data parallelism with very low overhead.

2.4 A Data Parallel Bin-based Indexing Strategy (DP-BIS)

To effectively utilize a GPU, an indexing data structure must provide high levels of concurrency to fully benefit from the GPU's large number of concurrent execution threads, and make effective use of the GPU's relatively small memory. In this section I present the DP-BIS method and show how it successfully addresses these requirements by integrating two key strategies: data binning (Section 2.4.1) and the use of Data Parallel Order-preserving Bin-based Clusters (OrBiC) (Section 2.4.2).

When answering a query, the binning strategy I utilize significantly reduces the amount of data the DP-BIS index must access, transfer, and store on the GPU. The Data Parallel OrBiC structure I employ ensures that candidate checks only access the base data of the boundary bins. The concurrency offered by both of these data structures facilitates full utilization of the GPU's data parallelism. The DP-BIS strategy builds one index for each column in a database, where each index consists of an encoded data table (i.e. the bin numbers), and a Data Parallel OrBiC structure. When answering a simple range query with the DP-BIS index, I access the encoded data table, and the base data of two bins (the boundary bins) from the data parallel OrBiC structure.

2.4.1 Base Data Encoding

The index construction process begins by binning all of the base data records contained in a single column. To minimize data skew in my binning strategy, I select the bin boundaries so that each bin contains approximately the same number of records. In cases where the frequency of a single value exceeds the allotted record size for a given bin, a *single-valued bin* is used to contain all records corresponding to this one value. This technique to address data skew is consistent with other binning strategies [114]. I then encode the base data by representing each base data record

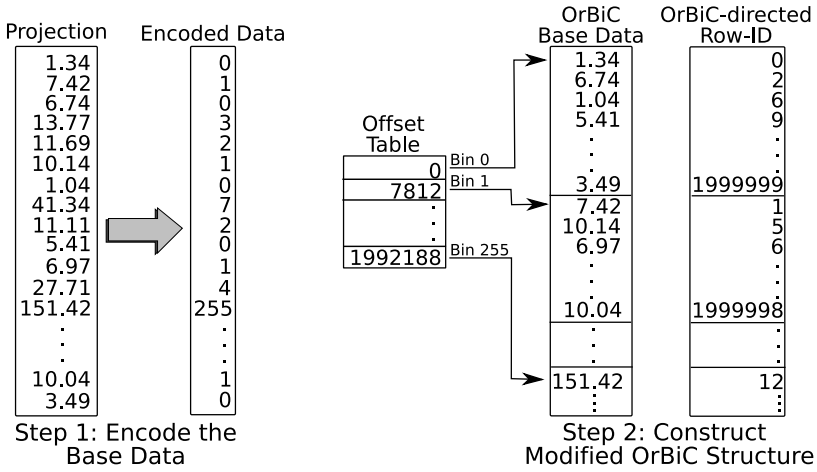


Figure 2.1: This figure shows the two-step DP-BIS index construction process. The first step encodes a column's full-resolution base data (Section 2.4.1). The second step (Section 2.4.2) utilizes this same full-resolution information to generate a modified (i.e. Data Parallel) OrBiC Structure.

with its associated bin number. Figure 2.1 shows an example of this encoding. In later discussions, I refer to the bin numbers as low-resolution data and the column's base data as full-resolution data. I always utilize 256 bins in my encoding procedure. As I now show, the amount of data generated by using this number of bins facilitates near-optimal usage of data bus bandwidth and GPU memory space when answering a query.

Assume that all full-resolution data is based on 32-bit values, and that there are N records in a given database column. Using x bits to represent each bin, I can create 2^x bins where each bin will contain, on average, $\frac{N}{2^x}$ records. The total size of the low-resolution data will then be $x \times N$ bits. The candidate data for each boundary bin, assuming each row-id can be stored in 32-bits, will consist of $\frac{32 \times N}{2^x}$ bits for row-identifiers, and $\frac{32 \times N}{2^x}$ bits for data values. The total number of bits (written as B below) DP-BIS utilizes to answer a simple range query is therefore:

$$B = \underbrace{x \times N}_{Low-Resolution\ Bits} + \underbrace{(4 \times \frac{32 \times N}{2^x})}_{Candidate\ Check\ Bits\ for\ Boundary\ Bins} \tag{2.1}$$

Note that the candidate check bit cost for boundary bin data is based on two boundary bins; this data size represents the more typical, and expensive, workload for answering a simple range query.

Number of Bins	Low-Resolution Size(%)	Boundary Bin Size(%)	Total Data Size(%)
$2^{32} = 4294967296$	100.0	0.0	100.0
$2^{16} = 65536$	50.0	$100.0 \times \frac{2}{65536} = 0.003$	50.0 + 0.003 = 50.0
2^8=256	**25.0**	**$100.0 \times \frac{2}{256}$ = 0.78**	**25.0 + 0.78 = 25.8**

Table 2.1: This table presents the total benefit for DP-BIS to utilize a specific number of bins in its encoding strategy. All values in column two, three, and four are given in terms of a percentage of the total full-resolution data (assuming 32-bits are utilized to represent each full-resolution record). Note the *Boundary Bin Size* reflects the cost for *two* boundary bins. The use of 256 bins in the DP-BIS index's encoding strategy reduces the amount of data that the DP-BIS index must transfer and store by over 74%.

Taking the derivative of B with respect to x, I get:

$$\frac{dB}{dx} = N - (128 \times N \times \frac{\ln 2}{2^x}) \tag{2.2}$$

By setting this derivative to 0 and solving for x, I compute the optimal number of bits to use for the DP-BIS index's encoding strategy:

$$B_{min} = 7 + \log_2(\ln(2)) \approx 6.4712(bits) \tag{2.3}$$

In my encoding strategy, bins can be represented with either 32-bits, 16-bits, or 8-bits; these are the most easy and efficient data sizes for GPUs and CPUs to evaluate. Use of alternate data sizes, like the "optimal" 6-bit data type I have derived in Equation 2.3, are not convenient for GPU-processing. The closest integer type that is conveniently supported on the GPU is the 8-bit integer. Therefore I use 8-bit integers to represent bin numbers and I use 256 bins in the DP-BIS index's encoding strategy.

Table 2.1 illustrates the benefit of using 256 bins from a less formal standpoint. This table shows realized data transfer costs for answering a simple range query based on data types efficient for CPU and GPU computation. The last row validates Equation 2.3; 8-bit bins reduce the amount of data that the DP-BIS index must access, transfer, and store on the GPU by over 74%, thereby provide the best encoded-based compression.

2.4.2 Extending OrBiC to Support Data Parallelism

Wu et al. [114] introduce an Order-preserving Bin-based Clustering (OrBiC) structure; this structure facilitates highly efficient candidate checks for bitmap-based query evaluation strategies. Unfortunately, the OrBiC structure does not offer enough concurrency to take advantage of the GPU's data parallelism. In this section, I present the constructs of the original OrBiC data structure, and then address how I extend this index to provide greater levels of concurrency.

In the approach presented by Wu et al., the full-resolution data is first sorted according to the low-resolution bin numbers. This reordered full-resolution table is shown as the "OrBiC Base Data" table in Figure 2.1. In forming this table, each bin's start and end positions are stored in an offset table. This offset table facilitates contiguous access to all full-resolution data corresponding to a given bin.

I extend the work of Wu et al. by building an OrBiC-directed table; this is the "OrBiC-directed Row-ID" table in Figure 2.1. This table holds row-identifier information for the full-resolution data records. The appended "directed" statement refers to the fact that the ordering of this table is directed by the ordering of the OrBiC Base Data table. With consistent ordering between these tables, start and end locations for a given bin in the offset table provide contiguous access to both the full-resolution data contained in this bin and the correct row-identifier information for each of the bin's records.

The OrBiC-directed row-identifier table facilitates data parallelism by addressing a fundamental difference between my data parallel bin-based strategy and the bitmap work of Wu et al. [114]. Specifically, Wu et al. create a single bitmap vector for each bin in the OrBiC Base Data table. As the bitmap vector associated with a given bin stores the bin's row-identifier information implicitly, their procedure does not need to keep track of the row-identifiers. In my case such a strategy is not inherently parallelizable. I thus employ an explicit representation of the row-identifier information by storing them in the OrBiC-directed Row-ID table. Using this table, threads can simultaneously and in parallel perform candidate checks on all records in a given boundary bin.

2.4.3 DP-BIS: Answering a Query

In this work, I focus on using DP-BIS to answer simple and compound range queries. Range queries in general are a common database query expressed as a boolean combination of two simple predicates: $(100.0 \leq X)$ AND $(X \leq 250)$, or alternatively $(100.0 \leq \mathrm{X} \leq 250)$. Compound range queries logically combine two or more simple range queries using operators such as AND, and OR: $(X \leq 250)$ AND $(\ \mathrm{Y}\ \leq 0.113)$.

Index strategies that answer range queries efficiently and rapidly are a crucial underpinning for many scientific applications. In this dissertation I couple DP-BIS with scalable visualization methods to accelerate the performance of query-driven visualization (QDV) strategies. With DP-BIS, my QDV implementations rapidly pare down large complex data, and allow smaller more meaningful subsets of data to be efficiently analyzed and visualized.

Simple Range Queries

The DP-BIS process for answering a simple range query consists of three stages: load necessary input data onto the GPU, execute the GPU kernel, and download the output data (i.e. the query's solution) from the GPU to the CPU. The input for this process consists of a single low-resolution database column, all necessary full-resolution record and row-identifier data, and two real values that will be used to constrain the column. The process returns a boolean bit-vector—a boolean column with one entry per data record that indicates which records have passed the query.

Given a query, the CPU kernel first accesses the appropriate low-resolution data column from disk. Next, space is allocated in GPU memory to hold both this data as well the query's solution. After allocating memory, and sending the low-resolution data to the GPU, the CPU kernel identifies the boundary bins of the query. The query's boundary bins are the bins whose ranges contain the query's real-valued constraints. The CPU kernel uses these bin numbers as an index into the OrBiC offset table. Values in the offset table provide the start and end locations in the OrBiC Base Data, and Row-ID tables for the candidate record's full-resolution data and corresponding row-identifiers. After the candidate data is sent to the GPU, the CPU kernel then calls the necessary GPU kernels.

The first GPU kernel, shown in Algorithm 1, processes the column's low-resolution data.

Algorithm 1
GPU Kernel for Low-Resolution Data

Require: **Integer** lowBinNumber, **Integer** highBinNumber, **Integer []** lowResolution

```
1:  position← ThreadID
2:  binNum← lowReslution[position]
3:  if (binNum>lowBinNumber) then
4:     if (binNum<highBinNumber) then
5:        Sol[position] ← TRUE
6:     end if
7:  end if
8:  if (binNum<lowBinNumber) then
9:     Sol[position] ← FALSE
10: end if
11: if (binNum>highBinNumber) then
12:    Sol[position] ← FALSE
13: end if
```

Algorithm 2
GPU Kernel for Candidate Checks

Require: **Float** lowReal, **Float** highReal, **Float []** fullResolution, **Integer []** rowID

```
1:  position← ThreadID
2:  recordVal← fullReslution[position]
3:  record_RowID← rowID[position]
4:  if (recordVal>lowReal) then
5:     if (recordVal < highReal) then
6:        Sol[record_RowID] ← TRUE
7:     end if
8:  end if
9:  if (recordVal< lowReal) then
10:    Sol[record_RowID] ← FALSE
11: end if
12: if (recordVal>highReal) then
13:    Sol[record_RowID] ← FALSE
14: end if
```

In setting up this kernel, the CPU instructs the GPU to create one thread for each record in the column. The CPU then calls the GPU kernel, passing it the boundary bin numbers; these boundary bin numbers enable threads to answer an initial low-resolution query. At launch time, each thread first determines its unique thread identifier[1]. Threads use their identifier to index into the *lowResolution* data array (line 2); this array is the low-resolution data column loaded earlier by the CPU. The thread characterizes its record as passing or failing depending on whether the record's bin number lies interior, or exterior to the boundary bins (lines 3, 4, 8, and 9). The answer to each thread's query is written to the query's solution space in GPU memory. This space, previously allocated by the CPU kernel, is shown in Algorithm 1 as *Sol*[].

The next GPU kernel, shown in Algorithm 2, performs a candidate check on all records contained in a given boundary bin. In the DP-BIS strategy, I launch the candidate check kernel twice: once for the lower boundary and once for the higher boundary bins.

The candidate check kernel is similar to the previous GPU kernel. Thread identifiers enable each thread to index into the *Full-Resolution* and *rowID* arrays of their respective boundary bin; these arrays are the OrBiC tables previously loaded onto the GPU. These arrays enable the

[1]Each GPU thread has a unique ID that aids in coordinating highly parallel tasks. These unique IDs form a series of continuous integers, $0 \rightarrow maxThread$, where maxThread is the total thread count set for the GPU kernel.

Logic-Based Kernel	Algorithm 1 line 1.4, and Algorithm 2 line 2.5 change to:	Algorithm 1 line 1.7, and Algorithm 2 line 2.8 change to:
AND	"Sol[x] ← Sol[x]"	"Sol[x] ← **FALSE**"
OR	"Sol[x] ← **TRUE**"	"Sol[x] ← Sol[x]"

Table 2.2: This table shows the required changes to make to Algorithms 1 and 2 to form the logic-based kernels DP-BIS uses to answer compound range queries.

kernel's threads to access the full-resolution data and corresponding row-identifier information for all records that lie in the boundary bin. Threads characterize each record as passing or failing based on comparisons made with the accessed full resolution data (lines 4, 5, 9, and 12 in Algorithm 2). The results of these logical comparisons are written to *Sol*[], ***not*** using the thread's identifier as an index, but the accessed row identifier (obtained from *rowID*) corresponding to the evaluated record.

Compound Range Queries

From a high level, the DP-BIS index answers a compound range query by logically combining the solutions obtained from a sequence of simple range queries. To perform this task efficiently, the DP-BIS index directs each simple query's kernel to utilize the same solution space in GPU memory. The compound range query's solution is produced once each simple query has been answered. In more complicated cases, e.g. "(X1 AND X2) OR (X3 AND X4)", the solution to each basic compound query can be written to a unique bit in the GPU's solution space; the bits can then be combined in each GPU kernel as needed (through bit-shifts) to form the solution to the query.

From a lower level, DP-BIS answers the first simple range query with the kernels outlined in Algorithms 1 and 2. These kernels perform unconditional writes to the compound range query's solution space. More specifically, all threads "initialize" this solution space with the first simple range query's solution. All subsequent simple range queries, however, utilize logic-based (AND, OR, etc.) derivatives of these kernels. These logic-based kernels only differ from the kernels outlined in Algorithms 1 and 2 in the final write operation each thread performs (lines 5, 9, and 12, and lines 6, 10, and 13 respectively). These changes, shown in Table 2.2, ensure that each thread, logically combines the current simple range query's solution with the existing compound range query's solution. Section 2.6.2 demonstrates the implementation and performance of this approach.

	Column Size (-in millions of rows-)						
	50	100	145	200	250	300	350
Base Data Size (-in MB-)	200	400	580	800	1000	1200	1400
DP-BIS Index Size (-in MB-)	450	900	1305	1800	2250	2700	3150
Index Build Time (-in minutes-)	1.22	2.65	4.12	5.82	7.49	9.37	11.36

Table 2.3: This table shows the index sizes and DP-BIS index build times for each column used in my tests. The size for the DP-BIS index includes the size for the encoded data table, as well as the size for the OrBiC base and row identifier data tables. All times represent an average build time calculated from ten test builds.

2.5 Datasets, Index Strategies, and Test Setup

In this section I describe the datasets and index strategies I use in my performance analysis. I discuss testing parameters at the end of this section.

All tests were run 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. My hardware setup consists of an Intel QX6700 quad-core multiprocessor, 4 GB of main memory, and a SATA storage system that provides 70 MB/s sustained data transfer rates. The GPU co-processor I use is NVIDIA's 8800GTX. This GPU provides 768 MB of memory and can manage over 12,000 concurrent execution threads.

2.5.1 Datasets

I use two datasets in my performance analysis. In both datasets, the records consist of 32-bit floating point data. The time to build the DP-BIS index for each column in the datasets is shown in Table 2.3; note the cost in time to build the indices scales well with the increasing size of the base data. In this table, the size for the DP-BIS index includes the size for the encoded data table, as well as the size for the OrBiC base and row identifier data tables.

The first dataset I use is produced by a scientific simulation modeling the combustion of hydrogen gas in a fuel burner. This dataset consists of seven columns where each subsequent column increases in row size: 50 million rows for column one, 100 million rows for column two,..., 350 million rows for column seven. I use this dataset in Section 2.6.1 to measure and compare the effect that increasing column size has on processing and I/O performance.

The second dataset I use is synthetically produced. This dataset consists of 7 columns each with 50 million rows. Each column consists of a series of randomly selected, randomly distributed values from a range of floating point values [-32767.0, 32767.0]. In Section 2.6.2 I answer a series of compound range queries over this data. This experiment measures and compares the processing and I/O costs of finding the union or intersection between an increasing number of columns.

2.5.2 Index Strategies

In my tests, I evaluate the I/O and processing performance of two indexing strategies: DP-BIS and the projection index. I independently evaluate the concurrency each index affords by implementing and testing the performance of a CPU-based and a GPU-based version of the index.

The CPU-based DP-BIS index is implemented on a multi-core CPU that contains four CPU cores. In this implementation, the work of answering the query is divided separately and equally over each CPU core through the use of Pthreads [74]; here each CPU core is assigned an individual thread and a portion of the DP-BIS low-resolution and full-resolution data to evaluate.

The GPU-based DP-BIS index is implemented on a GPU using the constructs of the data parallel programming language CUDA. This implementation is directly based on the method presented in Section 2.4.3.

The CPU projection index begins by reading each full-resolution column into CPU memory space. The query is answered by simply performing comparisons on the array(s) without any additional data structure. I use this strategy in my tests because it provides a good baseline for assessing performance.

The GPU projection index is similar to the CPU projection index, with the exception that the full-resolution columns are read into GPU memory space. Additionally, all indexed values in a given column are simultaneously evaluated in parallel by the query. This indexing strategy supports the same level of concurrency offered by DP-BIS (i.e. each thread evaluates a single record), but does not provide the benefits of encoding. On the other hand, this index approach does not require performing candidate checks; a procedure that requires additional computation and read requests to GPU memory.

2.5.3 Test Setup

To ensure that all queries reflect cold-start, cold-cache behavior, I force all read operations to bypass the OS cache to prevent Windows-based data caching. Therefore, all performance times, unless otherwise stated, are based on the complete time to answer the query. This time measurement, which I refer to as the query's "total performance time", includes:

1. Disk access and data transfer times (including the cost for allocating necessary memory on the CPU and GPU),
2. time to upload data to the GPU (not applicable for the CPU-based index),
3. the time to answer a query on the uploaded data, and
4. the time to download the solution from the GPU to the CPU (again, not applicable for the CPU-based index).

In my performance analysis, I divide this total performance time into two separate time metrics, based on work-related tasks. The first time I refer to as the "I/O performance time". This time includes the time to perform all data transfers and memory allocation: numbers 1, 2, and 4 from the list above. The second time, which I refer to as "processing performance time", includes the time to perform all computation-related work (number 3 from the list above). In my experiments realized total, I/O, and processing performance times are recorded individually, and simultaneously. Finally, unless specified, each reported performance value represents the mean value calculated from 25 separate test runs.

2.6 Query Performance

Typically, when answering a simple or compound range query over a large amount of data, far more time is spent accessing and transferring data than computing the query's solution. The performance of such I/O-intensive tasks are commonly limited by data transfer speeds. This I/O-based performance bottleneck is an especially significant challenge for multi-core architectures, like GPUs, where processing rates can far exceed bandwidth speeds [78].

Indexing Method	Mean Time Spent Transferring Data -as a percentage of the total time-	Mean Time Spent Answering the Query -as a percentage of the total time-
CPU-Projection	96.70 ± 0.19	3.30 ± 0.19
GPU-Projection	99.48 ± 0.26	0.52 ± 0.26
DP-BIS (GPU)	98.13 ± 1.03	1.87 ± 1.03
DP-BIS (CPU)	93.33 ± 0.7	6.67 ± 0.7

Table 2.4: This table shows how the total performance time for each index strategy is composed based on I/O-related workloads, and compute-based workloads. Each value in this table represents the mean percentage of time observed for a given index strategy, based upon all tests performed.

I demonstrate in this section how the strategy behind DP-BIS presents one way to ameliorate this I/O-based performance bottleneck. By operating primarily on encoded data, the DP-BIS index significantly reduces the effects of this bottleneck, and uses CPU and GPU memory resources more efficiently. Additionally, the level of concurrency afforded by DP-BIS facilitates a full utilization of the data parallelism provided by both multi-core CPU and GPU architectures. In this section I demonstrate the benefits of this concurrency by directly comparing processing performance times for CPU and GPU-based DP-BIS implementations. From this comparison, I show that the GPU-based implementation accelerates processing performance by a factor of 8X over the CPU-based implementation.

2.6.1 Answering a Simple Range Query

In my first performance evaluation, each index strategy answers a series of seven simple range queries, where each query operates on one of the scientific dataset's seven columns. In this dataset, the size of each subsequent column increases: 50 million rows, 100 million rows,..., 350 million rows.

In these experiments, I expect both the CPU and GPU-based DP-BIS index to answer a simple range query using approximately 75% less time than either the CPU or GPU projection index strategies. I base this expectation on the fact that DP-BIS primarily utilizes 8-bit low-resolution data, whereas the projection index strategies utilize 32-bit full-resolution data. I additionally expect that the GPU-based DP-BIS index will be very competitive, with respect to computational performance, with the GPU projection index. This expectation is based on the fact that both strategies support the

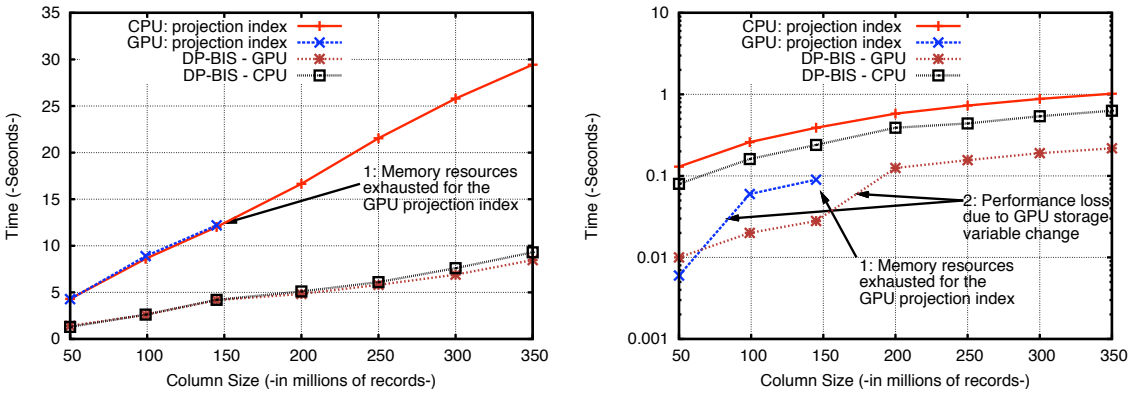


Figure 2.2: Here, (a) shows the total performance times for the three indexing strategies. In contrast (b) shows, based on the data from the same test series, *only* the processing performance time for each index. Side by side, these figures show how performance is affected by I/O plus computational workloads versus pure computational work.

same level of concurrency: a one-to-one mapping of threads-to-records.

Analysis

Figure 2.2(a) shows the realized total performance time of each index strategy. These performance times show that both DP-BIS implementations answer queries approximately 3X faster than both the GPU and CPU projection index. Table 2.4 shows how these total performance times are composed based on I/O and processing performance. Note values in Table 2.4 represent the average I/O and processing performance times realized for each index strategy based on the performance observed for all columns. Table 2.4 confirms the majority of time spent answering a simple range query is used to transfer data; each index uses over 93% of their total performance time for I/O-related tasks.

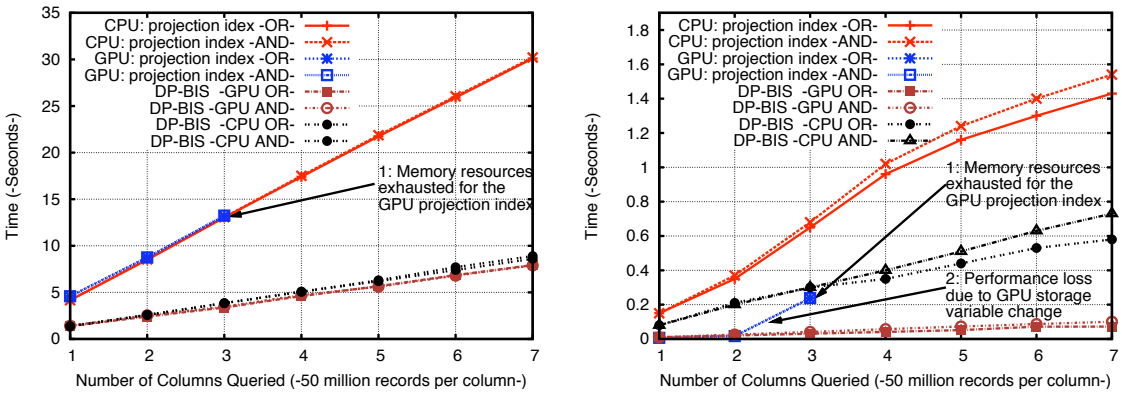
Note the GPU projection index and GPU-based DP-BIS index support the same level of concurrency when answering a simple range query. When performing this task I know both indexing strategies spend the vast majority of their time transferring data. I conclude that the disparity in total performance time experienced by the GPU projection index is directly attributable to an I/O-based performance bottleneck. This experiment illustrates the benefit of the encoding-based compression utilized by DP-BIS to accelerate the process of transferring data, and therefore the task of answering a selection query.

