To the Graduate Council:

I am submitting herewith a dissertation written by Markus Glatter entitled "Concept-Driven Visualization for Terascale Data Analytics". I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Jian Huang, Major Professor

We have read this dissertation
and recommend its acceptance:

Jack Dongarra

Jens Gregor

James S. Plank

Pengcheng Dai

Accepted for the Council:

Carolyn R. Hodges,
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Concept-Driven Visualization for Terascale Data Analytics

A Dissertation
Presented for the
Doctor of Philosophy
Degree

The University of Tennessee,
Knoxville

Markus Glatter
May 2009

Acknowledgments

I would like to express, first, my gratitude to all faculty, staff, and fellow students at The University of Tennessee's Department of Electrical Engineering and Computer Science. Witnessing the excellent quality of the program, curriculum, and classes at this institution during my years as a student has been an inspiring and humbling experience.

I would like to acknowledge the members of my committee for not only agreeing to accept me as a Ph.D. candidate, but also for their intellectual contributions, guidance, and feedback. I value their opinions highly and am very grateful for their continuous input which has significantly enhanced the quality and completeness of this dissertation. In particular, I would like to thank Dr. Jian Huang, my Ph.D. advisor, for recognizing and nurturing my interest in the field at a time when I was still an undergraduate student, for providing an engaging atmosphere for research in the SeeLab group, for his financial support through the years, for exposure to many of his contacts in the field and the interior mechanisms and politics of it, and finally for challenging me every day to become a better researcher and colleague. I would also like to thank Dr. Jack Dongarra, Dr. Jens Gregor, and Dr. James S. Plank, for allowing me to take some of the most inspiring and challenging classes I have taken in the fields of Parallel Computing, Systems Programming, and Image Analysis.

There are several acknowledgments I would like to make in reference primarily to software developed in this dissertation, and data sets and hardware utilized to obtain many of the results herein: Sean Ahern, Visualization Task Leader at Oak Ridge National Laboratory (ORNL), and his colleagues, for continuous feedback and discussions, and

for allowing me to use the hawk and lens visualization clusters and EVEREST PowerWall at ORNL; Colin Mollenhour, Wesley Kendall, Andrew Gaston, Chad Jones, and Brandon Langley for all their efforts in helping to make my software much better; my fellow Ph.D. students Roberto Sisneros, Joshua R. New, Chris R. Johnson, and Wesley Kendall for their valuable feedback and many discussions.

This work was supported by funding provided through the Institute of Ultra-Scale Visualization (www.ultravis.org) under the auspices of the Scientific Discovery through Advanced Computing (SciDAC) program within the U.S. Department of Energy (DOE). This work is also supported in part by the National Science Foundation (NSF) grant CNS-0437508 and a DOE Early Career PI grant awarded to Dr. Jian Huang (No. DE-FG02-04ER25610). This research used resources of the National Center for Computational Science (NCCS) at ORNL, which is managed by UT-Battelle, LLC., for DOE under Contract No. DE-AC05-00OR22725.

The Terascale Supernova Initiative (TSI) data set has been provided by John Blondin, at North Carolina State University, and Anthony Mezzacappa, at Oak Ridge National Lab, under the auspices of the DOE SciDAC Terascale Supernova Initiative. The Richtmyer-Meshkov Instability (RMI) data set has been provided by Mark Duchaineau at Lawrence Livermore National Lab. Climate data has been provided by Dr. John Drake, Dr. David Erickson, and Forrest Hoffman, from the Carbon-Land Model Intercomparison Project (C-LAMP), partially sponsored by DOE SciDAC and the Climate Change Research Division of the Office of Biological and Environmental Research. MODIS (Moderate-resolution Imaging Spectroradiometer) satellite data is distributed by the Land Processes Distributed Active Archive Center (LP DAAC), located at the U.S. Geological Survey (USGS) Earth Resources Observation and Science (EROS) Center (lpdaac.usgs.gov) as a component of NASA's Earth Observing System Data and Information System.

Personally, I would like to thank the several people in my life who have taught me valuable lessons, have challenged me to become a better person, and have thus helped me to become who I am today. It would not have been possible without them. In

particular, I thank my parents and family for cultivating in me at an early age a curiosity and desire for understanding, knowledge, and education. I am grateful to have found and thank my fiancée, Hanna Bachmińska, for completing my life. Furthermore, I would like to thank my circle of friends, too many to name them all herein, on both sides of the Atlantic Ocean, for allowing me to have a life outside my studies, work, and research.