Aside from the performance benefits offered by DP-BIS, Figure 2.2(a) also highlights the benefits DP-BIS provides for GPU memory space. The GPU projection index exhausts all memory resources after columns have reached a size of 150 million rows. In comparison, DP-BIS is able to continue answering queries on columns until they reach in excess of 350 million rows. The data encoding DP-BIS utilizes thus provides over 233% better utilization of GPU memory resources when compared to the GPU projection index.

Figure 2.2(b) shows the processing performance times from my experiment; note that the scale of the y-axis is $\log_{10}$. Label 2 in Figure 2.2(b) highlights a sharp loss in performance for both the GPU-based projection index (between 50-100 million records) and DP-BIS (between 100-145 million records). This performance loss is due to a GPU implementation detail associated with how the query's solution is written for columns containing in-excess of 95 million (for the projection index) or 145 million (for DP-BIS) rows. Specifically, for columns whose row numbers exceed these values, the GPU projection index and DP-BIS can no longer store the query's solution with a 32-bit variable type (due to limited memory resources); instead an 8-bit variable type is utilized to conserve space. Writing 8-bit data to the GPU's global memory incurs significant performance penalties for both indexing strategies (as Label 2 highlights). Note however that based on the data in Table 2.4, this processing performance loss minimally impacts the total performance time for either of these two indexing strategies.

Figure 2.2(b) shows that before this performance loss, the GPU-based DP-BIS index answers queries up to 18X faster than the CPU projection index, 8X faster than the CPU-based DP-BIS index, and (for columns containing more than 95 million records) 3.4X faster than the GPU projection index. After this loss in performance, the GPU-based DP-BIS index outperforms the CPU-based projection and DP-BIS index by 4.9X and 3X respectively.

The concurrency afforded by DP-BIS is evident in comparing the processing performance times for the CPU-based and GPU-based implementations. The GPU-based DP-BIS index answers the query up to 8X faster than the CPU-based implementation. This acceleration in processing performance is a direct consequence of the GPU's increased data parallelism over the multi-core CPU. Accelerated processing performance times are critical for many scientific applications, e.g. query-driven visualization (QDV) [18, 39, 96], where data can be presumed to be read once, cached by the OS, and queried repeatedly during analysis stages. In these applications, user workloads are



(a) (b)

Figure 2.3: Here, (a) shows the total performance times for my three indexing strategies when they perform a series of conjunction and disjunction tests. In contrast (b) shows, based on the data from the same test series, *only* the processing performance time for each index.

driven more by processing performance times, which make up a larger percentage of the analysis workload, then by disk access times. For these applications, GPU-based implementations of DP-BIS provide significant performance benefits.

2.6.2 Answering a Compound Range Query

In this second performance evaluation, I use both indexing strategies to answer seven separate compound range queries. The columns the queries evaluate are taken from the synthetic dataset, where each column contains 50 million rows. In this series of tests, each subsequent query will constrain an additional column; in the final test, each index strategy will answer a query constraining seven columns. In this experiment, I perform this series of tests twice: once where I find the intersection, and once where I find the union of all columns queried. I refer to these logic-based series of tests as conjunction ($X_1 \bigwedge X_2 \bigwedge \ldots \bigwedge X_7$), and disjunction ($Y_1 \bigvee Y_2 \bigvee \ldots \bigvee Y_n$) tests.

I expect to see some level of disparity in processing performance times between the conjunction and disjunction tests. This expectation is based on the fact that, in my GPU kernels, identifying the intersection between two records requires slightly more computational overhead than identifying their union. I additionally note on these tests that the GPU-based DP-BIS implementation will not require a variable change for the GPU's solution space. More specifically, 50 million rows is a comparatively small solution space and therefore DP-BIS will be able to utilize the more efficient 32-bit data type throughout the entire experiment. As a result, I expect DP-BIS will maintain

its performance trend and not suffer the performance drop highlighted by Label 2 in Figure 2.2(b) from the previous experiment.

Analysis

Figure 2.3(a) shows the total performance times of all index strategies for both the conjunction and disjunction tests. In both experiments, the DP-BIS implementations answer compound range queries some 3–3.7X faster than the projection strategies. Note that Figure 2.3(a) shows no disparity between the conjunction and disjunction tests; such performance disparities are processing based and are more easily revealed in the processing performance times, shown in Figure 2.3(b).

Figure 2.3(b) highlights several important trends. First, as expected, the conjunction tests require more time to answer than the disjunction tests: 5–7% more time for the CPU projection index, and 20–27% more time for the CPU and GPU-based DP-BIS index. This performance trend is not readily seen in the GPU projection index; the lack of data points, due to exhausted memory resources (Label 1), obscures this performance disparity. Label 2 in Figure 2.3(b) highlights the loss of performance experienced by the GPU projection index due to the variable type change made in the GPU's solution space (see Section 2.6.1). In comparison, DP-BIS does *not* require such a change to the solution pace. Unlike the experiments performed in Section 2.6.1 (see Figure 2.2(b)), the smaller solution space employed by these tests (50 versus 350 million rows) enables DP-BIS to consistently use the more efficient 32-bit variable type. Finally, the processing performance benefits for a GPU-based implementation of DP-BIS are clearly seen in Figure 2.3(b); the GPU-based implementation of DP-BIS is 8X faster then the CPU-based implementation.

2.7 Summary

In the next decade, the evolution and predominance of multi-core architectures will significantly challenge and change the way data processing is done in the database community. As CPUs rapidly continue to become more like parallel machines, new strategies must be developed that can fully utilize the increasing thread-level parallelism, and thus the processing capabilities, of these architectures.

In presenting DP-BIS, I provide a parallel indexing data structure that will scale effec-

tively with the future increase of processor cores on multi-core architectures. Additionally, DP-BIS provides a means to drive a GPU-based Query-Driven Visualization (QDV) engine. More specifically, I can use DP-BIS to answer queries on the GPU and then leverage the GPU's tremendous processing capabilities to accelerate analysis and visualization tasks for QDV strategies. In Part III I demonstrate the benefits of such an engine by using DP-BIS to accelerate QDV strategies that analyze multitemporal, multiresolution data.

Part III

Extending Query-Driven Strategies for Time-Varying, Multiresolution Data

Chapter 3

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

3.1 Introduction

Query-Driven Visualization (QDV) strategies are well-suited for analyzing and visualizing large, highly-complex data [96, 110–112]. In this work, I address the challenges of using QDV strategies to visualize time-varying multiresolution data. Such data is used extensively in the scientific community to efficiently and accurately model many important physical phenomena. By extending QDV methods to address the dynamic spatiotemporal properties of time-varying multiresolution data, I provide scientists with a powerful tool for visually analyzing the data generated from numerical simulations.

3.2 Adaptive Mesh Refinement Strategies

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 understand-

ing 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. multi-temporal 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.

I present a two-step method for compositing and synchronizing AMR data from a series of timesteps. I 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. I then synchronize each timestep's grid hierarchy to the composite template. This approach enables my method to process queries on a common AMR grid hierarchy. Using this data structure, I move the work of query processing to the GPU to realize the benefit of greatly accelerated QDV analysis. On the GPU side, I integrate my new method with the DP-BIS index (detailed in full in Part II of this dissertation).

The main contributions of this work are the following.

- I 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.

- I 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, I discuss work germane to my efforts. This is followed in Section 3.4 by an overview of AMR grid fundamentals, my composite template construction and timestep synchronization process, and a description of how the DP-BIS index is integrated into a GPU-based QDV engine. Finally, I present the results of my method from both a qualitative and quantitative analysis perspective.

3.3 Previous Work

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

3.3.1 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.

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 [17], and then extended in 1989 [16], 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 [69] in his cell-sorting method for volume rendering. Norman et al. [77] 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 [68] describes a parallel rendering strategy for AMR data and presents two contrasting visualization approaches. Weber et al. [102, 104] 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. [103] also present crack-free isosurface extraction methods for AMR data. Park et al. [84] present a hierarchical multi-resolution splatting technique for AMR data that utilizes *k*d-trees and octrees. Their novel approach provides interactive performance for modest sized data.

Kähler and Hege [56] 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 [57]. Their method utilizes so-called "keyframe" timesteps to generate intermediate grid hierarchies. Data for these grid structures is acquired through interpolation strategies (e.g. Hermite or linear methods) by using the existing data in the keyframes.

Kähler et al. have more recently extended their earlier work by presenting a GPU-assisted raycasting strategy for accelerating the visualization of AMR data [58]. This work utilizes a *k*d-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.

3.3.2 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 [13, 45]. 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: $(1100 < temperature < 1800)$ AND $(pressure < 780)$. 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. [95] 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 [111, 112].

Gosink et al. [39] 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 [18]. They use compressed bitmap indices to visualize and characterize over 2.5 billion records of network connection data. Stockinger et al. [94] 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.

3.3.3 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. [47, 48] 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. [8] 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. [19] 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. [109] 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 [108] 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. I 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 my method allow for the interactive visualization of multivariate, spatiotemporal AMR data. I also extend previous QDV work in two aspects: I present the first application of QDV techniques on AMR data, and I 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.

3.4 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.

My 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.

I base my method on the AMR grid hierarchy outlined by Berger et al. [16, 17]. I outline this hierarchy and its properties next. For more details regarding AMR-based simulations, I refer the reader to the works of Almgren et al. [2, 16, 17].

3.4.1 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 [16]. I 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).

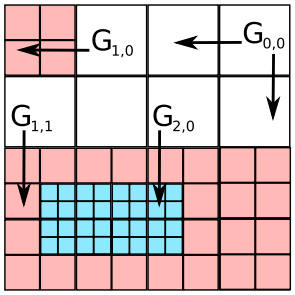


Figure 3.1: This image depicts an AMR grid hierarchy consisting of four grids and three levels of refinement: $G_{0,0}$, $G_{1,0\rightarrow1}$, and $G_{2,0}$. Grid cells are refined with a refinement ratio of r=2 and are properly nested: grid cells at level 2 do not abut grid cells at level 0.

$$\Delta x^{l+1} = \frac{1}{r}\Delta x^{l}, \quad \Delta y^{l+1} = \frac{1}{r}\Delta y^{l}, \quad \Delta z^{l+1} = \frac{1}{r}\Delta z^{l} \tag{3.1}$$

Note that the refinement ratio may change between successive grid levels: e.g., 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 [14]. This convention establishes a good balance between coding simplicity and an effective realization of refinement benefits.

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 3.1.

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) [2], or on factors that are more complex [16]. 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 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 [62]:

- 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 my method I 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, I am sure to be using the most accurate and detailed information produced by the computational model.

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:

$$\Delta t^{l+1} = \frac{1}{r}\Delta t^{l} \tag{3.2}$$

The time-stepping algorithm can be thought of as a recursive approach that advances grids cells in time according to their level of refinement [2]. 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 3.2 to determine the length of the timestep. At each of the r timesteps, calculate data values for these grid cells. Addition-

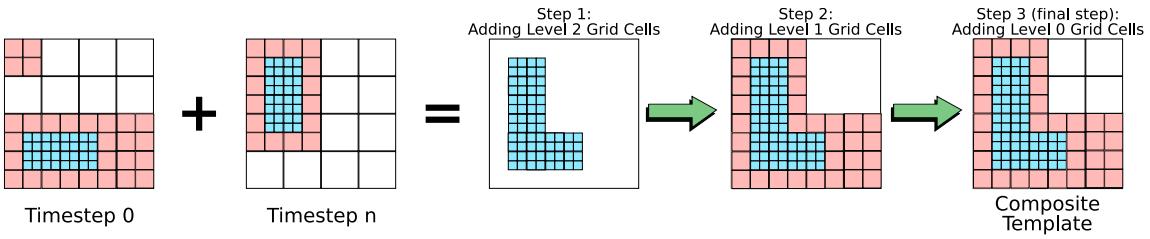


Figure 3.2: This figure illustrates the sequential process of compositing the AMR grid hierarchies of two selected timesteps. The process begins by filling the composite template with all grid cells, from both timesteps, of the finest level of refinement. In each subsequent pass, the procedure adds grid 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, I add grid cells of the coarsest level of refinement to the template.

ally, 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.

3.4.2 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. I achieve this consistency of refinement by first constructing a composite template from the series of AMR timestep's grid hierarchies. I then use this template to direct the synchronization of these grid hierarchies for purposes of QDV.

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

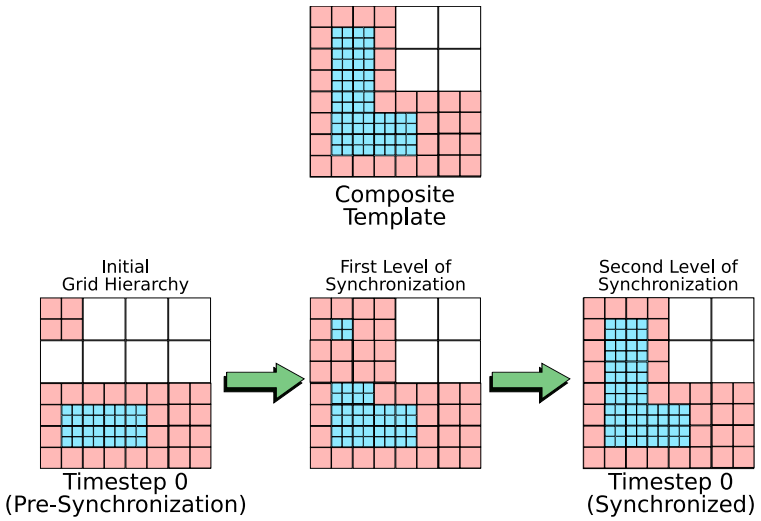


Figure 3.3: This figure depicts the sequential process of synchronizing the grid hierarchy of a given timestep with a composite template. At each level of synchronization, grid cells conditionally refine themselves by one additional level according to whether or not they are synchronized with the composite template. In this example, synchronization is complete for the grid hierarchy in the second level of synchronization.

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 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 3.2 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, thus preserving 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.

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, I propagate the cell centered value of the original grid cell to the centers of the newly created grid cells. Figure 3.3 illustrates this process. With each timestep's grid hierarchy synchronized, multitemporal query-driven visualization of AMR data is now possible.

3.4.3 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 users' queries, and render the results generated from these queries, as fast as possible. I meet these needs by employing the DP-BIS index (detailed in Part II) to answer multidimensional Boolean range queries. By utilizing a GPU-based index strategy, I can implement the entire QDV process on the GPU and accelerate QDV performance as a whole with the GPU's parallel processing power. In this implementation, the CPU serves as a host to the GPU, only streaming the minimal data necessary to perform full-resolution queries. All queries are evaluated (and rendered) on the GPU by executing kernels written in NVIDIA's data-parallel programming language CUDA [78]. QDV in literature typically evaluates scalar data. However, the DP-BIS index can also evaluate vector data, as well as an arbitrary number of timesteps or variables.

The integration of the DP-BIS index into QDV is similar to previous integrations that

utilized a CPU-based index [96]. Both strategies use index building, index searching, record processing, and rendering procedures. The difference between my work and previous work is that I query and render adaptively refined spatiotemporal data. This difference requires, in addition to the use of the composite template and synchronized grid hierarchies (Section 3.4.2), mapping query results to direct the rendering of grid cells during the rendering stage. I discuss DP-BIS index building, searching, and rendering next.

DP-BIS Index Construction

The DP-BIS strategy 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 DP-BIS index construction algorithm takes as input the full-resolution AMR data from a single timestep and generates both an encoded *and* a clustering-based, reordered 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 (i.e. encoded) version of the data. In the second stage it utilizes a data partitioning strategy, based on a modified Order-preserving Bin-based Cluster (OrBiC), to re-order and store all full-resolution data according to bin value.

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. DP-BIS 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-resolution 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 index construction process requires partitioning and re-ordering the original full-resolution data. To perform this, records are first partitioned according to their bin numbers. Next, these subsets of data, along with their corresponding row-ID numbers, are sequentially stored in a modified OrBiC table (Section 2.4.2). This operation is performed for all 256 bins. Once the second stage is completed, the modified OrBiC table contains a reordered,

sorted version (i.e. the records are sorted according to bin number) of the original full-resolution data. The total index size is approximately 2.25X the size of the original AMR data. This encoding and reordering of the data is essential to the search process as the next section details.

DP-BIS 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 partitions of data from the modified OrBiC table (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 I first determine the boundaries of the query. Consider an example. Given a user's range constraints on a given variable and timestep, such as ($pressure > 100$), I first determine the bins 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 partitions of data in the modified OrBiC table 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, as a maximum of $\frac{2}{256}$ the size of the original dataset is transferred over the bus (i.e. 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 full-resolution data and row-ID information in the data partitions to evaluate a full-resolution 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.

Mapping AMR-Based Bit-Vectors to Three-Space

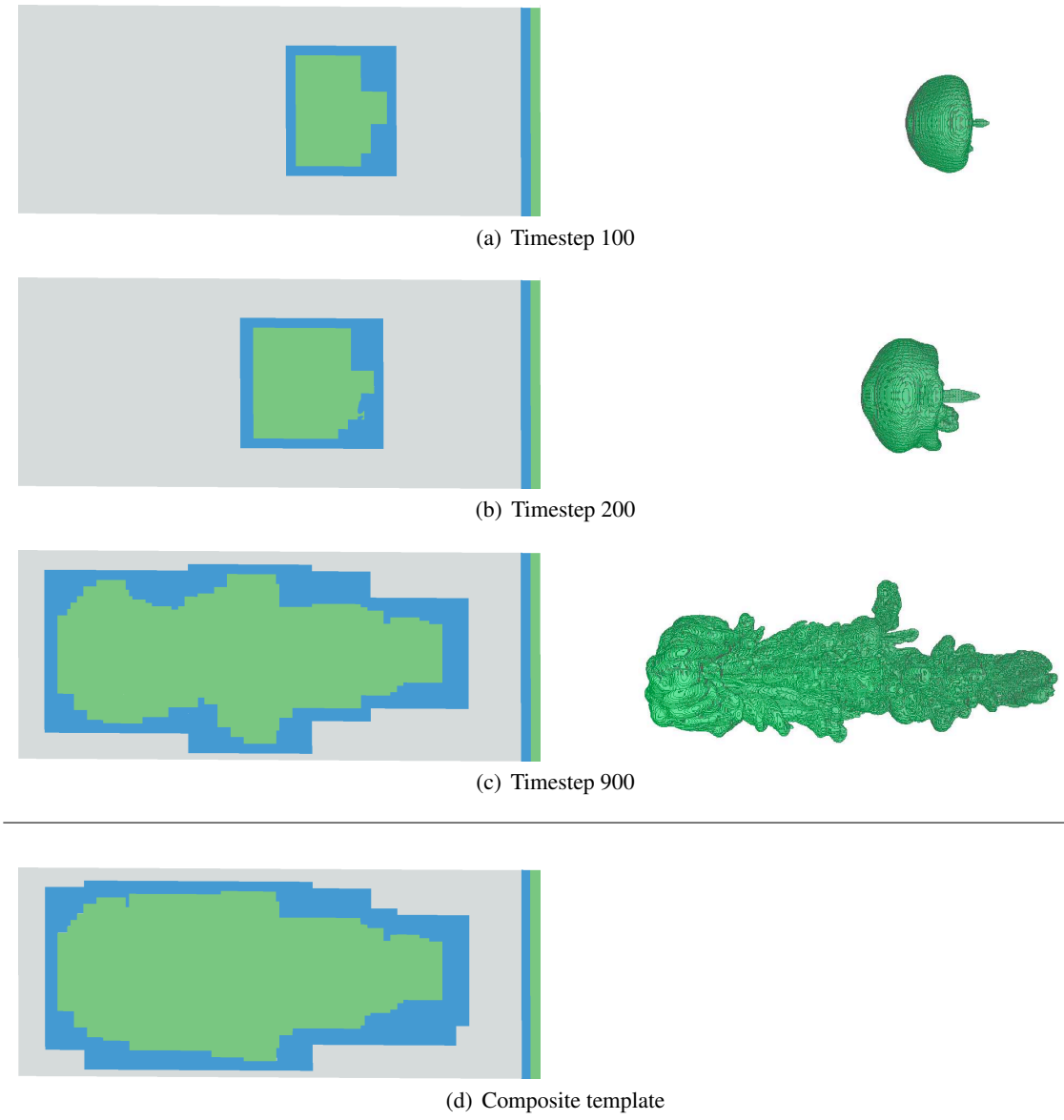
My rendering algorithm takes the bit-vector solution created in the index search stage, and generates (for cells passing the query) render-able coordinates for dynamically-sized hexahedral cells in three-space. For uniform datasets, bit-vectors are rendered by mapping each record's unique index to a 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.

My 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 DP-BIS index. Thus, the rendering stage in my 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.

3.5 Visualization Applications and Analysis

I apply my new query-driven visualization (QDV) method to two datasets. I demonstrate my 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 my 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



(a) Timestep 100

(b) Timestep 200

(c) Timestep 900

(d) Composite template

Figure 3.4: 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 I construct from all (18) timesteps I use in my 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.

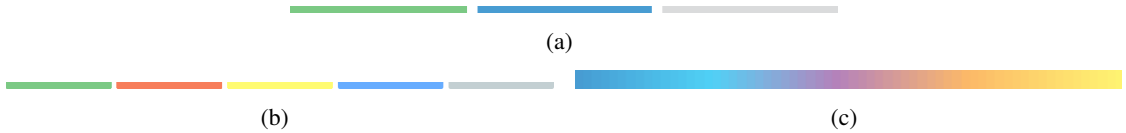


Figure 3.5: These images depict the three transfer functions I employ in my work. In (a) and (b) the colors correspond to levels of AMR grid refinement for the respective Argon Bubble and Hurricane datasets: green colors indicate grid cells of finest refinement, and gray colors indicate grid cells of 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.

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. I additionally used the following hardware:

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

3.5.1 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 I analyze is gas density; i.e. mass of gas per unit volume (ranging in values from 1.3 to 5.1). I 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 3.4(a) through 3.4(c) depict two-dimensional slices of selected AMR grid hierarchies; these are the grid hierarchies I use to construct my 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

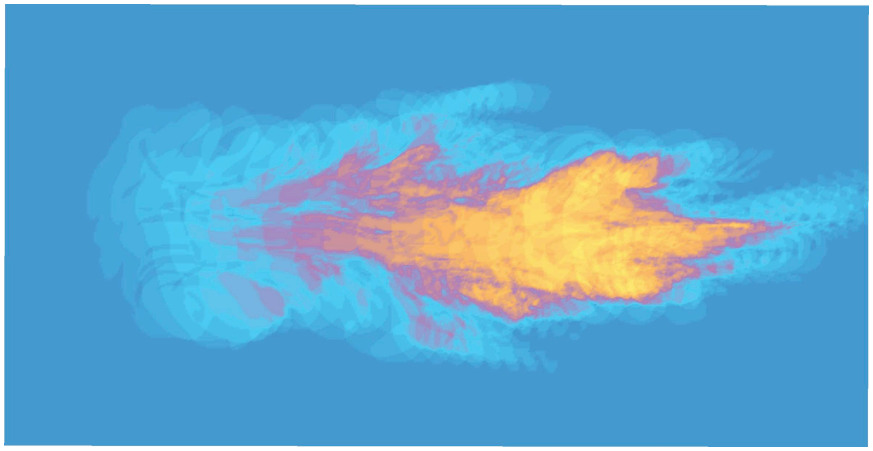


Figure 3.6: This multitemporal image depicts summary statistic information gathered from queries processed over 18 select timesteps from the Argon Bubble dataset. In this image, yellow regions indicate areas where high gas density predominantly resides over the course of the Argon Bubble simulation. Light blue regions show areas where only a few timesteps indicate the presence of high gas density; these regions denote where argon is dispersing.

grid refinement and are based on the transfer function shown in Figure 3.5(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 3.4(d).

I use this composite template to generate a multitemporal visualization based on summary statistic information accumulated from queries that select regions of high gas density. I 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. I 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. I apply the *same* query to the next timestep's synchronized AMR data (Timestep 150); results of this query too are added to the array. I 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.

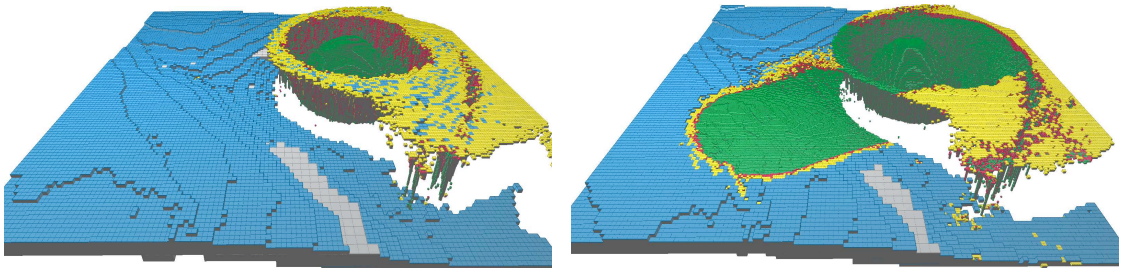
I visualize these summary statistics in Figure 3.6. This figure is colored according to the transfer function shown in Figure 3.5(c). Cells that contain low gas density over the entire length of the simulation (i.e. no query from any timestep indicated the cell passed the query for high density)

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 the query for density) are indicated in yellow. In this multitemporal visualization 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.

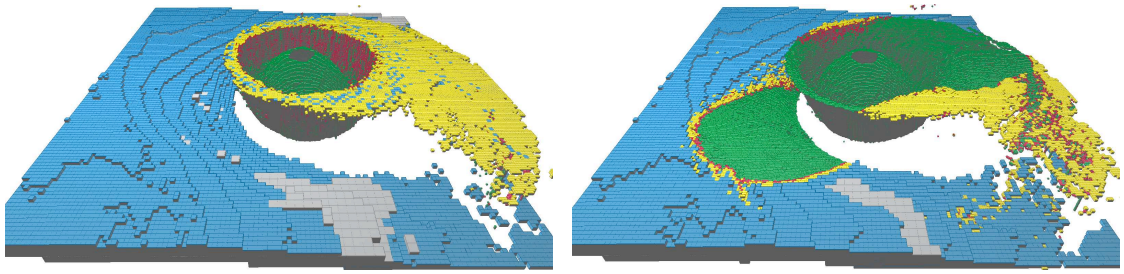
3.5.2 Dataset 2: Hurricane Isabel

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, I recast this uniform data in a multi-resolution framework to ease storage and visualization demands. I 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. I 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.

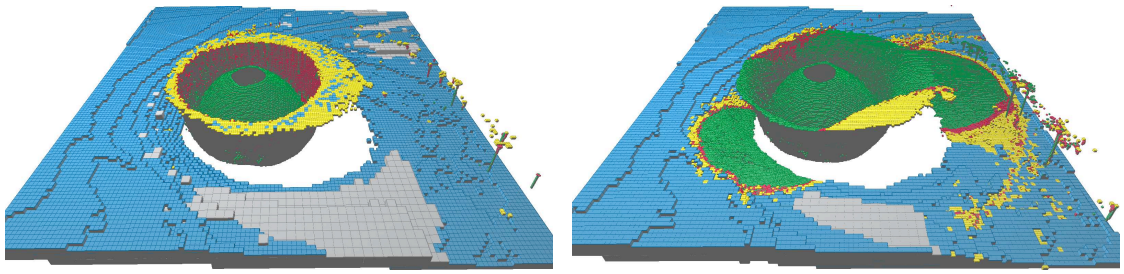
The results of this adaptive coarsening are a series of multi-resolution, time-dependent datasets that follow the structural properties of an AMR grid hierarchy. Each timestep in the original source data contains 25 million cells. After coarsening, each timestep contains 5 levels of refinement and a 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. I apply my method for generating multitemporal visualizations to these datasets; I 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 my new multitemporal visualization method, I effectively characterize these regions. In this section, all queries for pressure are in Pascal units.



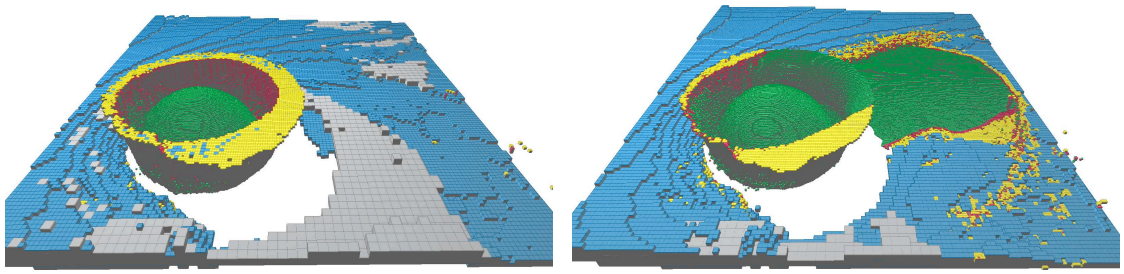
(a) Timestep 10



(b) Timestep 25

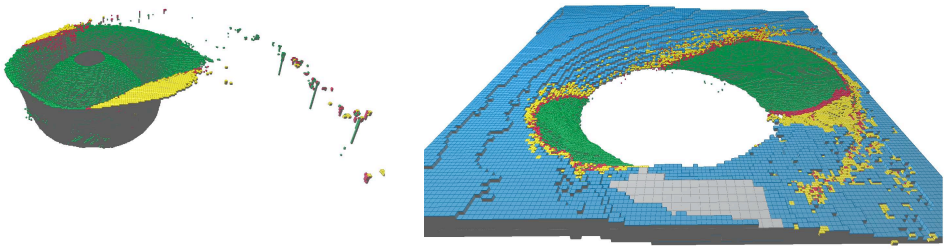


(c) Timestep 35



(d) Timestep 45

Figure 3.7: This series of images, selected from 48 timesteps, compares query results from non-synchronized (left column), and synchronized (right column) AMR grids of the Hurricane Isabel dataset. The query used on each timestep consists of two parts; I query for regions of low pressure $(-200 \leq pressure \leq 20)$ OR regions of high pressure $(500 \leq pressure \leq 1000)$. To assist in interpreting these images, the individual regions generated from these low and high pressure queries are shown in Figures 3.8(a) and 3.8(b).



(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: $(500 \leq pressure \leq 1000)$

Figure 3.8: This figure depicts grid cells in the Hurricane dataset that contain relatively low (a) and high pressure (b).

Multitemporal Visualization

Figure 3.7 depicts select non-synchronized (left column) and synchronized (right column) timesteps from the hurricane dataset. The top row illustrates the individual grid hierarchies generated from my 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 $(500 \leq pressure \leq 1000)$. To assist in interpreting the data in Figure 3.7, I show the regions of isolated low pressure in Figure 3.8(a) and isolated high pressure in Figure 3.8(b).

The regions in Figure 3.7, as well as those in Figure 3.8(a) and Figure 3.8(b), are colored according to the transfer function in Figure 3.5(b); green regions indicate grid cells of finest refinement, and gray regions indicate grid cells of coarsest refinement. In both columns of images in Figure 3.7 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 my adaptive coarsening method. In the right column of images in Figure 3.7, which show results for querying timesteps of composited and synchronized AMR grids, the path of the hurricane is preserved at the finest level of refinement in the composite template as indicated by the green path.

I utilize this composite template to generate a multitemporal visualization based on summary statistics from queries—processed over each of the simulation's 48 timesteps—that select

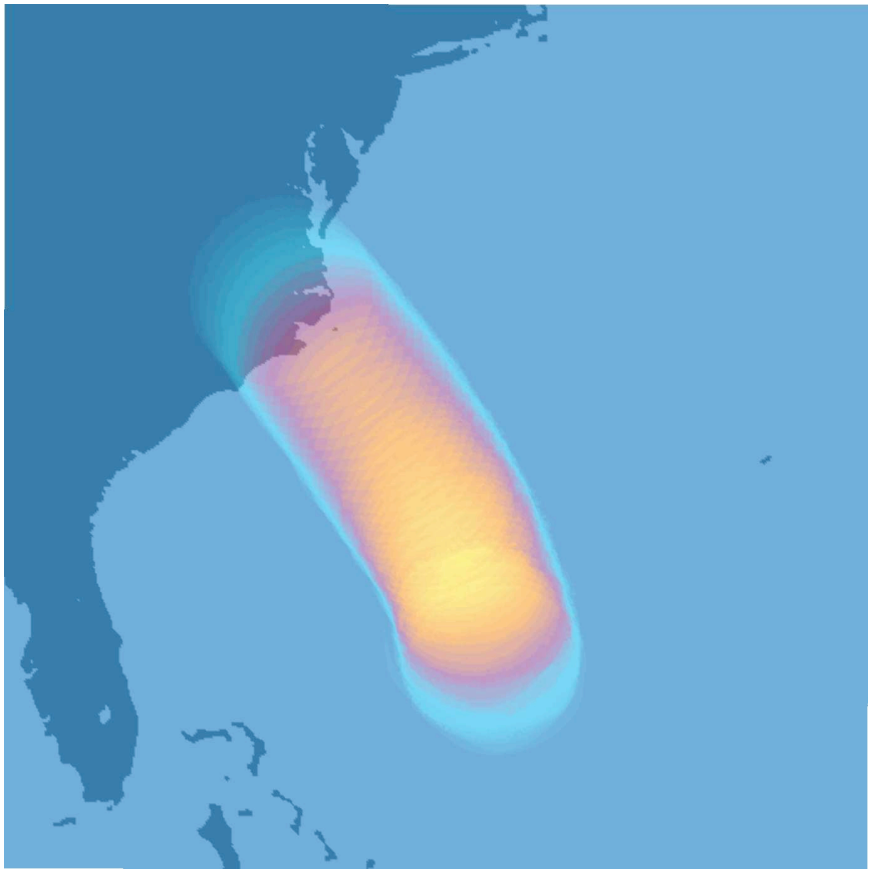


Figure 3.9: This multitemporal image depicts summary statistic information gathered from queries processed over 48 timesteps from the Hurricane dataset. In this image, yellow regions indicate where low pressure predominantly exists across the 48 timesteps.

regions of low pressure. This process is analogous to the one performed in Section 3.5.1 for the Argon Bubble dataset. I begin by querying the synchronized AMR data of the first timestep (Timestep 1) for all cells with pressure values in the range of $(-200 \leq \textit{pressure} \leq 20)$—note that this query characterizes regions of hurricane activity in the numerical 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 the queries for low-pressure, evolves spatiotemporally.

I visualize these summary statistics in Figure 3.9. This figure is colored according to the transfer function shown in Figure 3.5(c). In this multitemporal visualization, cells that predominantly contain low pressure over time—indicated in yellow—define the path of the hurricane.

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 3.1: This table depicts, for an increasing number of timesteps and records queried, performance times taken during the analysis of the Hurricane dataset (Section 3.5.2). 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 encapsulates the two previous times.

Performance

There are two factors that contribute to the response time of a QDV application: the time it takes to process a query (Section 3.4.3), and the time it takes to render the query's solution (Section 3.4.3). To the user, these two times appear as a unified sum that I define as "query-driven response time". I analyze the performance of my work by presenting the query-driven response times in terms of the number of queries I can process and render in a single second. I additionally show this metric in terms of its two principal 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. I evaluate my method's performance by independently analyzing two query characteristics important to QDV. I 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.

I begin the tests by querying a single timestep with queries that select 1%, 10%, and 20%

of the cells as hits. As mentioned, I 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 I process and render in a single second. I then consider an additional timestep in the queries, and repeat the same sequence of tests. I repeat this process until a total of five timesteps are simultaneously evaluated by all queries. The results of these tests are shown in Table 3.1.

The values shown in Table 3.1 indicate that performance for the 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 my 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 3.1), my 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.

3.6 Summary

I have presented a new method for performing query-driven visualization of time-varying AMR data. With this new analysis and visualization approach, scientists can construct multitemporal visualizations that convey, in a single image, how queries characterizing important interactions or properties evolve over time.

As I have shown, well-characterized range queries are capable of identifying temporal and spatial regions where many domain-specific events occur: e.g. the path of a hurricane's vortex region (Section 3.5.2) or the leading front of a shock-wave traveling through an inert atmosphere. (Section 3.5.1). Beyond indicating these regions, however, queries reveal little about the variable interactions or complex trends that lie in the domain of these characterizations. In the next part of this dissertation, Part IV, I present two statistical-based methods for generating greater insight and understanding into the solution space of a user's query.

Part IV

Integrating Statistical Analysis Methods with Query-Driven Visualization

Chapter 4

Variable Interactions in Query-Driven Visualization

4.1 Introduction

Obstacles hindering scientific research may be broadly categorized into two separate but overlapping groups. The first category, concerned mainly with issues of throughput, includes the challenges inherent to efficiently managing and visualizing large-scale datasets. The second category includes the difficulties associated with attaining insight from datasets of high-complexity.

Query-driven visualization (QDV) is well-suited for performing analysis and visualization on datasets that are both large *and* highly complex [111, 112]. Tools like FastBit leverage highly efficient (in terms of speed and compression) data management techniques to rapidly identify and visualize "regions of interest" within a dataset [41, 94–96]. Specified with Boolean range queries (e.g. ($Methane < 0.25$M) AND (760 Torr $\leq Pressure \leq$ 820 Torr)), these regions of interest tend to be significantly smaller subsets of the original dataset; thus, these regions require less time and computational effort to analyze, visualize, and interpret.

Well-characterized range queries are capable of identifying spatial regions where many domain-specific events occur: combustion flame fronts, vortices, chemical reaction fronts, etc. Beyond indicating these regions, however, queries reveal little about variable interactions or complex trends that lie in the domain of these characterizations. In such regions of interest, it is the behav-

ioral trends *between* variables, or groups of variables, that are more important in providing insight than the traits or locations of individual variables alone. The challenge is to extend the strengths of QDV with methods that identify behavioral trends and provide insight into regions of interest through coherent and meaningful visualizations.

The novel contributions of this work are techniques that extend the capabilities of QDV by providing intuitive insight in determining:

- how sets of variables in complex datasets interact throughout regions of interest, and
- the role other variables play in influencing these interactions.

I utilize the cumulative distribution functions (CDFs) of all variables in a query, to reveal initial information about statistical regions of interest within the query's solution space. The CDF for each variable is formed by integrating over the query's solution space and accumulating the variable's values as a histogram. Statistically, the solution set of a query is represented as an aggregate of histograms, one histogram for each variable expressed in the query.

I extend this analysis further by incorporating the use of correlation fields. Correlation fields provide insight into *localized* correlation between any two variables. By mapping a correlation field onto a third variable's isosurfaces (specifically, the statistically important isovalues suggested by the variable's CDF), statistically important interactions between any three variables in a dataset are readily visualized, allowing for trends between variables in a user's query to be identified.

In this method, CDFs and correlation fields are constrained to the query's *solution space*. By working exclusively in the query's solution space, this method takes full advantage of the performance benefits inherent to QDV strategies. Specifically, computational efforts are only focused on regions that have been rapidly identified (via a query engine) as "interesting" by the user's query. This method's integrated analysis extends current query solutions by revealing statistical trends of interactivity (i.e. dependency and independence) between any triad of variables in the solution space of the query.

I discuss research related to my efforts in the next section. Section 4.3 defines the terminology used to describe multivariate queries. I utilize this terminology, in Section 4.4, to describe my method. In Section 4.5, I present a detailed discussion of variable interactions in the context of flame-front analysis.

4.2 Previous Work

Numerous methods have been developed for the management and visualization of large datasets. Some of the more effective software solutions include *local adaptive mesh refinement* (AMR) [16], and the bitvector compression techniques used in query based strategies [112].

Based on adaptive numerical discretization techniques used to approximate solutions for partial differential equations (PDE), AMR algorithms work on the principal that many physical phenomena (e.g. second-order systems such as wave equations, particles interfaces, radiative heat transfer, charged-fluid plasma models, low-Mach combustion, etc.) exhibit variations in scale [16]. Standard numerical solutions require the discretization of the phenomenon's domain space to be within an acceptable margin of error (as determined by the algebraic analogues of the PDE). In many cases, due to the need for high resolution (i.e. due to discontinuities), the discretization becomes very dense and costly.

Rather than creating a uniform grid of high density, AMR begins with a relatively course grid and adaptively refines the mesh resolution *only* in regions requiring higher levels of accuracy. The range of resolution is usually fixed with an assigned initial coarse resolution. Regions requiring further detail are refined recursively until either the maximum level of refinement is reached, or the desired level of accuracy has been achieved. For a single simulation, numerous levels of refinement are possible. Advantages of AMR include dramatically reduced storage requirements and computational analysis (on a per-node basis).

Strategies based upon QDV are designed on the premise that smaller subsets of data are usually the genesis of insight or breakthroughs to new trends [13, 45]. The goal of query-driven analysis is the rapid isolation and visualization of "interesting data" (as determined by user dictated Boolean range queries) from the dataset. One of the most daunting challenges in data analysis is the rapid identification and retrieval of these records. Efficient compressed bitmap index technologies, like FastBit [111, 112], achieve this goal through compression and accelerated algorithms that allow for the rapid retrieval of queried records.

The field of multivariate visualization has also been actively researched. Scatterplot matrices are perhaps the most direct visualization of correlation between multiple variables [107]. The primary limitation of scatterplot matrices lies in the fact that scatterplots show correlations in data

range space, but do not offer the ability to see such correlations in the native spatial coordinates of the dataset.

Methods for determining correlation have been well studied, and there are many approaches established for measuring correlation between pairs of scalar variables [52]. The use of scalar field gradients as a comparison measure of correlation [31] was recently used in work that utilized Multifield-graphs [89]. In that work, correlation between sets of variables (i.e. fields) was visualized with a graph structure.

Parallel coordinates are a compact, two-dimensional visualization of trends between multiple variables [53]. The concept of parallel coordinates was extended by Fua et al. [33] to support large datasets, clustering, and interactive brushing. Blended texture approaches for two-dimensional scalar fields have also been explored with the goal of compactly conveying as much information as possible about variable trends on a single surface [100, 101].

While two-dimensional visualizations of multivariate correlation are common, three-dimensional and time-varying visualizations are comparatively rare. Akiba et al. [1] have performed initial work in the area of higher dimensional multivariate rendering by utilizing a parallel coordinates-based transfer function widget to direct the rendering of multivariate and time-varying datasets.

There is still a need for methods, applicable to large multivariate data, that elucidate how variables behave and interact within regions of interest. In this chapter, I combine CDFs and correlation fields to extend the solutions generated by QDV approaches. With this method I am able to evaluate (with respect to the query's solution space) any three variables, and the interactions between these variables, in a meaningful way.

4.3 Cumulative Distribution Functions and Correlation Fields

In this section I formalize two principal statistical concepts used in my method. The first portion of this section defines the CDF construction performed for each variable used in a given query. The second part of this section explains the process for creating correlation fields between two variables. Correlation fields are used, with the CDFs of the query's variables, to visually evaluate any three variables, and the interactions between these variables, within the solution space of the query.

4.3.1 The Univariate Universe

Let a finite universe $\mathscr{U}$ be defined such that $\mathscr{U} \subsetneq \mathbb{R}^3$. Additionally, let $\mathscr{U}$ be defined to be univariate; there exists exactly one mapping function, **f**, considered valid on $\mathscr{U}$ that maps each of $\mathscr{U}$'s elements $\mathbf{f} : \mathbb{R}^3 \rightarrow \mathbb{R}$. This is the usual definition for a scalar field dataset in $\mathbb{R}^3$ space.

A common notion in such a universe is that of the level set. In $\mathscr{U}$, level sets are defined implicitly through the mapping function **f**. Thus, the explicit level set for any real value **a** is equivalent to the exhaustive solution of the *inverted* mapping function when evaluated at **a** (i.e. $\mathbf{f}^{-1}(\mathbf{a})$). Notationally, $\mathscr{S}_{\mathbf{f}\rightarrow\mathbf{a}}$ denotes the explicit level set formed for the real value **a** by the mapping function **f** in the universe $\mathscr{U}$:

$$\mathscr{S}_{\mathbf{f}\rightarrow\mathbf{a}} = \{\mathbf{x} : (\mathbf{x} \in \mathscr{U}) \wedge (\mathbf{f} : \mathbf{x} \rightarrow \mathbf{a})\} \tag{4.1}$$

When $\mathscr{U} \subsetneq \mathbb{R}^2$ or $\mathscr{U} \subsetneq \mathbb{R}^3$, these level sets are used to construct isocurves and isosurfaces respectively.

Without loss of generality, I assume that a finite number of level sets $(\mathscr{S}_{\mathbf{f}\rightarrow\mathbf{a}}, \ldots, \mathscr{S}_{\mathbf{f}\rightarrow\mathbf{n}})$ are generated by **f**, and that the cardinality of these sets is also finite. I observe that the mapping function **f** may then be represented by the discrete random variable $\mathscr{X}_{\mathbf{f}}$. This random variable takes any real value $(\mathbf{a}, \ldots, \mathbf{n})$ from the possible functional mappings of **f**, and thus describes the distribution behavior of the real-valued function **f** over the sample space of $\mathscr{U}$. Appropriately, a *probability mass function* (*PMF*) for $\mathscr{X}_{\mathbf{f}}$, denoted $p_{\mathscr{X}_{\mathbf{f}}}$, may be constructed:

$$p_{\mathscr{X}_{\mathbf{f}}}(\mathbf{a}) = P(\mathscr{X}_{\mathbf{f}} = \mathbf{a}) = \frac{\|\mathscr{S}_{\mathbf{f}\rightarrow\mathbf{a}}\|}{\|\mathscr{U}\|} \tag{4.2}$$

Here $P(\mathscr{X}_{\mathbf{f}} = \mathbf{a})$, or more generally P(**E**), denotes the probability of event **E** occurring. Specifically, Equation 4.2 states the probability that the mapping function **f** will map a randomly selected element in $\mathscr{U}$ to the real value **a**. Combining all *PMF*s for a single random variable forms the appropriate *CDF*, denoted $F_{\mathscr{X}_{\mathbf{f}}}$, for $\mathscr{X}_{\mathbf{f}}$:

$$F_{\mathscr{X}_{\mathbf{f}}}(\mathbf{n}) = P(\mathscr{X}_{\mathbf{f}} \leq \mathbf{n}) = \frac{\|\bigcup_{i=\mathbf{a}}^{i=\mathbf{n}} \mathscr{S}_{\mathbf{f}\rightarrow\mathbf{i}}\|}{\|\mathscr{U}\|} \tag{4.3}$$

Here Equation 4.3 states the probability that the mapping function **f** will map a randomly selected

element in $\mathscr{U}$ to any real value between **a** and **n** (inclusive).

4.3.2 The Multivariate Universe

I extend this definition of a random variable to the case of multivariate universes where there exist multiple valid mapping functions $(\mathbf{f}, \mathbf{g}, \ldots, \mathbf{k})$ that map each of $\mathscr{U}$'s elements $(\mathbf{f}, \mathbf{g}, \ldots, \mathbf{k} : \mathbb{R}^3 \rightarrow \mathbb{R})$. As with the univariate universe, a unique random variable, $\mathscr{X}_{\mathbf{f}}, \mathscr{X}_{\mathbf{g}}, \ldots, \mathscr{X}_{\mathbf{k}}$, may be created for each valid mapping function that describes the distribution behavior of this real-valued function over the sample space of $\mathscr{U}$.

For each of these random variables, a *joint PMF* and *CDF* may be defined. For example, given real constraints for random variable $\mathscr{X}_{\mathbf{g}}$ in a multivariate universe $\mathscr{U}$, the joint mass and distribution functions of $\mathscr{X}_{\mathbf{f}}$ are expressed:

$$p_{\mathscr{X}_{\mathbf{f}}|\mathscr{X}_{\mathbf{g}}}(\mathbf{a}|\mathbf{c} \leq \mathscr{X}_{\mathbf{g}} \leq \mathbf{d}) = \frac{\|\mathscr{S}_{\mathbf{f}\rightarrow\mathbf{a}} \cap \bigcup_{i=\mathbf{c}}^{i=\mathbf{d}} \mathscr{S}_{\mathbf{g}\rightarrow\mathbf{i}}\|}{\|\mathscr{U}\|} \tag{4.4}$$

$$F_{\mathscr{X}_{\mathbf{f}}|\mathscr{X}_{\mathbf{g}}}(\mathbf{n}|\mathbf{c} \leq \mathscr{X}_{\mathbf{g}} \leq \mathbf{d}) = \frac{\|\bigcup_{i=\mathbf{a}}^{i=\mathbf{n}} \mathscr{S}_{\mathbf{f}\rightarrow\mathbf{i}} \cap \bigcup_{i=\mathbf{c}}^{i=\mathbf{d}} \mathscr{S}_{\mathbf{g}\rightarrow\mathbf{i}}\|}{\|\mathscr{U}\|} \tag{4.5}$$

Observe that Equations 4.4 and 4.5 above may be extended such that the conditioning of random variable $\mathscr{X}_{\mathbf{f}}$ includes any additional random variables ($\mathscr{X}_{\mathbf{h}}, \ldots, \mathscr{X}_{\mathbf{k}}$) that define distributions of valid mapping functions on $\mathscr{U}$.

4.3.3 Multivariate Queries

I define a Boolean query, **q**, as a composite function that operates on those mapping functions considered valid on $\mathscr{U}$. More formally, $\mathbf{q} : (\mathbf{f} : \mathbb{R}^3 \rightarrow \mathbb{R}) \rightarrow \mathbb{B}$, resulting in the solution set for the query, denoted as $\mathscr{Q}$:

$$\mathscr{Q} = \{\mathbf{x} : (\mathbf{x} \in \mathscr{U}) \wedge (\mathbf{q} : (\mathbf{f} : \mathbf{x} \rightarrow \mathbb{R}) \rightarrow 1)\} \tag{4.6}$$

Note that if $\mathscr{U}$ is multivariate, the query function may extend to include any of the possible valid mapping functions for $\mathscr{U}$.

The solution set $\mathscr{Q}$ is a reduced sample space of $\mathscr{U}$. Appropriately, the level sets valid

in $\mathscr{Q}$ for each mapping function will also be reduced (to some degree) from their unconditioned counterparts. These reduced level sets visually represent the conditional reduction imposed by the constraints of the composite query function $\mathbf{q}$. For my purposes I restrict such general query functions to be Boolean range queries (e.g. "$X < 100$, $X > 10$, or $0 < X < 10$") where continuous regions of $\mathscr{U}$ are defined to be in $\mathscr{Q}$.

4.3.4 Correlation Fields

The differential operator $\bigtriangledown$ acts as a composite function that can take any mapping function valid on $\mathscr{U}$ to construct a gradient field in $\mathbb{R}^3$.