Abstract

Over the past couple of decades the amount of scientific data sets has exploded. The science community has since been facing the common problem of being drowned in data, and yet starved of information. Identification and extraction of meaningful features from large data sets has become one of the central problems of scientific research, for both simulation as well as sensory data sets. The problems at hand are multifold and need to be addressed concurrently to provide scientists with the necessary tools, methods, and systems. Firstly, the underlying data structures and management need to be optimized for the kind of data most commonly used in scientific research, i.e. terascale time-varying, multi-dimensional, multi-variate, and potentially non-uniform grids. This implies avoidance of data duplication, utilization of a transparent query structure, and use of sophisticated underlying data structures and algorithms. Secondly, in the case of scientific data sets, simplistic queries are not a sufficient method to describe subsets or features. For time-varying data sets, many features can generally be described as local events, i.e. spatially and temporally limited regions with characteristic properties in value space. While most often scientists know quite well what they are looking for in a data set, at times they cannot formally or definitively describe their concept well to computer science experts, especially when based on partially substantiated knowledge. Scientists need to be enabled to query and extract such features or events directly and without having to rewrite their hypothesis into an inadequately simple query language. Thirdly, tools to analyze the quality and sensitivity of these event queries itself are required. Understanding local data sensitivity is a necessity for enabling scientists

to refine query parameters as needed to produce more meaningful findings. Query sensitivity analysis can also be utilized to establish trends for event-driven queries, i.e. how does the query sensitivity differ between locations and over a series of data sets. In this dissertation, we present an approach to apply these interdependent measures to aid scientists in better understanding their data sets. An integrated system containing all of the above tools and system parts is presented.

Contents

1 Introduction **1**

1.1 Overview 1

1.2 Relevant Previous Work and Classification 3

1.2.1 Scientific Visualization and Information Visualization 3

1.2.2 Query Driven Visualization 4

1.2.3 Data Partitioning and Load Balancing 5

1.2.4 Database Management in Visualization 6

1.2.5 Patterns in Volume Visualization 8

1.2.6 Visualization Specific Languages 8

1.2.7 Volume Visualization 9

2 Scalable Data Servers for Visualization of Large Multivariate Data **11**

2.1 Introduction 12

2.2 Our System 14

2.2.1 Design Considerations 14

2.2.2 Data Distribution and Preprocessing 15

2.2.3 Server-side Data Structure 19

2.2.4 Query Handling and Complexity of Query 23

2.2.5 The Communication Interface 26
2.3 System Deployment and Results 27
2.3.1 Deployment / Requirements 27
2.3.2 Datasets and Pre-Processing 27
2.3.3 Timing Results 29

3 Visualizing Temporal Patterns using Textual Pattern Matching 33
3.1 Introduction 33
3.2 Overview 36
3.2.1 Data Organization 36
3.2.2 Pattern Matching and Syntax 38
3.2.3 Matching and Querying 41
3.3 Results 42
3.3.1 Global Climate Modeling Simulation 43
3.3.2 Turbulent Combustion Simulation 51
3.3.3 Performance 55
3.4 Discussion and Evaluation 56
3.4.1 Integration of Data Management 56
3.4.2 Query Generation 57
3.4.3 Temporal Pattern Matching vs. Traditional Techniques 59

4 Event Sensitivity Driven Visualization of Scientific Data 62
4.1 Introduction 63
4.2 Our Approach 65
4.2.1 Overview 65

4.2.2 Syntax of Query Language 66
4.2.3 System Design 66
4.2.4 Event Sensitivity Analysis 68
4.2.5 Event Trend Sensitivity over Time Series 71
4.3 Results 72
4.3.1 Visualizing Event Sensitivity in Scientific Data Sets 74
4.3.2 Visualizing Temporal Trends in Climate Data Sets and Sensory Data 79
4.3.3 Timing and Scalability Results 83

5 Conclusions **86**

Bibliography **89**

Vita **99**

List of Tables

2.1 Load imbalance for different queries from TSI data (6 time steps, $864 \times 864 \times 864$, 11 attributes per voxel) in a configuration with 20 data servers. 32

3.1 Performance results on 20 compute servers. "Running time" is the wall clock time between query invocation and receipt of all voxels. "# Voxels considered" is the average number of matching voxels returned by each server. "# Queries generated" is the number of specific queries generated by parsing and interpreting the query specified. 55

4.1 Timing results for MODIS data queries with different resolutions for the continental U.S. The final run utilized 16,384 processors in 1,024 groups and contained more than 5.5 billion queries. It lasted almost one hour. . . . 85