Given two mapping functions (for illustrative purposes, $\mathbf{f}$ and $\mathbf{g}$), represented by respective random variables $\mathscr{X}_\mathbf{f}$ and $\mathscr{X}_\mathbf{g}$, a scalar field indicating the localized correlation measured between these two variables may be constructed by observing the cosine of the angle between the two corresponding gradient fields generated by $\bigtriangledown\mathbf{f}$ and $\bigtriangledown\mathbf{g}$. In my method, this measurement of correlation is taken by observing the *normalized* dot product between the two gradient fields. Using random variables $\mathscr{X}_\mathbf{f}$ and $\mathscr{X}_\mathbf{g}$, the correlation field is represented:

$$\mathbf{C}_{\mathscr{X}_\mathbf{f}\mathscr{X}_\mathbf{g}} = \bigtriangledown\mathbf{f} \cdot \bigtriangledown\mathbf{g} \tag{4.7}$$

Correlation coefficients (being summary statistics) and scatterplots are only indicators of "global" trends. Correlation fields, on the other hand, are useful for the spatial and "local" analysis of correlation; they identify regions, restricted to a particular query's solution set, where important variable interactions and trends take place.

4.4 Method

In my method, I utilize CDFs in conjunction with correlation fields to elucidate how variables in a given query interact. Given the solution set $\mathscr{Q}$ for a specific Boolean range query in a multivariate universe, I conduct my analysis in the following manner. I first begin by determining the individual "contributions" to $\mathscr{Q}$ made by each variable in the query. This is performed by integrating over the reduced sample space of $\mathscr{Q}$, and recording each variable's values in a histogram

(one histogram per variable).

Next, these variables are classified based upon their respective histograms. This classification is performed by sorting the entries in each variable's histogram by frequency of occurrence in $\mathscr{Q}$. From the definitions in Sections 4.3.1 and 4.3.2, I observe that this is equivalent to defining each variable's distribution in $\mathscr{Q}$ as a random variable, and sorting all possible values for these random variables based upon probability (as expressed in Equation 4.4). Additionally, I observe that these histogram entries are represented by level sets, and thus isosurfaces, in $\mathscr{Q}$. Continuing with this notion of level sets, the following general classification is performed:

- Level sets for a given variable with a probability greater than one standard deviation above the mean of the variable's other level set probabilities are labeled as *principal level sets*. Principal level sets make up the majority of the query's solution. These level sets offer the most potential for gaining insight, when used in conjunction with a correlation field, as they convey the most representative behavior of the variable in the reduced sample space of $\mathscr{Q}$.
- All other level sets are *secondary level sets*.

After determining the principal level sets, any two variables of interest are selected to construct a correlation field—performed by observing the *normalized* dot product between two generated gradient fields as shown in Equation 4.7. Note that the chosen variables do not have to be part of the query, merely constrained by it. Working exclusively in the query's solution space allows for this method to take full advantage of performance benefits inherent to QDV strategies. Specifically, computational efforts are only focused on regions that have been rapidly identified (via a query engine) as "interesting" by the user's query.

Principal level sets are then rendered through this correlation field as isosurfaces and "colored" with the transfer function corresponding to the correlation field generated from the two previously selected variables. This method of exploration offers several levels of insight. Consider two variables $\mathscr{X}_{\mathbf{f}}$ and $\mathscr{X}_{\mathbf{g}}$. Let $\mathbf{C}_{\mathscr{X}_{\mathbf{f}}\mathscr{X}_{\mathbf{g}}}$ be the correlation field between $\mathscr{X}_{\mathbf{f}}$ and $\mathscr{X}_{\mathbf{g}}$. To explore the interaction of a third variable $\mathscr{X}_{\mathbf{h}}$ with $\mathbf{C}_{\mathscr{X}_{\mathbf{f}}\mathscr{X}_{\mathbf{g}}}$, I render isosurfaces (from respective principal level sets) of $\mathscr{X}_{\mathbf{h}}$ colored by scalar values from $\mathbf{C}_{\mathscr{X}_{\mathbf{f}}\mathscr{X}_{\mathbf{g}}}$. Below, I discuss two behaviors that may be observed in this type of visualization (additional behaviors are addressed in Section 4.5):

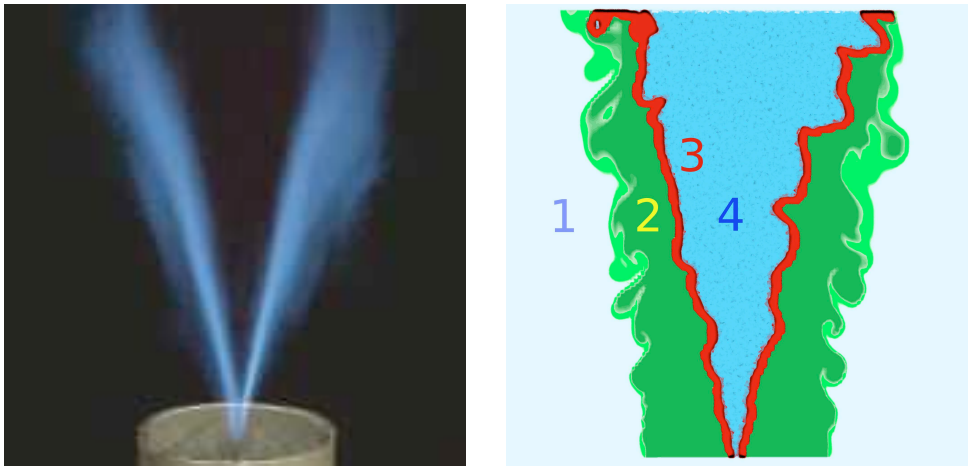
- **Behavior:** Isosurfaces of $\mathscr{X}_{\mathbf{h}}$ are consistently multi-colored, with positive, zero, *and* negative correlation intermixed across the surface. **Or**, alternatively, isosurfaces of $\mathscr{X}_{\mathbf{h}}$ are predominantly and consistently colored by *either* positive, zero, *or* negative correlation.

  Interpretation: This type of behavior indicates little correlation between $\mathscr{X}_{\mathbf{h}}$, and the correlation field $\mathbf{C}_{\mathscr{X}_{\mathbf{f}}\mathscr{X}_{\mathbf{g}}}$. If varying $\mathscr{X}_{\mathbf{h}}$ consistently shows *either* no change in the correlation between $\mathscr{X}_{\mathbf{f}}$ and $\mathscr{X}_{\mathbf{g}}$, *or* "random" changes, it is difficult to infer any relationship between $\mathscr{X}_{\mathbf{h}}$ and $\mathbf{C}_{\mathscr{X}_{\mathbf{f}}\mathscr{X}_{\mathbf{g}}}$.

- **Behavior:** Isosurfaces of $\mathscr{X}_{\mathbf{h}}$ are colored predominantly with a single correlation value (e.g. positive, zero, *or*, negative). As isosurface values for $\mathscr{X}_{\mathbf{h}}$ change, the predominant correlation value coloring the surface changes.

  Interpretation: This behavior indicates the possible existence of a scientifically-important relationship between $\mathscr{X}_{\mathbf{h}}$ and $\mathbf{C}_{\mathscr{X}_{\mathbf{f}}\mathscr{X}_{\mathbf{g}}}$. One explanation of this behavior may be that variable $\mathscr{X}_{\mathbf{h}}$ influences the correlation between $\mathscr{X}_{\mathbf{f}}$ and $\mathscr{X}_{\mathbf{g}}$. Alternatively, the correlation between $\mathscr{X}_{\mathbf{f}}$ and $\mathscr{X}_{\mathbf{g}}$ may effect the variable $\mathscr{X}_{\mathbf{h}}$.

Using this visualization method, it is instructive to look for isosurface regions corresponding to near-zero correlation. Unlike areas of positive or negative correlation (low entropy, high mutual information), near-zero correlation regions correspond to areas of high entropy and low mutual information. The Intermediate Value Theorem may be used to show that to move from an area of positive correlation to an area of negative correlation, or vice versa, one must pass through a point of zero correlation. As with any system, the corresponding increase in entropy must be caused by external events or internal processes. If large parts of an isosurface of one variable coherently follow such a transformation between two other variables (positive to negative, or negative to positive), then it is likely that the isosurface rendered at the point of near-zero correlation between the two other variables is important to the external event or internal process causing the entropy change. This method's utility is its ability to identify the transitions (i.e. areas of maximal entropy) that indicate regions of important change or variable interactivity in a user's query.



(a) Photographed Flame (b) Illustrated Flame

Figure 4.1: These images show photographed (a) and illustrated (b) methane combustion. The illustration, at right, also labels the principal regions of the methane flame involved in the combustion process. The outer region (1) corresponds to the atmospheric environment, around the methane fuel source, consisting of nitrogen, oxygen, and small amounts of water. The mixing region (2) contains the fuel source and thus the highest concentration of methane. The reaction region (3) is where the "flame fronts" and a diverse collection of chemical intermediates reside. The product region (4) contains the highest concentration of water and carbon dioxide—the final products in the combustion process.

4.5 Application and Analysis

I apply the techniques developed in Sections 4.3 and 4.4 to two combustion datasets. The first dataset models the combustion of a methane flame, while the second dataset models the turbulent combustion of an ultra-lean premixed hydrogen flame.

In the study of combustion, it is important to understand flame anatomy. Figure 4.1 shows photographed and illustrated methane combustion. In the illustrated image, important regions of the flame are labelled:

1. The *outer region*, in light blue, corresponds to the atmospheric environment, found around the fuel source, consisting of nitrogen, oxygen, and trace amounts of water.

2. The *mixture region*, in green, corresponds to the region in which the fuel source, and thus the highest concentration of methane, exists. Note that in this region, oxygen and other atmospheric gasses also still exist.

3. The *reaction region*, in red, is the most complex region chemically. In this region, "flame

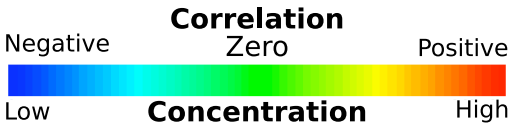


Figure 4.2: Colormap used for pseudocoloring both correlation and scalar concentration fields. Blue corresponds to negative correlation within correlation fields, green to very little or no correlation, and red to positive correlation. For scalar concentrations, the lowest concentration values map to blue, while higher values map to red.

fronts" arise along the leading fronts of fuel and oxygen combustion.

4. The *product region*, in blue, contains the highest concentration of both water and carbon dioxide—the final products of methane combustion.

The above information also generalizes to hydrogen combustion flames. In both methane and hydrogen combustion mechanisms, intermediate compounds and radical species are produced and consumed in the *reaction region*. The numerous interactions between these species, and the more stable chemical species (i.e. the hydrogen or methane fuel, oxygen, nitrogen, etc.), form a sequence of events I define as combustion. In the remainder of this section I explore the methane and hydrogen combustion datasets, using my method to identify scientifically interesting relationships between variables.

4.5.1 Methane Combustion

I discuss in detail the application and resulting analysis of my method to a methane combustion dataset[1]. The abbreviated, general chemical equation for methane combustion is shown in Figure 4.3. Here, oxygen (O_2) and methane (CH_4) react to produce water (H_2O), carbon dioxide

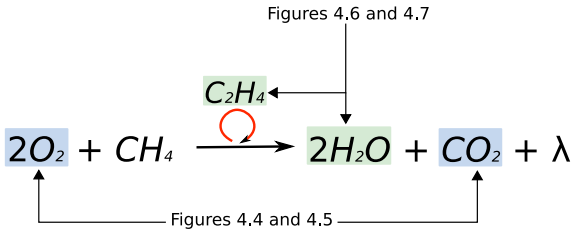
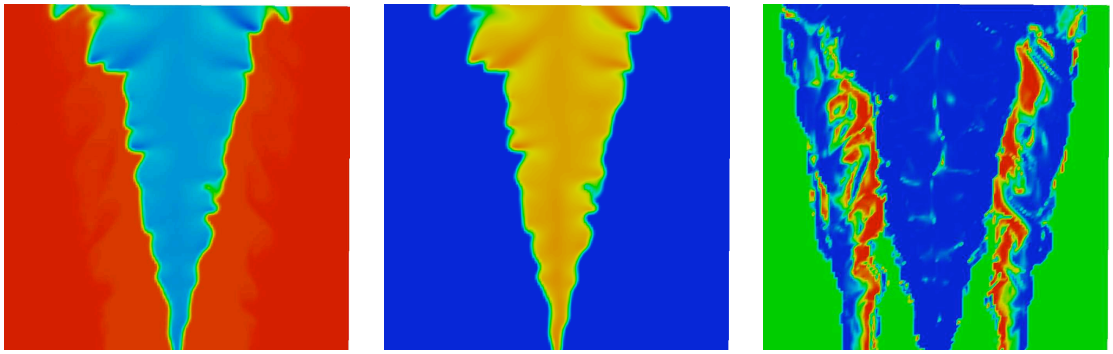


Figure 4.3: Combustion process for methane.

[1]The sample data used in this example is related to a simulation [15, 27] of a lean premixed turbulent methane flame, which incorporates 20 chemical species and 84 reactions.



(a) Oxygen (O_2) concentrations

(b) Carbon Dioxide (CO_2) concentrations

(c) Correlation space for O_2 and CO_2

Figure 4.4: Oxygen (O_2) and carbon dioxide (CO_2) concentrations from the methane combustion dataset are shown in (a) and (b), respectively. The derived correlation field for these two gasses is shown in (c). Areas of strong positive correlation (shown in red) in the mixing regions in (c), which do not appear in either (a) or (b), are a result of using *normalized* dot products to construct the correlation field. Normalization allows for the observation of interactions between these gasses, even at trace concentration levels.

(CO_2), and energy (λ). Ethylene (C_2H_4), one of the intermediate chemical species produced and consumed in the combustion process, is shown above the red loop.

My analysis focuses on isosurfaces of temperature (i.e. isotherms) in respect to two separate correlation fields: the correlation field derived from O_2 and CO_2, and the correlation field derived from H_2O and C_2H_4.

To begin, Figure 4.4 depicts the concentrations of O_2 and CO_2 ((a) and (b), respectively). In these images, red regions indicate areas of high concentrations and blue regions indicate areas of low concentrations. Figure 4.4(c) shows a 2-dimensional slice of the correlation field derived from O_2 and CO_2. For this image, red regions indicate strong positive correlation, blue regions indicate strong negative correlation, and green regions indicate regions where there is little to no correlation.

Note that areas of strong positive correlation (visible in the mixing regions) in Figure 4.4(c), which are not discernible in either Figure 4.4(a) or (b), are a result of using *normalized* dot products to construct the correlation field. Normalization allows for the observation of interactions between these gasses, even at trace concentration levels.

To interpret the correlation image in Figure 4.4(c), observe that in the blue regions (i.e. when moving to the center of the image from either just left or right of center), the concentrations of CO_2 and O_2 are simultaneously increasing and decreasing respectively; in red regions, O_2 concen-

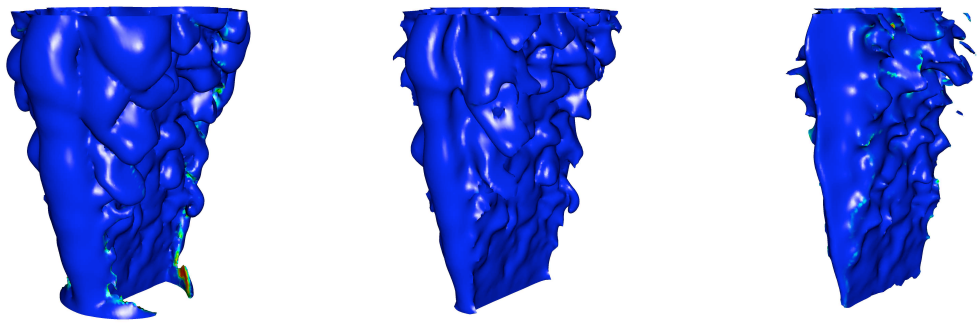
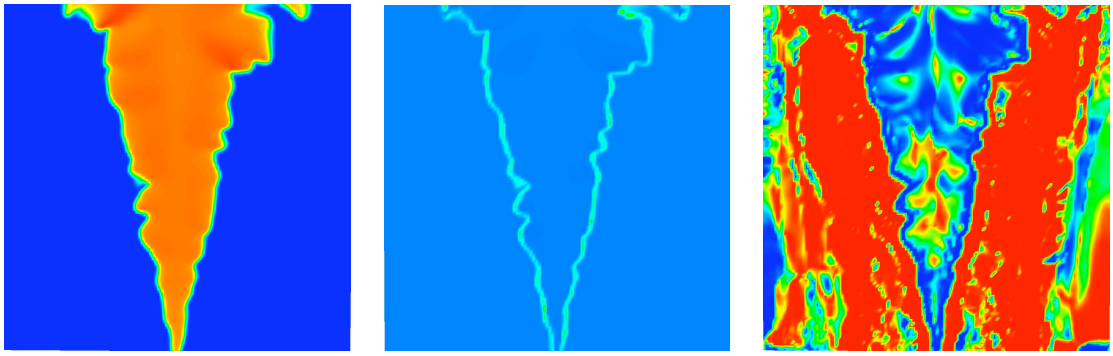


Figure 4.5: From the methane combustion dataset, temperature isosurface values of 300, 1000, and 1800 degrees Celsius are shown "colored" with the correlation field derived from oxygen (O_2) and carbon dioxide (CO_2). Note that the isotherms lie in regions of strong negative correlation, regardless of temperature value; this is visually evident because each surface is predominantly colored blue (negative correlation from Figure 4.2). Physically, this indicates that any correlation between O_2 and CO_2 (shown in Figure 4.4(c)) is independent of temperature.

trations increase in the same direction as CO_2. In the green regions, because they are uncorrelated, no discernible trend exists between O_2 and CO_2. Observing Figure 4.1(b), it is clear that the region where O_2 and CO_2 are positively correlated in Figure 4.4(c) corresponds to the mixture region. Figure 4.4(c) shows strong negative correlation outside the mixture region, especially on the mixture region's outer edge, and in the whole of the product region. This negative correlation in the product region is intuitive from the inverse relationship shared between these two chemical species as defined in Equation 8 (i.e. O_2 is a reactant and CO_2 is a product).

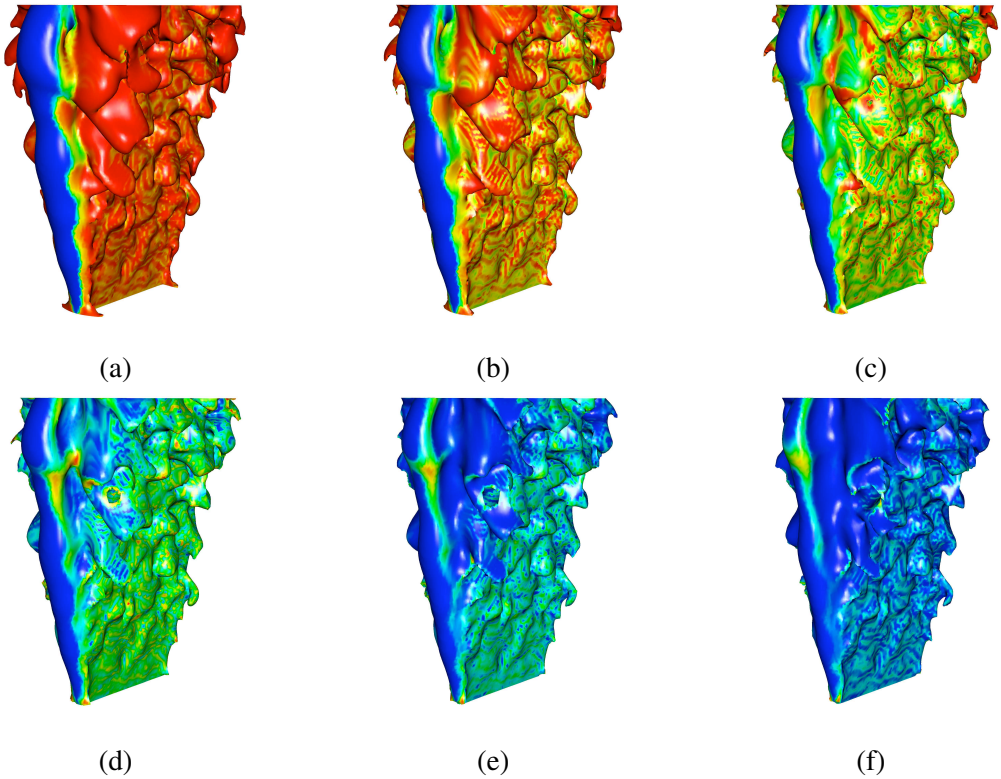
Coloring the isosurfaces of a single variable with the correlation field derived from two other variables is useful in identifying the important interactions occurring, or trends shared, between all three variables (as discussed in Section 4.4). Figure 4.5 depicts isosurfaces of increasing temperature that are colored with the correlation field shown in Figure 4.4(c). All isosurfaces in Figure 4.5 (shown at 300, 1000, and 1800 degrees Celsius) lie in regions of strong negative correlation, regardless of temperature value; this is visually evident because each surface is predominantly colored blue (negative correlation from Figure 4.2). This indicates that temperature has little influence on the relationship between O_2 and CO_2, or conversely, that any interactions between O_2 and CO_2 are independent of temperature (i.e. they are neither exothermic or endothermic which indicates that these two species are NOT reactive in this simulation).

As a second example in this dataset, I examine the relationship between the stable species



(a) Water (H_2O) concentrations (b) Ethylene (C_2H_4) concentrations (c) Correlation space for H_2O and C_2H_4

Figure 4.6: Water (H_2O) and ethylene (C_2H_4) concentrations from the methane combustion dataset are shown in (a) and (b), respectively. The derived correlation field for these two gasses is shown in (c). The switch from strong positive correlation to strong negative correlation in the reaction region corresponds to the area in which C_2H_4 is both produced and consumed, and H_2O is produced, in the process of combustion. The strong correlation (both positive and negative) in the center of the flame, as well as the atmospheric region, demonstrates the correlation field's ability to show fine-scale interactions (see also Figure 4.4(c)).



(a) (b) (c)

(d) (e) (f)

Figure 4.7: These images depict increasing ((a) through (f)) isosurface values of temperature (isotherms) colored by values of the correlation field derived from water (H_2O) and ethylene (C_2H_4) (see Figure 4.6(c)). As temperature values increase, the predominant correlation between H_2O and C_2H_4 along the isotherms shifts from strongly positive (red) in (a), to strongly negative (blue) in (f). This shift suggests that temperature is itself negatively correlated with the H_2O-C_2H_4 correlation.

H_2O, which exists in the product region, and the intermediate species C_2H_4, which exists primarily in the reaction region. As previously mentioned, interactions between intermediate species and more stable species are essential for combustion to occur. Figure 4.6 shows concentrations of H_2O and C_2H_4 ((a) and (b) respectively), and the corresponding correlation field generated between these two variables in (c). Interestingly, the switch from strong positive correlation to strong negative correlation in the reaction region corresponds to the area in which ethylene is both produced and consumed, and water is produced, in the process of combustion. Areas of strong correlation (both positive and negative) in the center of the flame, as well as in the atmospheric region, demonstrate the correlation field's ability to show fine-scale trends (see also Figure 4.4(c)).

As with my first example, it is possible to color isotherms with values in the H_2O-C_2H_4 correlation field as shown in Figure 4.7. Unlike Figure 4.5, which showed little correlation between temperature and the O_2-CO_2 correlation, temperature seems to be correlated (to some degree) with the relationship between H_2O and C_2H_4. Visually, this is supported by Figure 4.7. At low temperature, Figure 4.7(a), the isotherm is in the mixture region. Along the isotherm in the concentration fields, the gradients point inward, thus yielding strong positive correlation in the correlation field. This correlation means that the direction of greatest increasing H_2O concentration is very similar to the direction of greatest increasing C_2H_4 concentration.

As the temperature value increases, the isosurface moves steadily into the reaction region. Along these isotherms the correlation between H_2O and C_2H_4 begins to change. By the third isotherm (Figure 4.7(c)) there is little correlation between the chemical species (as per Figure 4.2, green indicates regions of near-zero correlation). Thus, over the isotherm, one can infer very little about changes in H_2O concentration from changes in C_2H_4 concentration (and vice versa).

It is in these regions of near-zero correlation where I make the following observation. As discussed in Section 4.4, those regions where the correlation field for two variables is uniformly minimal over a predominant region on the isosurface of a third variable indicate areas of important changes or interactivity between all three variables. Additionally, if large parts of an isosurface of one variable coherently follow a transformation in correlation between two other variables (positive to negative, or negative to positive) as depicted in Figure 4.7(c), then it is likely that the isosurface rendered at the point of near-zero correlation between the two other variables is important to the external event or internal process causing the entropy change.

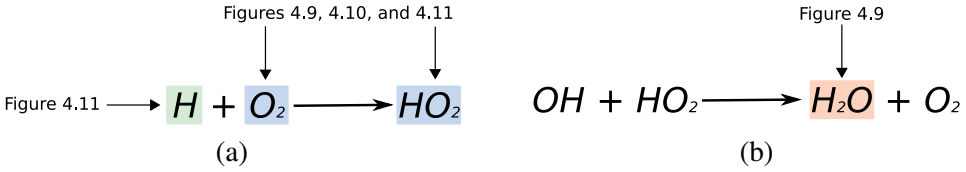


Figure 4.8: Chemical mechanisms used in hydrogen combustion.

To interpret this in the context of my dataset, I know that C_2H_4 is a temporary species formed and consumed in the process of combustion. In the flame front regions, I expect the concentration of such species to be relatively high. Conversely, I expect H_2O concentrations to be relatively low on the edge of the flame front. The correlation between C_2H_4 and H_2O will then be minimal, i.e. higher entropically, in these areas where the production and consumption of C_2H_4, and production of H_2O is greatest (i.e. the regions at, or near, the flame fronts).

Increasing the temperature further reveals regions of strong negative correlation, specifically in the product region. I know from basic knowledge about the regions (see Figures 4.6(a) and (b)), that concentrations of H_2O are low in the mixture and outer region (where the temperature is also low), thus the correlation field indicates that C_2H_4 concentrations should also be low. In the product region, where the temperature is hottest, I know that concentrations of H_2O are high so that from the correlation field I then know that concentrations of C_2H_4 should be very low. This pattern suggests that temperature itself is *negatively* correlated with the correlation between H_2O and C_2H_4.

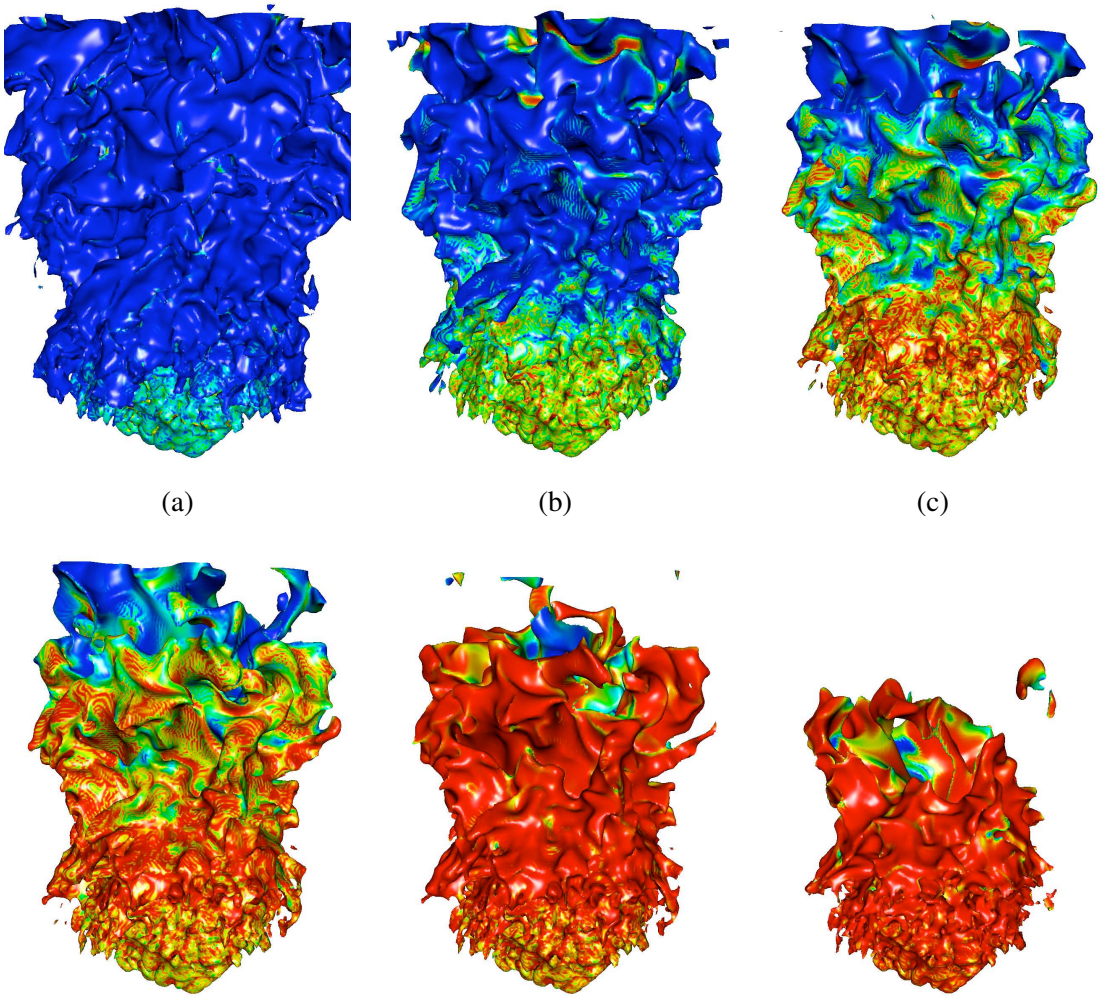
4.5.2 Hydrogen Combustion

In this section, I discuss the application and resulting analysis of my method to a hydrogen combustion dataset[2]. The general chemical equation for hydrogen combustion is: $O_2 + 2H_2 \Rightarrow 2H_2O + \lambda$. Rather than focusing on the abridged, general reaction, I instead analyze two intermediate reactions principal to the formation of water in combustion:

The chemical equation in Figure 4.8 (a) shows a hydrogen radical (H) interacting with elemental oxygen (O_2) to produce a perhydroxyl radical (HO_2). The chemical equation in Figure 4.8 (b) shows a hydroxyl radical (OH) reacting with HO_2 to produce water (H_2O) and O_2.

In the hydrogen combustion dataset, isosurfaces of increasing H_2O and H concentrations

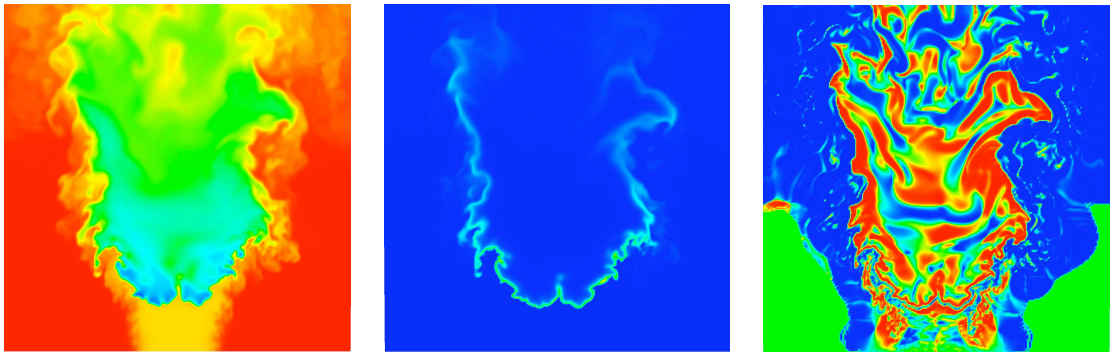
[2]The data for this example is extracted from an as yet unpublished simulation of a lean premixed turbulent hydrogen flame, incorporating 9 species and 27 reactions.



(a) (b) (c)

(d) (e) (f)

Figure 4.9: These images depict increasing ((a) through (f)) iso-concentrations of water (H_2O) colored by values from the correlation field of oxygen (O_2) and perhydroxyl radical (HO_2) (see Figure 4.10(c)). As water concentration increases, the predominant correlation along the isosurface shifts from strongly negative (blue) in (a), to strongly positive (red) in (f). This shift suggests that H_2O concentration is itself positively correlated with the O_2-HO_2 correlation. Local variations in this observed correlation (e.g. the bottom of the isosurfaces transitioning from negative correlation to positive correlation faster than the top of the isosurfaces) are likely due to the fact that in the hydrogen dataset, unlike the methane dataset, burning occurs unevenly along the isotherms. Such variations in combustion influences both the rates of reactions and the locations of reaction fronts. As such, transitions in correlation are expected to occur at different concentrations in the isosurfaces of H_2O (as this image depicts).



(a) Oxygen (O_2) concentrations

(b) Perhydroxyl (HO_2) concentrations

(c) Correlation field for O_2 and HO_2

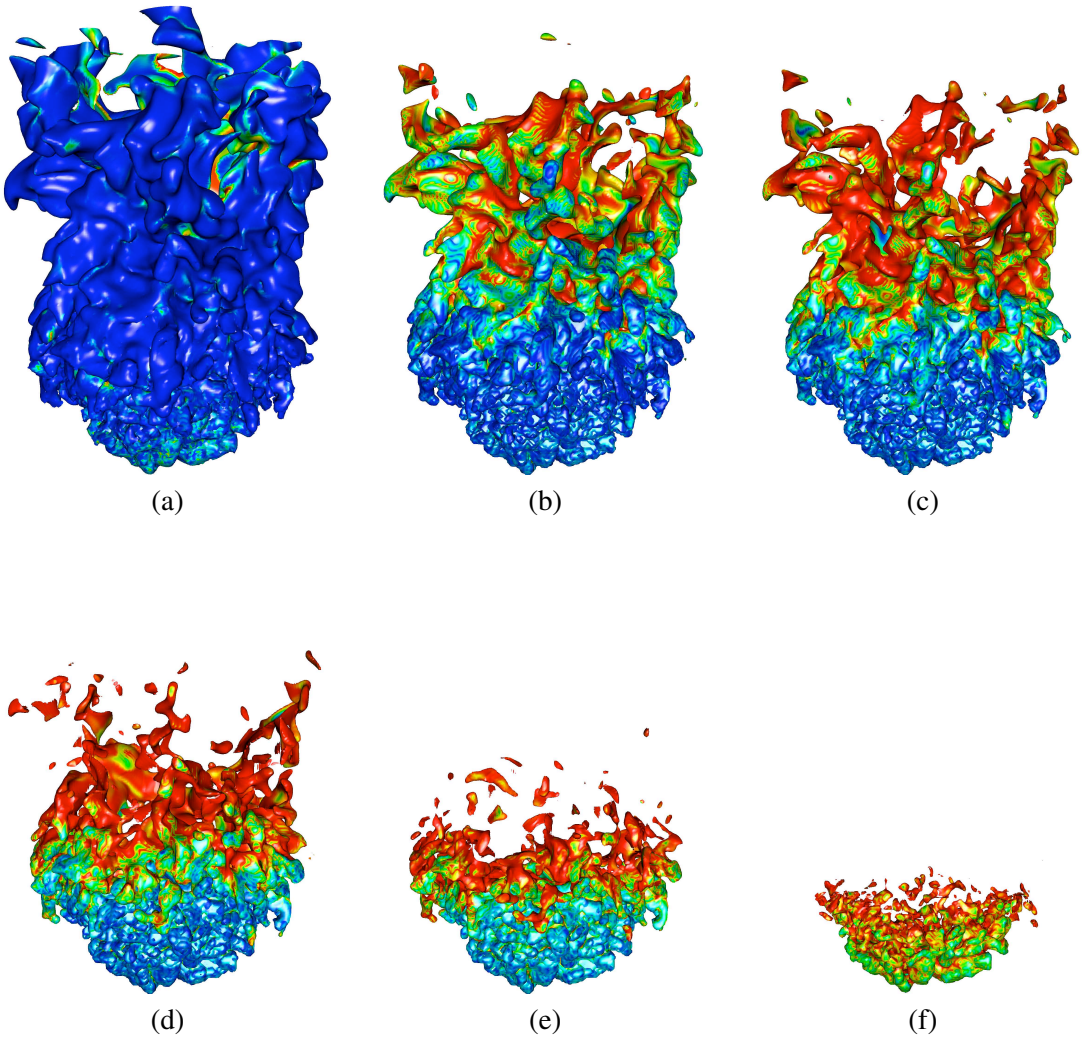
Figure 4.10: I show slices through an ultra-lean premixed hydrogen flame combustion dataset. Unlike the methane combustion dataset, in which burning occurs "evenly" along isotherms, combustion in the hydrogen dataset is very uneven and results in a characteristic "bubbling" shape. Oxygen (O_2) and perhydroxl radical (HO_2) concentrations are shown in (a) and (b), respectively. The derived correlation field for these two gasses, which highlights their turbulent interplay within the reaction region, is shown in (c).

are separately rendered through a correlation field constructed from O_2 and HO_2.

Figures 4.10(a) and 4.10(b) show the concentrations of O_2 and HO_2, respectively. Here, as with the methane example, red regions indicate the highest concentrations of a given species and blue indicates the regions of lowest concentrations. Figure 4.10(c) depicts a 2-dimensional slice of the correlation field generated between the gradients of these two chemical species. In this image, red regions indicate strong positive correlation, blue regions indicate strong negative correlation and green regions indicate regions where there is little to no correlation.

Figure 4.9 depicts isosurfaces of increasing concentrations of H_2O that have been colored by values from the correlation field derived from O_2 and HO_2 (see Figure 4.10(c)). These rendered isosurfaces exhibit striations (i.e. bands of negative, zero, and positive correlation) in the correlation field. With increasing concentrations of H, correlation is shown to increase within each striation. This behavior suggests that H concentration is positively correlated with the O_2-HO_2 correlation.

A possible explanation for the positive correlations, exhibited across striations of negative, zero, and positive correlation, is that in the hydrogen dataset, unlike the methane dataset, burning occurs unevenly along the isotherms. Such variations in combustion influence both the rates of reactions and the locations of reaction fronts. As such, transitions in correlation are expected to occur at different concentrations in the isosurfaces of H_2O (as Figure 4.9 seems to depict).



(a) (b) (c)

(d) (e) (f)

Figure 4.11: These images depict increasing ((a) through (f)) iso-concentrations of hydrogen radicals (H) colored by values from the correlation field of oxygen (O_2) and perhydroxyl radical (HO_2) (see Figure 4.10(c)). Each isosurface exhibits striations in the correlation field (i.e. bands of negative, zero, and positive correlation), and as H concentration increases, correlation increases within each striation (i.e. negative correlation tends to become positive). This behavior suggests that H concentration is itself positively correlated with the O_2-HO_2 correlation. Image (f) indicates the simultaneous existence of near-zero correlation (i.e. high entropy) *and* high H concentrations; in Section 4.5.2 I hypothesize that this combination is a driving force in the reaction shown in Equation 9.

Observe the areas of highest entropy (green) on isosurfaces of medium H_2O concentration in Figures 4.9(b), (c), and (d). These areas indicate the regions in which the reaction from Equation 10 is most likely to occur. Note that for higher-concentration H_2O isosurfaces (Figures 4.9(e) and (f)), entropy has decreased, corresponding to high positive correlation between O_2 and HO_2.

These observations suggest that areas of high H_2O production are not necessarily areas of high H_2O concentration; this indicates that H_2O is being moved by some means away from the primary reaction area. Alternatively, this may suggest that there are additional H_2O-producing reactions (beside that shown in Equation 10) driving the production of water. Having made this observation, it would be instructive to visually explore additional correlation fields, constructed from species involved in *other* H_2O-producing reactions.

In Figure 4.11, isosurfaces of increasing H concentration are shown. Isosurfaces of H, similar to H_2O, highlight striations in the correlation field that increase in correlation within each striation as the concentration of H increases. These changes in correlation (from negative to positive), that coincide with the increasing concentrations of H, suggest a positive correlation between H concentrations and the O_2-HO_2 correlation.

Additionally, there is one significant difference between H and H_2O in reference to O_2-HO_2 correlation. Figure 4.9 suggests that the reaction in Equation 10 is unable to drive H_2O concentrations to their highest values. In Figure 4.11(f), however, the highest concentrations of H correspond to regions of high entropy (i.e. near-zero correlation). This is significant because the simultaneous existence of high entropy with high concentrations of H suggests that (f) highlights a region in which Equation 9 consumes significant amounts of H and produces a significant amount of HO_2.

4.6 Summary

In this chapter I have presented a method for exploring and identifying variable interactions within the solution space of a query. One of the limitations of this method, which I discuss in more detail in Section 7.3.1, is that this strategy can only identify meaningful trends between, at most, three variables; i.e. one variable and a correlation field constructed between two other variables. In the next part of this dissertation, Chapter 5, I present a domain-agnostic, statistics-based

framework that can identify important trends between any number of variables constrained by the query. I show how the information obtained from this framework can be used to help refine the constraints expressed over variables in a scientist's query.

Chapter 5

An Application of Multivariate Statistical Analysis for Query-Driven Visualization

5.1 Introduction

Scientific visualization accelerates the discovery process in contemporary science by transforming abstract data into a visual representation that readily conveys comprehensible, meaningful information to the scientist. In the scientific discovery process, Butler et al. [22] identify three functional modalities for visualization use: visual-discovery, visualization-based analysis, and presentation visualization.

Discovery visualization strategies, which are employed early in the investigatory process, are designed to uncover new trends or features in data where the scientist is unsure—due to the complexity of the data or the undefined nature of the event being studied—what trends or anomalies to look for. Such strategies typically demand a great deal of I/O bandwidth and computation to support high levels of user interactivity. Analytical visualization strategies serve to help the scientist confirm the presence or absence of *known* features and trends in data. As visual analysis applications do not support the exploratory or interactive functionality of visual-discovery strategies, they are typically implemented as non-interactive, post-processing functions that possess comparatively low

I/O demands. Visualization's final role in the discovery process, i.e. presentation visualization, is to aid the scientist in conveying the characteristic trends observed in the data to others.

One of the most significant challenges faced by contemporary science is the explosive growth of ever-larger, increasingly complex information. Managing this mass of increasing information and deriving knowledge or insight from this data are the two most significant bottlenecks impeding discovery visualization and the scientific discovery process [71].

Query-based strategies, like Query-Driven Visualization (QDV), present one way to help address these challenges. QDV is a discovery visualization strategy that couples scalable database technologies with visualization techniques to efficiently manage large-scale data, perform rapid data analysis, and support unconstrained, interactive exploration of scientific data [96, 110–112]. The term "Query-Driven Visualization" refers to the strategy of restricting computation and cognitive workloads exclusively to records defined to be "interesting" by the scientist. This strategy is realized through the evaluation of user-specified, multidimensional Boolean range queries—e.g. $(temperature < 1200)$ AND $(pressure > 2.4)$—that rapidly filter away large portions of non-pertinent data, and allow smaller, more interesting subsets of data to be efficiently analyzed and visualized.

Query-based visualization research successfully addresses many important areas such as scalability and performance [38, 96], visualizing multitemporal data [37, 40], applying QDV strategies to adaptively meshed domains [40], addressing uncertainty in multivariate queries [37], and extending query-based strategies to address the challenges of visualizing function field data [4]. Researchers have also successfully combined domain-specific knowledge with QDV, e.g., to study network traffic [18] or combustion flame fronts [39].

Comparatively little work, however, has been performed on the explicit study of QDV solutions, i.e. the set of records selected by a scientist's multidimensional Boolean range query. This underdeveloped area of research highlights a central problem in QDV: while solutions to range queries can help scientists identify interesting visual features, trends, or anomalies within their data, existing query-based research offers little assistance in helping scientists to better understand these events. The challenge is to extend the strengths of QDV with new analysis methods that can help QDV users better understand the solutions to their queries. Such strategies are essential for scientists to progress from stages of discovery visualization to stages of visual analysis and presentation.

This work extends the utility of QDV with a statistical framework that enables scientists to attain greater levels of insight into the features and anomalies identified by QDV applications. With this framework, QDV users can identify:

- the individual significance (e.g. central statistical tendencies and important trends) of each variable to the query's solution in the comparative context of all other variables constrained by the query;
- the salient *joint* trends (i.e. trends based on the behavior and interaction of *all* variables combined) that are characteristically important to understanding the query's solution; and,
- how to adjust individual variable constraints in a query to focus on, or exclude, these newly identified trends in subsequent searches.

The new insight provided by this statistical framework will help to accelerate scientists from stages of visual-discovery into stages of visual analysis and presentation.

In this framework, the "statistical structure" of a query is composed of two statistical measures: a global measure that takes into account all variables constrained by the query, and a segmentation, which is based on the localized statistical contribution of all variables to the query's solution.

The global measure is comprised of the estimated joint distribution of the query's solution space. Exploring this joint distribution allows QDV users to interactively explore their query's solution and visually identify the regions where the *combined* behavior of constrained variables is most important or interesting to their search. To provide further insight into the query's joint distribution, I introduce a new segmentation strategy that extends the distribution estimation analysis by visually conveying the *individual* importance of each variable to these regions of high statistical significance. The global and localized measures, when integrated together, facilitate a means for refining variable constraints expressed in the QDV user's query. This framework addresses a critical need in query-based research by providing a domain-agnostic approach that solidifies a path for discovery visualization users to take in order to begin understanding the events and anomalies they have identified in their data.

The main contributions of this work include the following:

- I introduce a statistics-based framework that extends the utility of QDV strategies by helping users to better understand the solutions to their queries. The core of this framework is based on a strategy that integrates non-parametric distribution estimation techniques with a new segmentation strategy to visually identify statistically important variable trends—both individual variable trends and *joint* trends for groups of variables—within the solution space of a query.

- I show how users can use the information obtained from the query's joint distribution and segmentation to refine the constraints over individual variables in a multivariate query.

I demonstrate these methods across data from different application domains. Furthermore, I perform all statistical data processing on the graphics processing unit (GPU) to facilitate quick response times for the QDV user.

In the next section, I discuss work that is germane to my efforts. In Section 5.3 I present the statistical processing framework and approach for creating surfaces and segmentations of query solutions. This surfacing and segmentation enables meaningful query visualization and analysis, as I show in Section 5.4. I conclude by addressing important implementation issues, with a discussion of GPU implementation and performance in Section 5.5.

5.2 Previous Work

5.2.1 Distribution Estimation in Image Processing and Computer Vision

In the image processing community, distribution functions and methods for estimating distributions, e.g., kernel density estimates (KDE) [85,88], are used primarily for image classification—i.e. a Boolean segmentation of the image into regions of interest and regions of non-interest. For example, Zhang and Yang [118] utilize KDE and statistical analysis to detect and isolate moving objects of interest, e.g. pedestrians, cars, etc., from streaming images. Liu et al. [65] also demonstrate a KDE-based probabilistic framework for image classification. They use probability distribution functions based on a hybrid KDE and Gaussian Mixture Model (GMM) to isolate moving objects from movie frames while simultaneously removing artifacts like shadows and obstructing foreground.

Mean shift clustering [24, 36, 116], which is based on the mean shift procedure presented by Fukunaga and Hostetler [34], is a common, non-parametric segmentation technique used in computer vision research to facilitate feature space analysis. In mean shift clustering, the feature space of the image is modeled by its estimated joint distribution (obtained through KDE methods). Comaniciu and Meer [24] show that the dense regions in the feature space of the image directly correspond to local maxima in the image's estimated joint distribution. Segmentation of the image is realized by assigning pixels in feature space to the modality nearest the pixel in the image's estimated joint distribution. There is thus one unique segment in the image for every unique local maxima in the image's distribution estimate.

A fundamental difference between the segmentation generated by mean shift clustering and my segmentation strategy is that mean shift clustering is driven by the gradient of the estimated joint distribution, whereas my segmentation is based on the localized statistical significance each variable plays in *constructing* the joint distribution. Hence, mean shift clustering is a post-process performed *after* the KDE calculation, and my segmentation is obtained *during* the process of calculating the KDE. In Section 5.3.3 I discuss this difference more thoroughly when I introduce my segmentation procedure.

5.2.2 Distributions in Visualization

In the visualization community distribution functions, e.g. histograms [54], support and facilitate a wide variety of tasks. For example, the volume reconstruction equation used in splatting utilizes a distribution function to calculate and spread the color contribution a given voxel makes to a pixel region in screen space [105]. Westover [106] shows the model used to construct the splatting distribution function (e.g. Gaussian, bilinear) dramatically influences the quality of the rendered image during the splatting process. Crawfis and Max [26] present a cubic spline function for splatting that supports both accurate rendering from all viewing directions and generates images superior to those rendered with a Gaussian distribution kernel. Mueller et al. [72] use distribution kernels of varying size, where the kernel size is based on a given voxel's distance to the viewing plane, to effectively ameliorate aliasing artifacts in splatting. This type of approach is particularly effective when the volume resolution is higher than the image resolution.

Histograms are also used extensively in volume rendering. Ledergerber et al. [60] utilize a moving least squares method to reconstruct the underlying distribution of a series of data points along a single ray. They use the reconstructed distribution to volume render high-quality images with accurate shading. Kniss et al. [59] combine the underlying distribution functions of scalar data values with data attributes, such as gradient magnitude, to derive 2D and 3D transfer functions. These transfer functions provide a powerful and intuitive way to rapidly isolate important visual features (e.g. surfaces) in multivariate data that are not able to be isolated with simple 1D transfer functions. Finally, Lundstrom et al. [67] combine user domain knowledge, in the form of local histogram criteria, into a certainty-based classification strategy to create transfer functions for direct volume rendering. They apply their strategy on magnetic resonance data and show that their constructed transfer functions clearly detect and separate important tissues of interest, e.g. liver, spleen, kidney, during volume rendering.

Recently, the direct relationship between isosurface complexity, histogram distributions, and geometric statistics (i.e., the area, volume, and gradient magnitude corresponding to the surface of a given isovalue) has been established by Carr et al. [23] and Scheidegger et al. [90]. Scheidegger et al. show that isosurface statistics and histograms converge to the same results. They extend this finding by showing how their techniques can be seen as a way to calculate expectations of random variables on isosurfaces. Bajaj et al. [9] compute geometric statistics of isolines and isosurfaces and display them in an interface to assist users performing discovery visualization tasks. In this work, contour trees are used to provide global structure to the observed statistical measures in order to provide visual cues to the user about interesting and important isovalues in scalar data. This work presents an excellent tool to guide scientists tasked with visual discovery. Unfortunately, as the authors' strategy focuses on scalar field data, the work does not immediately lend itself to the challenges of visualizing multivariate data; for example, the important trends observed between multiple variables in high dimensional data.

Linsen et al. [63] focus on feature space analysis for Smoothed Particle Hydrodynamic (SPH) simulations with a d-dimensional binning strategy to estimate distributions for SPH-based scattered data. The resulting distribution gives rise to a hierarchical distribution-based clustering of feature space. However, in comparison to my approach, their method cannot be used to examine the influence individual variables have in the distribution or the distribution generated by the specific

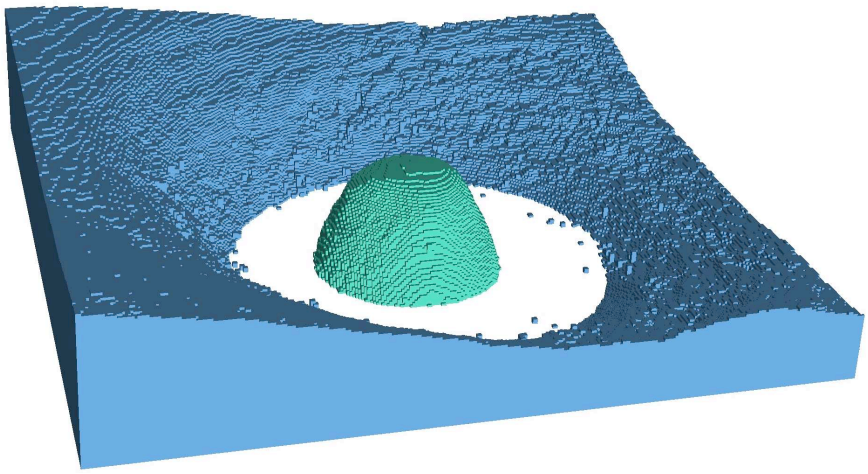


Figure 5.1: This figure depicts a typical cell-based rendering used to visualize query solutions in query-driven visualization. In this image, regions of low pressure (green, at center) and high pressure (blue) are visualized for a query that selects regions where pressure is less than -1500 Pascal, or above 250 Pascal.

subset contained in a query's solution set.

Query-driven visualization offers a rich setting in which to employ distribution functions for analysis. In the following section I apply a statistics-based framework to generate a joint distribution function for a query's solution. I use this distribution to construct accurate surfaces that indicate where important regions of interest lie within the solution space of the query. I also generate a segmentation, based on the contributing distribution of all variables, in order to isolate and visualize important statistical features of interest from the query. This segmentation can be used to help refine variable constraints in the query in order to further investigate regions of interest.

5.3 Method

Query-driven visualization is based upon the evaluation and direct visualization of queries over large scientific data [96]. Queries typically take the form of Boolean range constraints upon individual variables of multivariate data. The "solution" to a query are the regions of a dataset for which the variables satisfy the range constraints.

Consider a function $f : R^3 \rightarrow R^d$, and a query comprised of lower and upper limits, a and

b respectively, upon the range of f. The solution Q to the query may then be defined as:

$$Q := \{p \in R^3 | a \leq x \leq b\},$$

where $a, b \in R^d$, and x is the vector of variable values associated with point p, e.g. $f(p) = x = (x_0, \ldots, x_d)$. The solution set Q then corresponds to all the points in the spatial domain for which $a_i \leq x_i \leq b_i$ for $i \in (0, \ldots, d)$.

Classically, query solution sets have been visualized by using a straightforward depiction of their data constituents (i.e. points or cells). This visualization is carried out by rendering each cell that passes the query as a hexahedral cube [40, 96], or a solid sphere [37]. An example of this type of rendering is shown in Figure 5.1. In this image, regions of low pressure and high pressure are observed in a hurricane dataset. These regions correspond to areas where pressure is less than -1500 Pascal (green at center), or above 250 Pascal (blue).

Isosurfaces too can be used to visualize query solution sets by defining $h : R^d \rightarrow [0, 1]$:

$$(h \circ f)(p) = \begin{cases} 0 & \text{if } p \notin Q \\ 1 & \text{if } p \in Q \end{cases}$$

Visualizing the surface $(h \circ f)^{-1}(0.5)$ can approximate the boundary containing Q with a single surface and more smoothly than cell or sphere-based renderings.

In this approach, however, by estimating the joint distribution of the $p \in Q$, I can provide both a single, accurate surface that depicts the query's boundary, and also provide information to the QDV user that enables them to better understand their query's solution, and refine the constraints of their query.

The foundation of this method is the computation of the underlying joint variable distribution for multivariate data within a query solution (Section 5.3.1). Using this joint distribution estimate, I describe the construction of surfaces from the distribution fields (Section 5.3.2). Finally, I define a segmentation of the query solution based on the localized, statistical significance each variable plays in *constructing* the joint distribution (Section 5.3.3).

5.3.1 Distribution Estimation for Queries

I begin query analysis by constructing a distribution estimation for the multivariate solution space of a query. Kernel Density Estimation (KDE) can be applied to develop a statistical model of the underlying functional behaviour of multiple samples from one or more variables. Consider a set of N observed data samples $x^0, x^1, \ldots, x^N$ from a function $f : R^3 \rightarrow R$. The estimation $\hat{f}$ for the underlying distribution of f is:

$$\hat{f}(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x^i}{h}\right), \tag{5.1}$$

where h is the kernel bandwidth parameter for smoothing, and K is a Gaussian kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp^{-\frac{x^2}{2}}. \tag{5.2}$$

To determine the kernel bandwidth parameter, I employ an *adaptive estimate spread* method [92]. This method has been shown to work well for unimodal distributions, while not over-smoothing features in multimodal distributions:

$$h := 0.9 \left(min\left(\sigma^2, \frac{R}{1.34}\right)\right) N^{-\frac{1}{5}}, \tag{5.3}$$

where σ^2 and R are the standard deviation and inner quartile range for the data samples, respectively.

In this work I apply KDE over the solution set of a query. Thus the data samples are the multivariate values $x \in R^d$ corresponding to the points $p \in Q$. Correspondingly, I must construct the joint distribution estimate over multivariate data samples, where the multivariate KDE is defined:

$$\hat{f}(x) = \frac{1}{N\left\{\prod_{j=1}^{d} h_j\right\}} \sum_{i=1}^{N} \prod_{j=1}^{d} K\left(\frac{x_j - x_j^i}{h_j}\right). \tag{5.4}$$

Here I use unique, per-variable kernel bandwidth parameters h_j computed using equation 5.3 to evaluate this multivariate Gaussian kernel.

5.3.2 Visualizing Queries Using Their Distribution

Previous QDV surfaces (Figure 5.1) have presented a "blocky", binary separation of space due to a lack of interpolation away from points returned by the query engine [96]. In the previous

section I defined the construction of distribution estimates for a multivariate query's solution space. Now, I utilize these estimates to define surfaces that bound query regions.

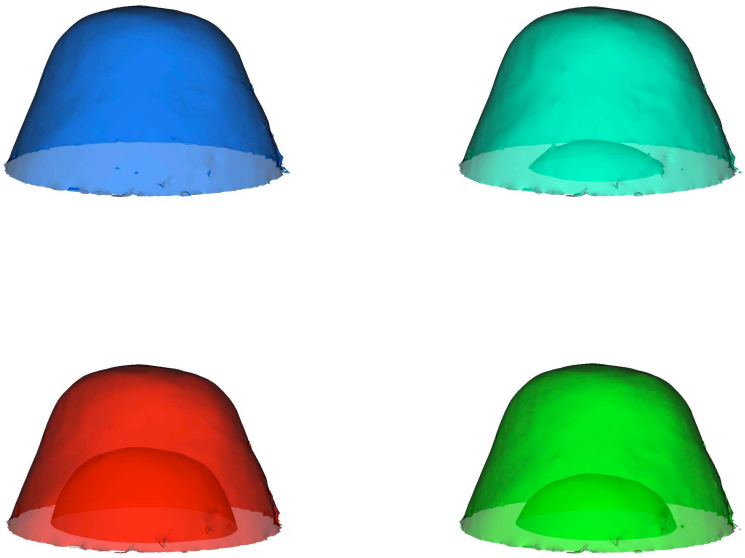
From a joint distribution estimate, a new scalar field $g : R^3 \rightarrow R$ is formed, where g maps all elements $p \in Q$ to their distribution values computed in equation 5.4. Elements outside the solution set, i.e. $p \notin Q$, are set to zero because they do not contribute to the query's underlying distribution:

$$g(p) = \begin{cases} \hat{f}(f(p)) & \text{if } p \in Q, \\ 0 & \text{otherwise.} \end{cases}$$

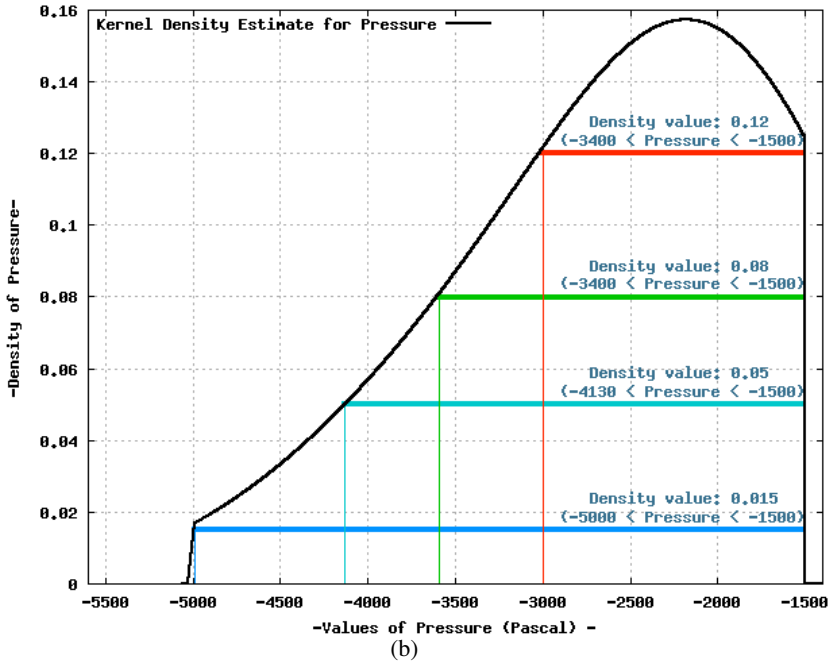
I use the clamped distribution estimate field g to construct surfaces that contain the solution space of the query. To generate a surface that bounds the query solution, observe that there will be some element $p^{min} \in Q$ with a minimum, non-zero distribution value. The solution to the query may then be visualized by the isosurface $g(p^{min})^{-1}$. I refer to such a surface as the "minimum distribution surface," and denote it simply as S_{min}. Because I map multivariate data samples to a scalar kernel density estimates (KDE), I can visualize query surfaces with common isosurfacing algorithms such as Marching Cubes [66] or raycasting.

Given $g : R^3 \rightarrow R$, it is possible to visualize surfaces corresponding to higher distribution values than S_{min}, with the goal of query analysis and refinement. Surfaces formed from increasingly higher distribution values will contain the regions for which data samples are more representative of the total data selected by the user's query. By examining these surfaces, the user is able to refine their variable constraints intuitively in a visual manner, and without losing the information critical to their query.

I illustrate this refinement procedure in Figure 5.2. Fig 5.2(b) shows the estimated distribution constructed from elements $p \in Q$ by a query selecting regions of low pressure in a hurricane dataset: pressure ≤ -1500 Pascal. Exploring this distribution with isosurfaces corresponding to increasing distribution values can help the user to locate new visual features and refine the constraints of the original query. I illustrate this constraint refinement in Figure 5.2(a); ovserve the surface corresponding to the original query's solution for low pressure in the bottom image (blue). This surface corresponds to the minimum distribution surface S_{min}. Using transparency, I show the effect



(a)



(b)

Figure 5.2: This figure visualizes distribution data calculated from a query that selects regions of low pressure in a hurricane dataset; pressure $\leq$ -1500 Pascal. Figure 5.2(a) shows specific isosurfaces corresponding to distribution values in this data: (clockwise from upper left) 0.015, 0.05, 0.08, and 0.12. Note the region this query captures can also be seen in Figure 5.1 as the green "dome" feature. The histogram in Figure 5.2(b) relates these increasing distribution values to an increasingly refined range of values contained in the query.

of examining surfaces for distribution values greater than S_{min}; moving up from the bottom image in Figure 5.2(a), elements with distribution values greater than 0.05 (blue-green), 0.08 (green), and 0.12 (red). Note that these isosurfaces also correspond to selecting an increasingly smaller subset of points $p \in Q$ with high distribution values. From Figure 5.2(b) I show that these subsets also correspond to an increasingly tighter range of values for pressure. With this type of exploration, the user can visually explore the solution (i.e. distribution) of their query to obtain information regarding the distribution behavior of its variables.

5.3.3 Multivariate Query Segmentation

When visualizing an estimated joint distribution constructed from equation 5.4, localized regions containing high distribution values can be the result of a single variable's contribution, or the cumulative contribution of several variables. To generate deeper insight into the query solution and help the user better understand regions of local maxima and minima in the joint distribution, I employ a strategy of feature analysis through segmentation.

There are many multi-labeled data segmentation algorithms [11, 51, 55, 75], but the most effective and common segmentation employed for the analysis of KDE is non-parametric mean shift clustering [24]. While mean shift clustering can classify and reveal distinct and major modalities in a distribution, it can't generate insight into the important variable trends occurring within these regions. For example, given a specific local maxima, or a group of maxima, in a query's joint distribution, a scientist may be interested in knowing:

- Are all variables constrained by the query well represented in these distribution features, or only certain ones? If certain variables are predominant, which ones and how predominant?
- Conversely, if a variable's distribution is *not* strongly contributing to specific modalities in the estimated joint distribution, where *is* this variable's distribution most predominant in contributing to the query's KDE?

Mean shift clustering can't generate enough insight to answer these questions, it can only identify the regions where the *joint* behavior of variables is significant. Hence, for Query-Driven Visualization (QDV) applications, mean shift clustering can't help the scientist progress from stages of visual discovery into stages of visual analysis and presentation.

To help answer these questions and generate deeper insight into the query's joint distribution, I present a new segmentation strategy based on each variable's individual contribution to the query's KDE. Let $\hat{f}_j$ denote the estimated univariate distribution of the j^{th} variable. I then extract the portion of the query solution Q_j associated with variable j as:

$$Q_j := \left\{ p \in Q |\ \hat{f}_j(x) > \hat{f}_k(x)\ \ \forall k \neq j \right\}.$$

Taken together, the subsets $Q_0, \ldots, Q_d$ form a partition of the query solution set Q. Note that in computing the joint distribution estimate in equation 5.4, the univariate distributions can be obtained by accumulating the individual Gaussian inner product terms for each j—thus the segmentation is obtained efficiently, with minimal additional overhead for computation and storage.

Interpreting Segmented Regions

From a high level the segmented regions visually convey—in the comparative context of all other variables constrained by the query—the individual significance of each variable to the query's solution. Visualizing segments concurrently by using isosurfaces (see Figure 5.4 and Figure 5.7 in Section 5.4) or direct volume rendering shows *where* the distribution of each variable is most important in defining the visual feature, trend, or anomaly the scientist has discovered.

Segmented regions, when visualized concurrently with local maxima regions in the query's KDE (see Fig 5.9), indicate which variables predominantly contribute to statistically important features in the query's joint distribution. Contrariwise, if a variable distribution is *not* strongly contributing to specific modalities in the estimated joint distribution, segmented regions can also indicate where a variable's distribution *is* most predominant in contributing to the query's KDE. I illustrate this strategy in Section 5.4.2 with a methane combustion dataset to show that regions corresponding to high distribution values in the query are predominantly influenced by *temperature* and CO_2 behavioral trends, and *not* trends due to *pressure*.

From a low level, the corresponding range of values for the $p \in Q_j$ contain a subset of values for the variable j that are important and significant for the user. To attain greater insight from the segmented regions, it is therefore important to consider the univariate distribution (i.e. histogram) of each variable j as found throughout the query's solution space Q, versus the vari-

able's segmented region Q_j. In this analysis I employ the univariate distribution estimates $\hat{f}_j$ for each segmented region *as it is defined exclusively to* Q_j and visualize the corresponding minimum distribution surfaces to represent each segmented region. As I show in Section 5.4.1, it is often the case that the distribution obtained for Q_j isolates distribution modalities from Q. This observation can then be used to perform multivariate query refinement. More specifically, it is possible to refine constraints over variable j to focus upon or exclude a modality isolated in Q_j. I illustrate this refinement strategy in Section 5.4.1 on a hurricane dataset. In this example, I refine an initial query by using a modality isolated in *temperature*'s segmented region.

I now apply this strategy—using KDE, segmentation, and query refinement—to two separate datasets to demonstrate its utility in generating greater insight for query-based discovery visualization strategies (Section 5.4).

5.4 Visualization Applications and Analysis

I apply my new method to two datasets and demonstrate the ability to generate surfaces that bind the query's solution and perform distribution-based segmentation. In the first example, this segmentation is utilized to refine the constraints expressed in the query.

5.4.1 Hurricane Dataset

This dataset was generated by a simulation modeling a hurricane over a 48 hour period. This dataset consists of 13 variables over a grid size of 300 x 300 x 90. In this experiment, I evaluate a query that selects all cells where records contain both low pressure, low wind velocity and fall in a broad range of temperature: pressure $\leq$ -350 Pascal AND velocity $\leq$ 10 mph AND -70 $\leq$ temperature $\leq$ 20 Celsius. The constraining characteristics of this query roughly approximate the features that classify the hurricane's eye in this dataset. In my analysis, I will analyze the variable-based segmentation of this region, and demonstrate my approach for multivariate query refinement.

I apply my method to the set of points that have been selected by the query after intersecting the regions of low pressure, low velocity, and broad temperature. For illustrative purposes, I show in Figure 5.3 slices through the hurricane's velocity (left) and pressure (right) scalar fields. Temperature is not depicted as the query selects all points based on values for temperature.

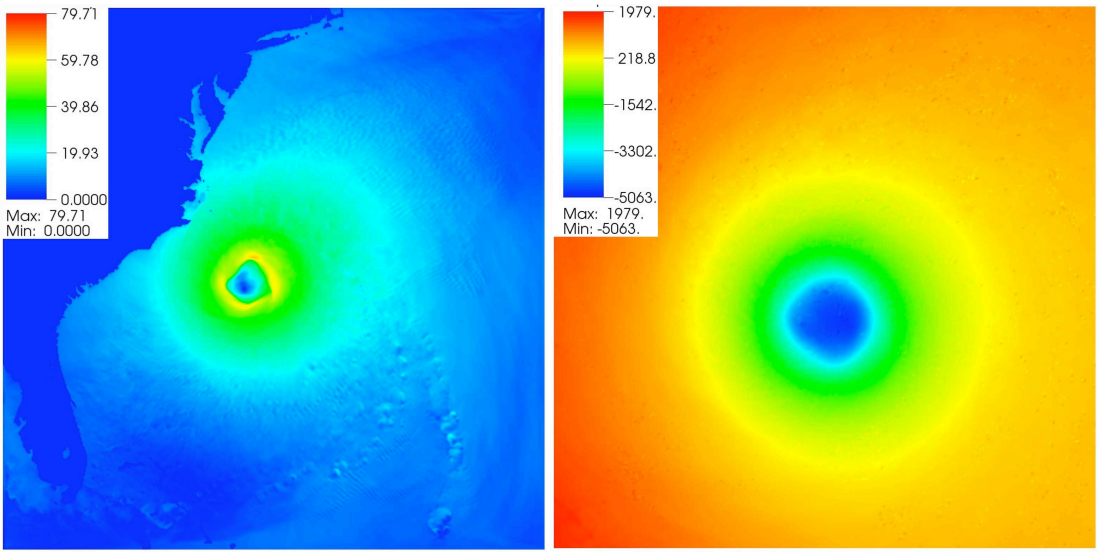
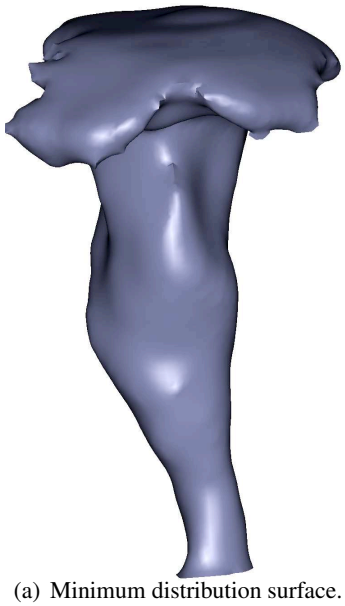


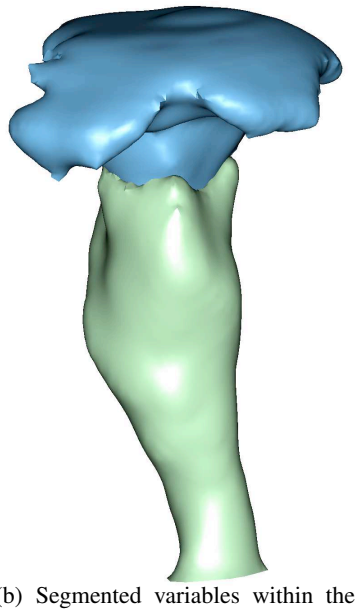
Figure 5.3: Slices through the velocity (left) and pressure (right) scalar fields of the hurricane dataset. In Section 5.4.1 I utilize these variables in a query that selects regions of low velocity and low pressure.

In Figure 5.4(a), I show the query solution set Q visualized by the minimum distribution surface S_{min}. Here I render surfaces using a traditional Marching Cubes implementation over the raw data of the scalar joint distribution field. The surface itself roughly resembles the center of the hurricane event. Note that the query is rather selective and that the mesh resolution over this feature is relatively low.

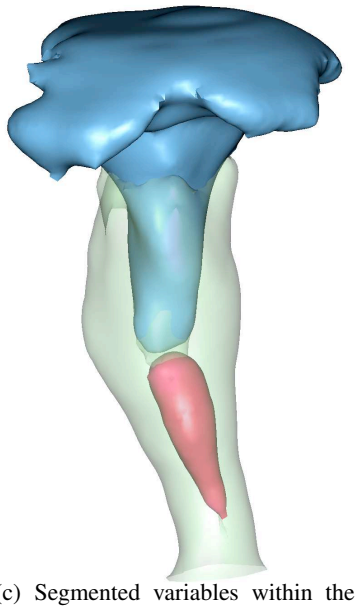
I next visualize the segmentation obtained when constructing the joint distribution for this query. In Figures 5.4(b) and 5.4(c), I show the results of the segmentation performed on the query's solution set. In these images there are three well defined visual regions of interest. The blue region in Figure 5.4(b) corresponds to the areas where pressure's univariate distribution contributes the most to the query's joint distribution. Correspondingly, the green regions indicate areas where velocity plays the most significant influence in raising the values of the query's joint distribution. In comparison to these larger surfaces, Figure 5.4(c) depicts a smaller, red surface at the center of the query's solution set. This region corresponds to the areas where temperature plays the most significant role in contributing to query's joint distribution.



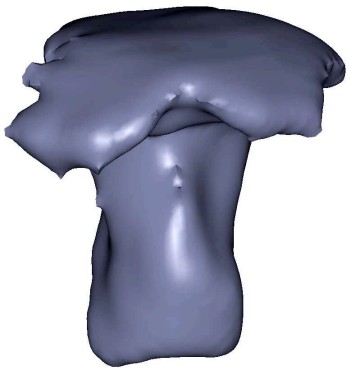
(a) Minimum distribution surface.



(b) Segmented variables within the query solution space; displayed are pressure (blue) and velocity (green).



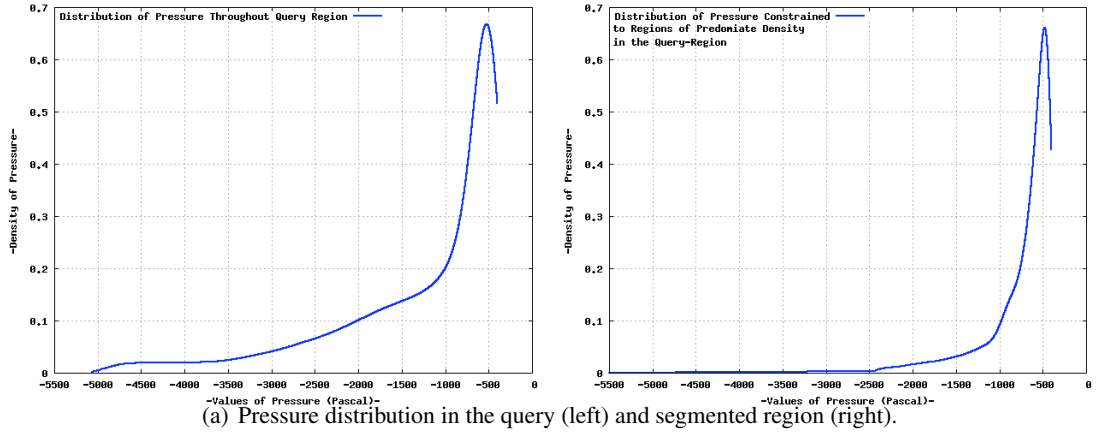
(c) Segmented variables within the query solution space. Regions where velocity contributes most to the joint density are shown in green, pressure and temperature are shown in blue and red respectively.



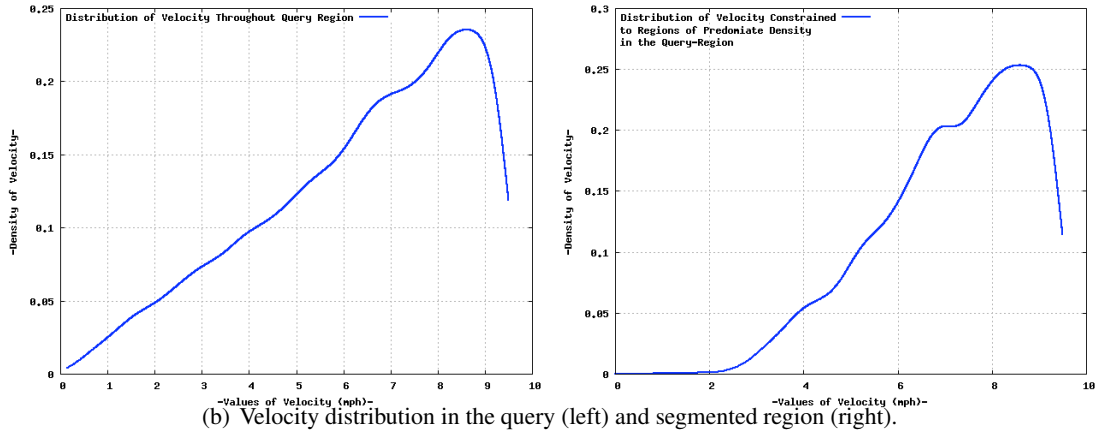
(d) Refined query solution surface visualizing the lower region of Figure 5.4(a).

(e) Refined query solution surface visualizing the upper region of Figure 5.4(a).

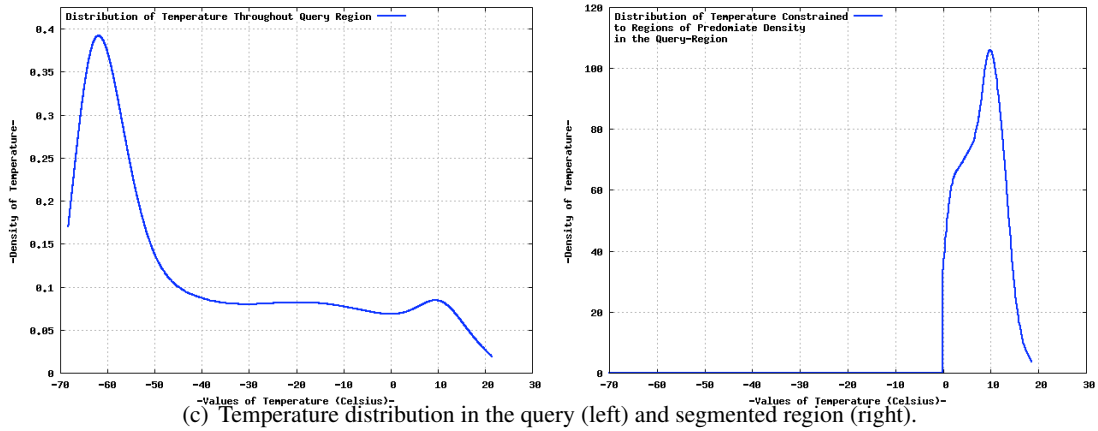
Figure 5.4: These images depict the surface surrounding a query's solution set in (a), as well as the segmentation based on predominant distribution contributions within this query region. (b) shows the regions where pressure (blue), and velocity (green) contribute most significantly to the joint distribution. In (c) I show, with the segmented region of velocity rendered transparent, the region (red) where temperature contributes most significantly to the joint distribution. Images (d) and (e) depict the result of refining the original query shown in (a) with the distribution information gathered for temperature obtained in (c). Here (d) shows where temperature is less than 0 degrees Celsius in the query region, and (e) shows the regions where temperature is between 0 and 20 degrees Celsius in the query region.



(a) Pressure distribution in the query (left) and segmented region (right).



(b) Velocity distribution in the query (left) and segmented region (right).



(c) Temperature distribution in the query (left) and segmented region (right).

Figure 5.5: Based on the example in Section 5.4.1, these histograms illustrate (top to bottom) the univariate distributions of pressure, velocity, and temperature in the hurricane dataset as found through a query region (left column) and the regions where the respective variables are predominant in the approximated joint distribution (right column) for the query. Note that the two histograms (i.e. distributions) for temperature display vastly different modalities; in Section 5.4.1 I use this isolated range of data to direct refinement of query constraints.

Analysis

I interpret the significance of these visualizations by analyzing the univariate distribution of each variable as it is defined within the variable's segmented region, versus the query's solution set.

The left column in Figure 5.5 shows the individual distributions of pressure (top), velocity (middle), and temperature (bottom) as they are found within the query's solution set. These histograms indicate that for this solution set Q, values above -1500 Pascal for pressure, above 5 mph for velocity, and below -50 degrees Celsius may play a predominant role in generating the query's joint distribution. I compare these distributions to those found in each variable's segmentation, shown at right in Figure 5.5.

In comparison to the distributions observed for pressure and velocity, the distribution for temperature's segmented region (bottom right in Figure 5.5) demonstrates the isolation of a distinct modality from the distribution of temperature observed through the query's solution space (bottom left in Figure 5.5). Specifically, the values for which temperature's univariate distribution most influences the joint distribution of the query are the range of values between 0 and 20 degrees Celsius. The strength of utilizing distribution-based segmentation is displayed in this example as the range of values from 0–20 degrees Celsius is obscured in temperature's observed univariate distribution in the query's solution space.

If the user was interested in further exploring this feature, the distribution of temperature's segmented region indicates a clear range of values for refining the query: $0 \leq \text{temperature} \leq 20$. Resubmitting the original query with this added constraint now isolates this region, as demonstrated by the new surface shown in Figure 5.4(d). Alternatively, if the user was interested in excluding this feature from the query, the user could use the same range of values to exclude this feature as shown in Figure 5.4(e).

5.4.2 Methane Dataset

I apply my distribution and segmentation method to a query that analyzes a combustion dataset modeling a lean, premixed turbulent methane flame. This dataset incorporates 20 chemical species and 6 different physical properties (velocity, temperature, pressure, etc.). The simulation

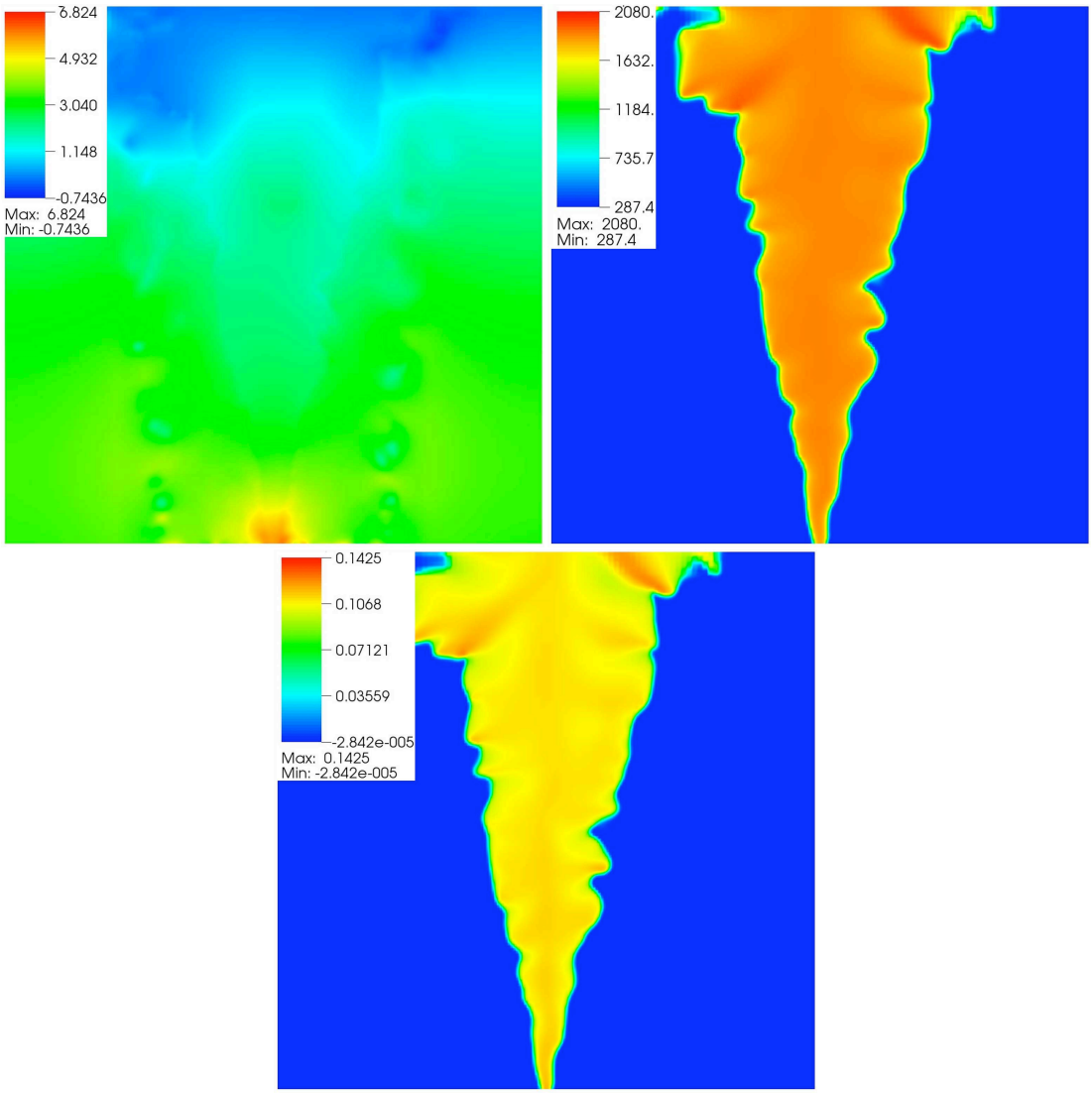


Figure 5.6: Slices taken for three variables of the methane combustion dataset. Depicted are pressure (top left), temperature (top right), and carbon dioxide (bottom). I utilize these variables for analysis in Section 5.4.2.

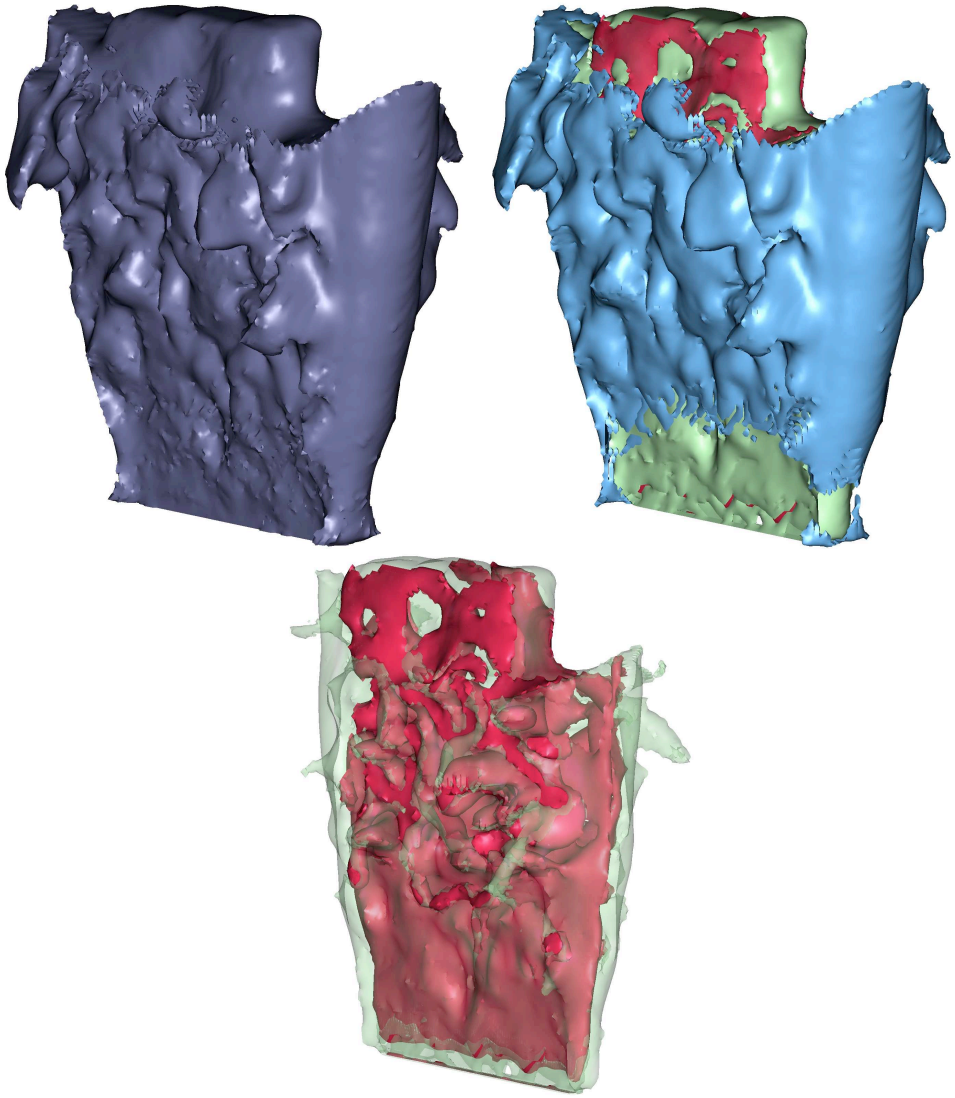
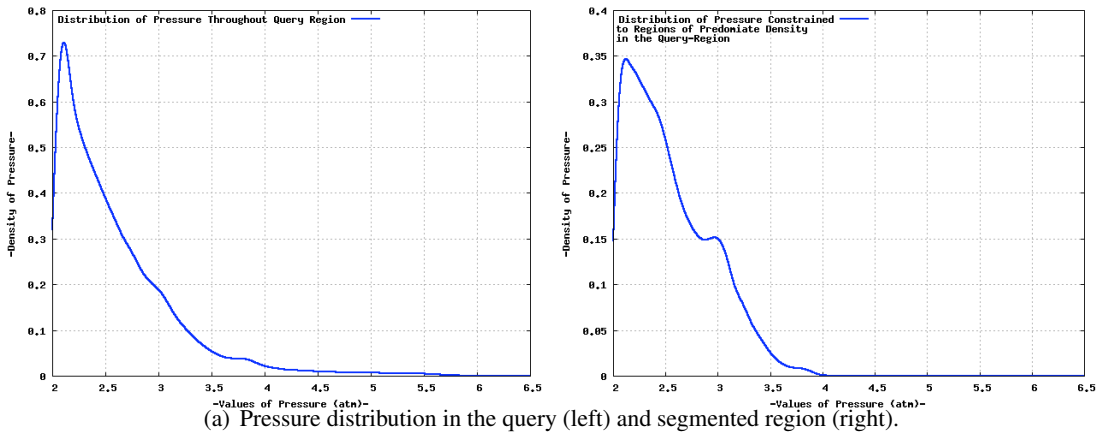
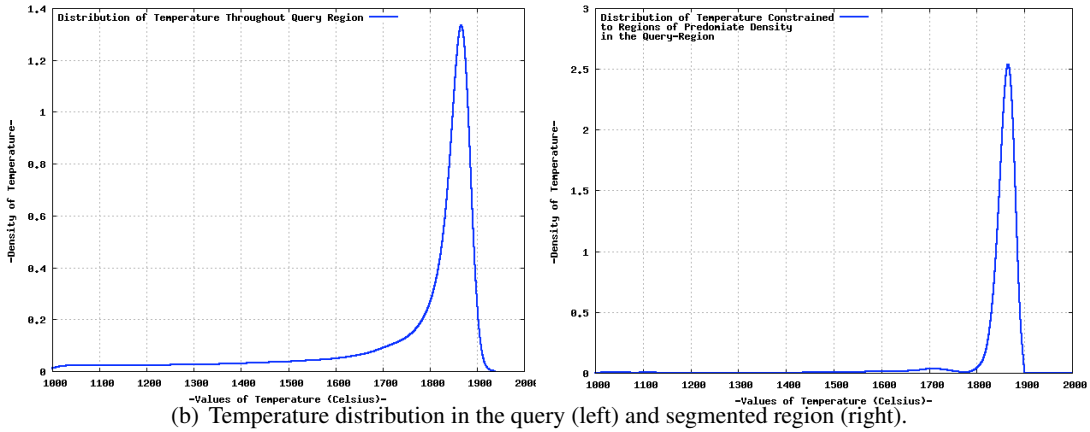


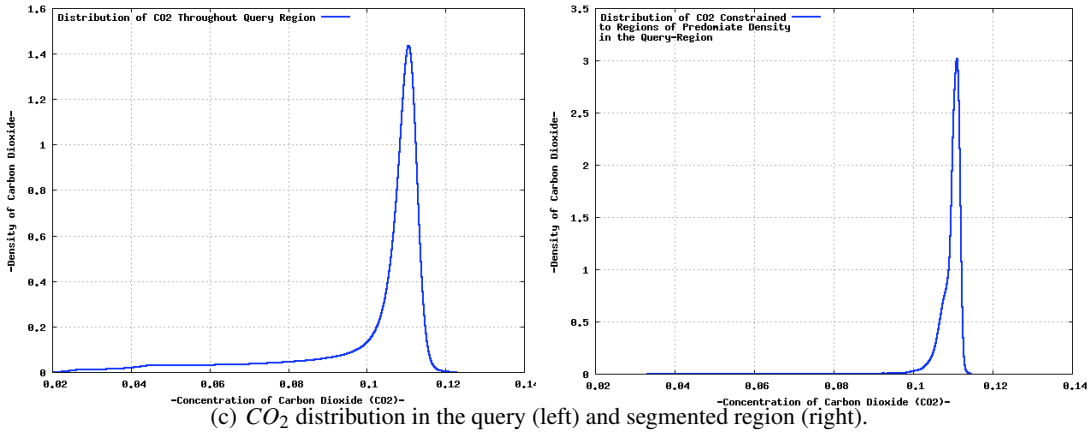
Figure 5.7: These images are visualized from a methane combustion dataset. The image at top left depicts a distribution-based surface that surrounds a query's solution set. The images at top right and bottom illustrate the segmentation for this query's joint distribution based on the maximal distribution of each variable's contribution. The figure at top right shows the regions where pressure (blue), and temperature (green) contribute most significantly to the joint distribution of the query. The figure at bottom highlights the segmented region where CO_2 (red) contributes most significantly to the joint distribution.



(a) Pressure distribution in the query (left) and segmented region (right).



(b) Temperature distribution in the query (left) and segmented region (right).



(c) CO_2 distribution in the query (left) and segmented region (right).

Figure 5.8: These histograms illustrate (top to bottom) the individual distributions of pressure, temperature, and concentration of Carbon Dioxide (CO_2) in the Methane Combustion dataset as found through the query region (left column) and the regions where the respective variables are predominant in the approximated joint distribution (right column). Note the histograms for each variable in the right column are more refined than those based on the entire query region (left column); specifically, major peaks for variables in the left column left have a higher relative distribution in the right column. Thus the blue (pressure), red (temperature), and green (Carbon Dioxide) colored regions in Figure 5.7 (center and right) are the areas where there is a narrow and refined range of values for each respective variable.

itself is simulated on a grid of size 300 x 300 x 300.

In this example, I utilize my method to analyze the data points selected from a query that constrains regions of high pressure, high temperature, and regions where the molecular concentration of CO_2 are above trace levels: pressure $\geq$ 2 atmospheres AND temperature $\geq$ 1000 Celsius AND $CO_2 \geq 1.0^{-8}$. The chemical species CO_2 is a final product of the combustion process of methane. The intent of this query is to extract data that can provide insight into how regions of increasing pressure and temperature within the combustion region propagate this chemical species throughout the flame.

The respective variables constrained by the query are depicted in Figure 5.6. In this figure, I show slices through the combustion data's pressure (top left), temperature (top right), and CO_2 concentration scalar fields (bottom).

Analysis

At the top image in Figure 5.7, I show the minimum distribution surface for the query. In the next series of images I show the segmentation obtained during the construction of the query's joint distribution. In the middle and lower images in Figure 5.7, the blue surface indicates the region where pressure contributes most to the query's joint distribution. Correspondingly, the red surface indicates the regions where CO_2 fundamentally increases the query's distribution. Wrapped between CO_2 and pressure, the segmented region for temperature is shown in green.

I next observe the univariate distributions for each variable with respect to the query's solution space, and each variable's respective segmentation region. The left column in Figure 5.8 shows the individual distributions of pressure (top), temperature (middle), and CO_2 (bottom) as they are found within the query's solution set. I compare these distributions to those found in each variable's segmentation, shown at right in Figure 5.8.

Though not as pronounced as the hurricane example, each variable's univariate distribution for their respective segmentation regions is subtly more refined. For example, a second modality (top right) has emerged for pressure in Figure 5.8 at the range of 3 atmospheres. Also, temperature and CO_2 have no distribution for values less than 1800 Celsius and 0.1 respectively (unlike the distributions shown at left in Figure 5.8).

In Figure 5.9, I apply the strategy discussed in Section 5.3.2 and examine ranges of higher

Figure 5.9: Slices oriented along the X (left) and Y (right) axis of the segmented regions of the Methane dataset. Here segments for pressure (blue), temperature (green), and CO_2 (red) are depicted only for high distribution values. Grey regions indicate areas of lower distribution values in the query's solution set.

distribution values in the query's joint distribution. In this figure I show slices through the query solution's segmented regions oriented along the X (left) and Y (right) axis. Here gray regions indicate lower distribution regions in the query's joint distribution. Colored regions correspond to areas of higher joint distribution value where pressure (blue), temperature (green), and CO_2 (red) segmentation occurs. Note the reduced representation of pressure in these images. Contrariwise, note the predominance of temperature and CO_2 indicating these variables, and the values corresponding to each variable within its segmentation, play a more predominant role in the joint distribution.

5.5 Implementation and Performance

Query-Driven Visualization (QDV) demands components that are high-preforming with respect to computation in order to support interactivity. Distribution approximations with KDE are of order $O(N^2)$ with a straightforward implementation and can be limiting upon overall performance for large N. There are methods for accelerating KDE calculations using up-front preprocessing, such as the Fast Gauss Transforms [115], Fast Multipole Method [46], and tree-based strategies [44]; however, the up-front processing is typically expensive and is amortized only if the KDE is evaluated frequently over fixed data values. For QDV applications, however, new KDE must be evaluated with

Variables Queried	select 1% (seconds)	select 2.5% (seconds)	select 5% (seconds)	select 10% (seconds)
1	1.09	8.3	27.6	109.0
2	2.2	16.4	57.2	227.0
3	3.1	23.02	78.7	301.0
4	3.8	30.4	97.53	386.0

Table 5.1: This table depicts the performance times, in seconds, for my GPU-based distribution approximation implementation. The axis are decreasing selectivity vs increasing variable count. Times include the time to access all raw data from CPU-memory, load the data to the GPU, compute the distribution, determine the segmentation (for multivariate queries), and write the solution back to CPU memory.

every ad hoc query, where the size of N for the KDE will typically be a small fraction of the total data. Paying a constant preprocessing cost for these acceleration methods (for a small and varying number of N) limits their utility for a QDV application. To accelerate my KDE implementation, I thus turn to hardware acceleration.

In this work, I compute the KDE distribution estimates and surfaces for visualization on the GPU. My KDE computation takes as input a list of data samples that pass the query. I launch a GPU thread per data sample to evaluate the multivariate Gaussian kernel in Equation 5.4. By staying on the GPU I can exploit the inherent data parallelism of the graphics hardware to accelerate the KDE computation, in lieu of transferring the data back to main memory for CPU computation.

In measuring the performance of my implementation, there are two factors that effect performance times: increasing size for query solutions sets (i.e. decreasing the query's selectivity), and increasing the number of variables for the joint distribution computation. I analyze these two metrics independently by analyzing increasingly larger subsets of data, in conjunction with queries that constrain an increasing number of variables. The performance for this test are presented in Table 5.1. The performance times are based on the hurricane dataset which consists of 8.1 million cells mapped to a grid of 300 X 300 x 90.

Table 5.1 indicates that my implementation follows an expected performance trend for an $O(N^2)$ algorithm. Additionally, this table shows that increasing the number of variables utilized to construct a distribution scales with an expected linear growth curve. In practice, I have found that these processing times are not prohibitive—once the KDEs have been computed, users are able to

interactively explore different distribution surfaces and the multivariate segmentation for analysis purposes.

Part V

Conclusion

Chapter 6

Summary

This dissertation has presented new methods that significantly advance the field of query-driven visualization (QDV). Each part has addressed a specific area important to QDV: performance (Part II), extensibility (Part III), and utility (Part IV).

In Part II of this dissertation, I have addressed the challenge of employing multi-core architectures to accelerate query evaluation and rendering performance times for QDV applications. This challenge is motivated from the perspective of QDV, but also from the perspective of the database community too. More specifically, in the next decade the evolution and predominance of multi-core architectures will significantly challenge and change the way data processing is done in the database community. As CPUs rapidly continue to become more like parallel machines, new strategies must be developed that can fully utilize the increasing thread-level parallelism, and thus the processing capabilities, of these architectures.

In Part III, I have focused on the challenge of applying QDV strategies to time-varying, adaptive mesh refinement data. This challenge requires addressing the temporally dynamic properties of AMR grids as they change in refinement throughout a simulation's duration. I have also focused on the challenge of creating temporally concurrent, multitemporal visualizations from queries that evaluate numerous timesteps in a simulation.

In the final part of this dissertation (Part IV) I have addressed the challenge of extending the utility of QDV strategies with efficient, intuitive statistical analysis techniques. Query systems have the potential to solve many large-scale visualization problems. Remarkably, little research has

been performed on the explicit study of QDV solutions. While well-characterized range queries are readily able to identify the regions where important physical events occur, e.g. combustion flame fronts, vortices, chemical reaction fronts, etc., existing query-based research offers little assistance in helping users to better understand these events. The challenge is to extend the strengths of QDV with new methods that can rapidly assist the scientist in transitioning from stages of visual-discovery into stages of visual analysis and presentation.

In the following, I briefly summarize the main novel scientific contributions that I have made to address these challenges.

In Part II, I have made several contributions to the fields of QDV and database indexing. First, I have introduced a data parallel bin-based indexing strategy (DP-BIS) for answering selection queries on multi-core architectures. I have shown that the concurrency provided by DP-BIS fully utilizes the data parallelism emerging in these architectures in order to benefit from their increasing computational capabilities. Secondly, I have presented the first strategy in database literature for answering selection queries on a GPU that utilizes encoded data. This encoding strategy facilitates a significantly better utilization of data bus bandwidth and memory resources than GPU-based strategies that rely exclusively on base data. Last, I have implemented and demonstrated DP-BIS's performance on two commodity multi-core architectures: a multi-core CPU and a GPU. I have shown in performance tests that both implementations of DP-BIS outperform the GPU and CPU-based projection index with respect to total query response times. I have additionally shown that the GPU-based implementation of DP-BIS outperforms all index strategies with respect to computation-based times.

In Part III, I have made notable contributions to the fields of QDV and multitemporal AMR visualization. First, I have developed 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. Secondly, I have used this framework to implement and demonstrate the first GPU-based QDV engine. Finally, I have demonstrated this framework's utility by constructing temporally concurrent, multitemporal visualizations. These visualizations are critical in helping scientists to better understand how important events and trends (as characterized by range queries) form and change over a simulation's duration.

In Part IV, Chapters 4 and 5, I have significantly enhanced the field of QDV by integrating efficient, intuitive statistical analysis methods into the QDV pipeline. The major contributions in Chapter 4, specifically, include techniques that extend the capabilities of QDV by providing intuitive insight in determining:

- how sets of variables in complex datasets interact throughout regions of interest, and
- the role other variables play in influencing these interactions.

In Chapter 5 I have introduced a statistics-based framework that extends the utility of QDV strategies by helping users to better understand the solutions to their queries. The core of this framework is based on a strategy that integrates non-parametric distribution estimation techniques with a new segmentation strategy to visually identify statistically important trends and features within the solution space of a query. I have shown how users can use the information obtained from the query's joint distribution and segmentation to refine the constraints over individual variables in a multivariate query. Such information is essential for users to move from stages of visual-discovery into stages of visual analysis and presentation in the scientific discovery process.

In the remaining portion of the dissertation, I outline key areas for new research in the field of QDV. While some of these topics are directly related to the subject material presented earlier in this dissertation, others present entirely new directions of research yet to be explored.

Chapter 7

Directions For Future Research

7.1 Data Parallel Compression Strategies

In the next decade, the evolution and predominance of multi-core architectures will significantly challenge and change the way data processing is done in the database community. My Data Parallel Bin-based Indexing strategy (DP-BIS) provides a parallel indexing data structure that will scale effectively with the future increase of processor cores on multi-core architectures. However, there is much room for further research in this area.

The performance of the DP-BIS index is somewhat limited by its use of encoded data. Even though the size of this encoded data is smaller than the size of the corresponding raw data, encoded data is still too large to stream to the GPU without negatively impacting performance; the performance tests in Section 2.6 effectively demonstrate this. In my current research I am exploring data parallel compression strategies to replace the bin-based encoding used in the DP-BIS index.

In general, the "right" strategy to follow in general-computation GPU work is the strategy that calls for more computation in trade for lighter bandwidth requirements. This is because technology trends for future GPUs, and multi-core architectures in general, support an *accelerated* growth in computational power over bandwidth growth [82,83]. Thus any compression strategy that significantly reduces the amount of data that is streamed to the GPU, *even if it is computationally costly to implement*, could provide significant performance benefits for the DP-BIS index—if not for current multi-core architectures, then perhaps for the next generation of hardware. Stated in other words, bandwidth forms the limiting factor on performance for most general-purpose GPU work,

especially for tasks that contain low arithmetic intensity (e.g. answering a query). By utilizing a compression strategy, I can lighten the constraints imposed by the CPU-GPU bandwidth bottleneck and, perhaps, maintain or even beat the performance of the current DP-BIS method.

The notion of a *data parallel* compression strategy, however, presents significant challenges. Claude Shannon first presented the theory of data compression in 1948 [91]. Since this time numerous compression methods have been developed, based on a wide assortment of strategies: adaptive dictionary approaches, run-length encoding, entropy encoding etc. Most of these strategies however, do not support a high level of task parallelism and thus are not suitable for parallel implementation. Compression strategies that *do* offer a high level of task parallelism are data lossy and are thus unsuitable for query processing.

The challenges of developing this data parallel compression strategy constitutes one of the primary works I am focusing on to extend QDV.

7.2 Challenges and Future Work for Adaptive Mesh Refinement Data

In Part III of this dissertation I present a new method for performing query-driven visualization of time-varying AMR data. With my new analysis and visualization approach, I construct multitemporal visualizations that convey and summarize important variable temporal trends, such as dispersion rates, or temporally-based central tendencies, in a single meaningful image.

One potential limitation of this 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 important 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 I am pursuing to address this worst-case scenario is to develop methods that optimally select the timesteps synchronized in the 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 I am 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 my future work.

Finally, AMR-based numerical simulations are not restricted to uniform axis-aligned grids; grids composed of curvilinear and unstructured cells also exist, though much less common. Applying my composite method to these non-tradition cases, i.e. addressing the challenge of time dependent visualization for unstructured AMR-based simulations, may present new challenges. Specifically, it is possible that these less-conventional grid structures will follow different re-griding approaches that will not be supported by my current compositing strategy.

7.3 Statistical Analysis and Query-Driven Visualization

7.3.1 Extending Correlation Analysis Strategies

One important extension to the work I presented in Chapter 4, is the ability to visualize correlation fields that represent relationships between multiple variables. Identifying correlation between level sets colored by the values from such fields, and the correlation represented by the fields themselves, is a challenging task. I am currently exploring an approach for analyzing three variables simultaneously. In this method I construct a cone from the gradients constructed from the three variables, and use the opening angle of the cone to indicate correlation between the three variables: regions of high correlation would be indicated by small angles, where as regions of low correlation would be indicated by a 180 degree cone (i.e. a plane).

Additionally, I am exploring ways to combine the method presented in Chapter 4 with machine learning strategies. The idea would be to develop methods that allow for the automatic identification and classification of correlation between multiple variables. In this approach, users could additionally use the generated correlation data to rapidly identify important data intervals for specific variables.

7.3.2 KDE-Based Segmentation in Future Work

I present a method in Chapter 5 that uses a statistical framework to estimate the underlying distribution of data within a query's solution. This approach allows for the construction of boundary surfaces for query regions based upon the behaviors of one or more variables. Furthermore, QDV users are able to directly visualize the structure of the query in terms of the multivariate distribution, or through a segmentation formed by the univariate distribution estimations. I demonstrate the utility of this method for QDV across two scientific datasets.

There are several avenues for extending this work. The first path I am exploring involves the use of new statistical metrics for segmenting the query's joint density. For example, Chapter 5 exclusively demonstrates segmentation based on maximal-contributions from a collection of variables. However, my preliminary research demonstrates that important visual features are also obtained from segmentations based on a mean metric as well as a metric based on furthest outlier—i.e. the variable furthest from the mean of variable distribution values.

A second avenue I am exploring is an integration of the segmentation strategy presented in Chapter 5 with the correlation field work done in Chapter 4. In this work I use textures to convey correlation information, constructed from two or three variables, on each variable's segmented surface. The principal challenge I am currently dealing with is how to guide the user to select the right variables to use in this strategy, and how to convey the results without overwhelming the user with too much data.

Bibliography

[1] Hiroshi Akiba, Kwan-Liu Ma, Jacqueline Chen, and Evatt Hawkes. Visualizing multivariate volume data from turbulent combustion simulations. *Computing in Science and Engineering*, 9(2):76–83, 2007.

[2] Ann Almgren, John Bell, Phillip Colella, Louis Howell, and Michael Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *Journal of Computational Physics*, 142(1):1–46, 1998.

[3] Sihem Amer-Yahia and Theodore Johnson. Optimizing queries on compressed bitmaps. In *Proc. of VLDB*, pages 329–338, 2000.

[4] John C. Anderson, Luke Gosink, Mark A. Duchaineau, and Kenneth I. Joy. Feature identification and extraction in function fields. In *EuroVis 2007*, pages 195–201, May 2007.

[5] Gennady Antoshenkov. Byte-aligned bitmap compression. In *Proc. of the Conference on Data Compression*, page 476, 1995.

[6] Gennady Antoshenkov and Mohamed Ziauddin. Query processing and optimization in ORACLE RDB. In *Proc. of VLDB*, pages 229–237, 1996.

[7] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2006.

[8] Chandrajit Bajaj, Valerio Pascucci, Guglielmo Rabbiolo, and Daniel Schikorc. Hypervolume visualization: a challenge in simplicity. In *Proc. of IEEE Symposium on Volume Visualization*, pages 95–102, 1998.

[9] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. The contour spectrum. In *In Proc. of the conference on Visualization*, pages 167–173, 1997.

[10] Mostafa Bamha and Gaétan Hains. Frequency-adaptive join for shared nothing machines. *Progress in Computer Research*, pages 227–241, 2001.

[11] David C. Banks and Stephen Linton. Counting cases in Marching Cubes: Toward a generic algorithm for producing substitopes. In *Proc. of IEEE Visualization*, pages 51–58, October 2003.

[12] Jacek Becla and K.-T. Lim. Report from the workshop on extremely large databases, 2007.

[13] Jacek Becla and Daniel Wang. Lessons learned from managing a petabyte. In *Proc. of Innovative Data Systems Research*, pages 70–83, 2005.

[14] J. Bell, M. Day, C. Rendleman, S. Woosley, and M. Zingale. Adaptive low Mach number simulations of nuclear flame microphysics. *Journal of Computational Physics*, 195(2):677–694, 2004.

[15] J. Bell, M. Day, I. Shepherd, M. Johnson, R. Cheng, J. Grcar, V. Beckner, and M. Lijewski. Numerical simulation of a laboratory-scale turbulent V-flame. In *Proc. of the National Academy of Science*, volume 102, pages 10006–10011, July 2005.

[16] Marsha Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, May 1989.

[17] Marsha Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, March 1984.

[18] E. Wes Bethel, Scott Campbell, Eli Dart, Kurt Stockinger, and Kesheng Wu. Accelerating network traffic analysis using query-driven visualization. In *Proc. of IEEE Symposium on Visual Analytics Science and Technology*, pages 115–122, October 2006.

[19] Praveen Bhaniramka, Rephael Wenger, and Roger Crawfis. Isosurfacing in higher dimensions. In *Proc. of IEEE Visualization*, pages 267–273, 2000.

[20] P. A. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proc. of Innovative Data Systems Research*, pages 225–237, January 2005.

[21] Greg L. Bryan. Fluids in the universe: adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, 1(2):46–53, 1999.

[22] David M. Butler, James C. Almond, R. Daniel Bergeron, Ken W. Brodlie, and Robert B. Haber. Visualization reference models. In *VIS '93: Proceedings of the 4th conference on Visualization '93*, pages 337–342, 1993.

[23] Hamish Carr, Duffy Brian, and Denby Brian. On histograms and isosurface statistics. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):1259–1266, 2006.

[24] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *Proc. of IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[25] Douglas Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[26] Roger A. Crawfis and Nelson Max. Texture splats for 3d scalar and vector field visualization. In *Proc. of IEEE Visualization*, pages 261–266, 1993.

[27] M. Day and J. Bell. Simulation of premixed turbulent flames. *Journal of Physics Conference Series*, 46:43–47, September 2006.

[28] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[29] Luc Dessart, Adam Burrows, Christian Ott, Eli Livne, Sung-Chul Yoon, and Norbert Langer. Multi-dimensional simulations of the accretion-induced collapse of white dwarfs to neutron stars. *The Astrophysical Journal*, 644:1063, 2006.

[30] David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.

[31] Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Local and global comparison of continuous functions. In *Proc. of IEEE Visualization*, pages 275–280, 2004.

[32] Rui Fang, Bingsheng He, Mian Lu, Ke Yang, Naga Govindaraju, Qiong Luo, and Pedro V. Sander. GPUQP: query co-processing using graphics processors. In *Proc. of SIGMOD*, pages 1061–1063, 2007.

[33] Ying-Huey Fua, Matthew O. Ward, and Elke A Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. In *Proc. of IEEE Visualization*, pages 43–50, 1999.

[34] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[35] Volker Gaede and Oliver Günther. Multidimension access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[36] Bogdan Georgescu, Ilan Shimshoni, and Peter Meer. Mean shift based clustering in high dimensions: A texture classification example. *Computer Vision, IEEE International Conference on*, 1:456, 2003.

[37] Markus Glatter, Jian Huang, Sean Ahern, Jamison Daniel, and Aidong Lu. Visualizing temporal patterns in large multivariate data using modified globbing. *Proc. of IEEE Trans. on Visualization and Computer Graphics*, 14(6):1467–1474, 2008.

[38] Markus Glatter, Jian Huang, Jinzhu Gao, and Colin Mollenhour. Scalable data servers for large multivariate volume visualization. *Proc. IEEE Trans. on Visualization and Computer Graphics*, 12(5):1291–1298, 2006.

[39] Luke J. Gosink, John C. Anderson, E. Wes Bethel, and Kenneth I. Joy. Variable interactions in query-driven visualization. *Proc. of IEEE Trans. on Visualization and Computer Graphics*, 13(6):1400–1407, 2007.

[40] Luke J. Gosink, John C. Anderson, E. Wes Bethel, and Kenneth I. Joy. Query-driven visualization of time-varying adaptive mesh refinement data. *Proc. of IEEE Trans. on Visualization and Computer Graphics*, 14(6):1715–1722, 2008.

[41] Luke J. Gosink, John Shalf, Kurt Stockinger, Kesheng Wu, and E. Wes Bethel. HDF5-FastQuery: Accelerating complex queries on HDF datasets using fast bitmap indices. In *Proc. of Scientific and Statistical Database Management*, pages 149–158, 2006.

[42] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: high performance graphics co-processor sorting for large database management. In *Proc. of SIGMOD*, pages 325–336, June 2006.

[43] Naga Govindaraju, Brandon Lloyd, Wei Wang, Ming C. Lin, and Dinesh Manocha. Fast computation of database operations using graphics processors. In *Proc. of SIGMOD*, pages 215–226, June 2004.

[44] Alexander G. Gray and Andrew W. Moore. Rapid evaluation of multiple density models. In *Proc. of the 9th International Workshop on Artificial Intelligence and Statistics*, 2003.

[45] Jim Gray, David T. Liu, María A. Nieto-Santisteban, Alexander S. Szalay, David J. DeWitt, and Gerd Heber. Scientific data management in the coming decade. *SIGMOD Rec.*, 34(4):34–41, 2005.

[46] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, December 1987.

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

[48] Andrew J. Hanson and Pheng A. Heng. Illuminating the fourth dimension. *IEEE Computer Graphics and Applications*, 12(4):54–62, 1992.

[49] Bingsheng He, Naga Govindaraju, Qiong Luo, and Burton Smith. Efficient gather and scatter operations on graphics processors. In *Proc. of Supercomputing*, pages 46–58, 2007.

[50] Bingsheng He, Ke Yang, Rui Fang, Mian Lu, Naga Govindaraju, Qiong Luo, and Pedro Sander. Relational joins on graphics processors. In *Proc. of SIGMOD*, pages 511–524, 2008.

[51] Hans-Christian Hege, Martin Seebass, Detlev Stalling, and Malte Zöckler. A generalized Marching Cubes algorithm based on non-binary classifications. Technical Report SC-97-05, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1997.

[52] Gerardo Hermosillo, Christophe Chefd'hotel, and Olivier Faugeras. Variational methods for multimodal image matching. *International Journal of Computer Vision*, 50(3):329–343, 2002.

[53] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91, 1985.

[54] Yannis Ioannidis. The history of histograms (abridged). In *Proc. VLDB*, pages 19–30, 2003.

[55] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of Hermite data. *ACM Trans. on Graphics*, 21(3):339–346, 2002.

[56] Ralf Kähler and Hans-Christian Hege. Texture-based volume rendering of adaptive mesh refinement data. *The Visual Computer*, 18(8):481–492, 2002.

[57] Ralf Kähler, Steffen Prohaska, Andrei Hutanu, and Hans-Christian Hege. Visualization of time-dependent remote adaptive mesh refinement data. In *Proc. of IEEE Visualization*, pages 175–182, 2005.

[58] Ralf Kähler, John Wise, Tom Abel, and Hans-Christian Hege. GPU-assisted raycasting for cosmological adaptive mesh refinement simulations. In *Proc. of Volume Graphics: Eurographics Assoc.*, pages 103–110, 2006.

[59] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Trans. on Visualization and Computer Graphics*, 8(3):270–285, 2002.

[60] Christian Ledergerber, Gaël Guennebaud, Miriah Meyer, Moritz Bächer, and Hanspeter Pfister. Volume MLS ray casting. *IEEE Trans. on Visualization and Computer Graphics*, 14(6):1372–1379, 2008.

[61] Michael D. Lieberman, Jagan Sankaranarayanan, and Hanan Samet. A fast similarity join algorithm using graphics processing units. In *Proc. of ICDE*, pages 1111–1120, 2008.

[62] Terry J. Ligocki, Brian Van Straalen, John M. Shalf, Gunther H. Weber, and Bernd Hamann. *A Framework for Visualizing Hierarchical Computations*, pages 197–204. Springer Verlag, January 2003.

[63] Lars Linsen, Tran Van Long, Paul Rosenthal, and Stephan Rosswog. Surface extraction from multi-field particle volume data using multi-dimensional cluster visualization. *In Proc. IEEE Transactions on Visualization and Computer Graphics*, 14(6):1483–1490, 2008.

[64] Witold Litwin, Marie-Anna Neimat, and Donovan A. Schneider. LH*—a scalable, distributed data structure. *ACM Trans. Database Syst.*, 21(4):480–525, 1996.

[65] Z. Liu, W. Chen, K.Q. Huang, and T.N. Tan. A probabilistic framework based on KDE-GMM hybrid model (KGHM) for moving object segmentation in dynamic scenes. In *Workshop on Visual Surveillance*, 2008.

[66] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, pages 163–169, July 1987.

[67] Claes Lundstrom, Patric Ljung, and Anders Ynnerman. Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1570–1579, 2006.

[68] Kwan-Liu 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, February 1999.

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

[70] Patrick McCormick, Jeff Inman, James Ahrens, Charles Hansen, and Greg Roth. Scout: A hardware-accelerated system for quantitatively driven visualization and analysis. In *Proc. of IEEE Visualization*, pages 171–178, October 2004.

[71] Richard Mount. The Office of Science Data-Management Challenge. Report from the DOE Office of Science Data-Management Workshops. Technical Report SLAC-R-782, Stanford Linear Accelerator Center, March 2004.

[72] Klaus Mueller, Torsten Mller, J. Edward Swan II, Roger Crawfis, Naeem Shareef, and Roni Yagel. Splatting errors and antialiasing. *Proc. of IEEE Trans. on Visualization and Computer Graphics*, 4(2):178–191, 1998.

[73] Aaftab Munshi. OpenCL Specification V1.0. Technical report, Khronos OpenCL Working Group. `http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf`, January 2008.

[74] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads programming*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1996.

[75] Gregory M. Nielson and Richard Franke. Computing the separating surface for segmented data. In *Proc. of IEEE Visualization*, pages 229–233, October 1997.

[76] Michael G. Norman, Thomas Zurek, and Peter Thanisch. Much ado about shared-nothing. *SIGMOD Rec.*, 25(3):16–21, 1996.

[77] Michael L. Norman, John Shalf, Stuart Levy, and Greg Daues. Diving deep: Data-management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science and Engineering*, 1(4):36–47, 1999.

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

[79] Patrick E. O'Neil. Model 204 architecture and performance. In *Second International Workshop in High Performance Transaction Systems*, volume 359 of *Lecture Notes in Computer Science*, pages 40–59, 1987.

[80] Patrick E. O'Neil and Dallan Quass. Improved query performance with variant indexes. In *Proc. of SIGMOD*, pages 38–49, May 1997.

[81] Jeremiah P. Ostriker and Michael L. Norman. Cosmology of the early universe viewed through the new infrastructure. *Commun. ACM*, 40(11):84–94, 1997.

[82] John D. Owens. Streaming architectures and technology trends. In *GPU Gems 2*, pages 457–470. March 2005.

[83] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Tim Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, March 2007.

[84] Sanghun Park, Chandrajit Bajaj, and Vinay Siddavanahalli. Case study: interactive rendering of adaptive mesh refinement data. In *Proc. of IEEE Visualization*, pages 521–524, 2002.

[85] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[86] J. Wenny Rahayu and David Taniar. Parallel selection query processing involving index in parallel database systems. In *Proc. of International Symposium on Parallel Architectures, Algorithms and Networks*, pages 309–314, 2002.

[87] Rajeev Raman and Uzi Vishkin. Parallel algorithms for database operations and a database operation for parallel algorithms. In *Proc. of the International Symposium on Parallel Processing*, pages 173–179, 1995.

[88] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. In *Annals of Mathematical Statistics*, volume 27, pages 832–835, 1956.

[89] Natascha Sauber, Holger Theisel, and Hans-Peter Seidel. Multifield-graphs: An approach to visualizing correlations in multifield scalar data. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):917–924, 2006.

[90] Carlos E. Scheidegger, John M. Schreiner, Brian Duffy, Hamish Carr, and Cláudio T. Silva. Revisiting histograms and isosurface statistics. *Proc. of IEEE Trans. on Visualization and Computer Graphics*, 14(6):1659–1666, 2008.

[91] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948.

[92] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, April 1986.

[93] Rishi Rakesh Sinha and Marianne Winslett. Multi-resolution bitmap indexes for scientific data. *ACM Trans. Database Syst.*, 32(3):16, 2007.

[94] Kurt Stockinger, E. Wes Bethel, Scott Campbell, Eli Dart, and Kesheng Wu. Imaging and visual analysis - detecting distributed scans using high-performance query-driven visualization. In *Proc. of Supercomputing*, page 82, 2006.

[95] Kurt Stockinger, John Shalf, E. Wes Bethel, and Kesheng Wu. Dex: Increasing the capability of scientific data analysis pipelines by using efficient bitmap indices to accelerate scientific visualization. In *Proc. of Scientific and Statistical Database Management*, pages 35–44, 2005.

[96] Kurt Stockinger, John Shalf, Kesheng Wu, and E. Wes Bethel. Query-driven visualization of large data sets. In *Proc. of IEEE Visualization*, pages 167–174, October 2005.

[97] Kurt Stockinger, Kesheng Wu, and Arie Shoshani. Evaluation strategies for bitmap indices with binning. In *Proc. of Database and Expert Systems Applications*, pages 120–129, September 2004.

[98] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. C-store: a column-oriented DBMS. In *Proc. of VLDB*, pages 553–564, 2005.

[99] Chengyu Sun, Divyakant Agrawal, and Amr El Abbadi. Hardware acceleration for spatial selections and joins. In *Proc. of SIGMOD*, pages 455–466, June 2003.

[100] Russell Taylor. Visualizing multiple fields on the same surface. *IEEE Computer Graphics and Applications*, 22(3):6–10, 2002.

[101] Timothy Urness, Victoria Interrante, Ivan Marusic, Ellen Longmire, and Bharathram Ganapathisubramani. Effectively visualizing multi-valued flow data using color and texture. In *Proc. of IEEE Visualization*, pages 115–121, 2003.

[102] Gunther H. Weber, Hans Hagen, Bernd Hamann, Kenneth I. Joy, Terry J. Ligocki, Kwan-Liu Ma, and John M. Shalf. Visualization of adaptive mesh refinement data. In *Proc. of SPIE (Visual Data Exploration and Analysis)*, volume 4302, pages 121–132, January 2001.

[103] Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, Bernd Hamann, and Kenneth 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.

[104] Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, Bernd Hamann, Kenneth I. Joy, and Kwan-Liu Ma. High-quality volume rendering of adaptive mesh refinement data. In *Vision, Modeling, and Visualization 2001*, pages 121–128, 522, November 2001.

[105] Lee Westover. Interactive volume rendering. In *Proc. of the Workshop on Volume Visualization*, pages 9–16, 1989.

[106] Lee Westover. Footprint evaluation for volume rendering. In *Proc. of SIGGRAPH*, pages 367–376, 1990.

[107] Pak Chung Wong and R. Daniel Bergeron. 30 years of multidimensional multivariate visualization. In *Proc. of Scientific Visualization, Overviews, Methodologies, and Techniques*, pages 3–33, 1994.

[108] Jonathan Woodring and Han-Wei 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.

[109] Jonathan Woodring, Chaoli Wang, and Han-Wei Shen. High dimensional direct rendering of time-varying volumetric data. In *Proc. of IEEE Visualization*, page 55, 2003.

[110] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. G. R. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Laurent, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rübel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang. FastBit: Interactively Searching Massive Data. *Journal of Physics Conference Series*, 2009.

[111] Kesheng Wu, Wendy S. Koegler, Jacqueline Chen, and Arie Shoshani. Using bitmap index for interactive exploration of large datasets. In *Proc. of Scientific and Statistical Database Management*, pages 65–74, 2003.

[112] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. On the performance of bitmap indices for high cardinality attributes. In *Proc. of VLDB*, pages 24–35, 2004.

[113] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Trans. on Database Systems*, 31(1):1–38, March 2006.

[114] Kesheng Wu, Kurt Stockinger, and Arie Shoshani. Breaking the curse of cardinality on bitmap indexes. In *Proc. of Scientific and Statistical Database Management*, volume 5069 of *Lecture Notes in Computer Science*, pages 348–365, 2008.

[115] Changjiang Yang, Ramani Duraiswami, Nail A. Gumerov, and Larry Davis. Improved fast Gauss transform and efficient kernel density estimation. In *Proc. of IEEE International Conference on Computer Vision*, volume 1, pages 664–671, 2003.

[116] Xiaotong Yuan, Bao-Gang Hu, and Ran He. Agglomerative mean-shift clustering via query set compression. In *In Proc. of the SIAM International Conference on Data Mining*, pages 221–232, 2009.

[117] Rui Zhang, Beng Chin Ooi, and Kian-Lee Tan. Making the pyramid technique robust to query types and workloads. In *Proc. of ICDE*, pages 313–325, 2004.

[118] Xiang Zhang and Jie Yang. Moving object detection based on shape prediction. *Journal of the Optical Society of America*, 26(2):342–349, 2009.