List of Figures

2.1 In-situ Preprocessing 18

2.2 An illustrated M-ary search tree. 21

2.3 An illustration of server-side query handling using an M-ary search tree. . . 24

2.4 Pseudo code describing how an out-of-bound compound key is "carried" back into range. 24

2.5 Measured rates of query using queries of different numbers of relevant voxels. 31

3.1 Tokens defined for lexical analyzer (*flex*) 40

3.2 Grammar rules (*bison* grammar file) 40

3.3 Northern hemisphere colored by temporal mark `T` representing the month as labeled in Figure 3.5. Variation between years highlighted as red circles. 44

3.4 Northern hemisphere colored by temporal mark `T` 46

3.5 Legend for 11 month-to-month changes (Figures 3.3, 3.4, and 3.12) 48

3.6 Two-bit correlation of TSA and TV 48

3.7 Two-bit correlation of FCEV and FSH_V 48

3.8 Two-bit correlation of H2OCAN and RAIN 49

3.9 Image generated using a splatting renderer (with shading disabled). 52

3.10 A lit sphere rendered for each individual data item better conveys the three-dimensional relationships. 53

3.11 Close-up of an extinction region. 54

3.12 2m air temperature data of the year 2050, colored by temporal mark T representing the month as labeled in Figure 3.5 58

3.13 Traditional parallel coordinates plot of the data shown in Figure 3.3(a), colored by the query number as labeled in Figure 3.13(d) 60

4.1 Tokens defined for lexical analyzer (*flex*), with elements concerning λ highlighted. 67

4.2 Grammar rules (*bison* grammar file), with elements concerning λ highlighted. 67

4.3 An example parameter analysis for a single location, colored by the time step in which the given event has been found. A refined $\Delta\lambda$ leads to more accurate results (right figure). 73

4.4 Local extinction event for a single data location and varying values of OH and mixture fraction, colored by time step. 76

4.5 Data locations exhibiting extinction characteristics and specified χ values, colored by time step of event occurrence. 78

4.6 Analysis of the transition between months April and May for "Control(CASA)" and "Full(CASA)". Measurements are displayed as $\Delta\lambda$ per annum. 79

4.7 Analysis of the transition between months April and May for "Control(CN)" and "Full(CN)". Measurements are displayed as $\Delta\lambda$ per annum. 80

4.8 Temporal trends in MODIS satellite data for early March exhibit clear structures of events shifting in time. The red/yellow cluster marked outlines New York's Adirondack National Park Reserve. Measurements are displayed as $\Delta\lambda$ per annum. 82

4.9 Scalability Results for the IPCC_A2 data set for a varying number of data server groups. Each run consists of about 6.8 million queries. 84

Chapter 1

Introduction

1.1 Overview

Over the past couple of decades, the amount of scientific data sets has exploded, both in number and size. Scientific simulation data sets now routinely break the terabyte barrier. The Petascale Data Storage Institute (PDSI) at Carnegie Mellon University estimates that by 2011 supercomputers' performance will exceed 10 petaflops (10 quadrillion floating point operations per second) and will output files of several terabytes per second. Sensory data sets are also exceeding larger and larger limits, e.g. NASA's Land Processes Distributed Active Archive Center (LP DAAC) currently holds satellite data in excess of 1.5 petabytes, available to the public. Consequently, the science as well as the visualization community have been facing the common problem of being drowned in data, and yet starved of information. Identification and extraction of meaningful features from large data sets has become one of the central problems of scientific research, for both simulation as well as sensory data sets. The problems at hand are multifold and need to be addressed concurrently to provide scientists with the necessary tools, methods, and systems.

First, the underlying data structures and management need to be optimized for the kind of data most commonly used in scientific research, i.e. time-varying, multi-

dimensional, multi-variate, and potentially non-uniform grids. In order to handle today's massive data sets, a data management system must meet the basic prerequisites of (1) small storage cost, (2) ad hoc query support, and (3) reasonable latency and throughput performance. The implications of these requirements are (a) no unnecessary data duplication, (b) a transparent, self-explanatory query structure, and (c) use of sophisticated underlying data structures and algorithms.

Second, in the case of scientific data sets, facile queries are not a sufficient method to describe subsets or features. In the case of time-varying data sets, many features can generally be described as local events, i.e. spatially and temporally limited regions with characteristic properties in value space. While scientists will often know exactly what they are looking for in a data set, at times they cannot formally or definitively describe their concept well to computer science experts, especially when based on partially substantiated knowledge. Hence, feature specification is as big a challenge as displaying the results. Scientists need to be enabled to query and extract such features or events directly and without having to rewrite their hypothesis into an inadequately simple query language. A more sophisticated feature-oriented query language or interface is required.

Third, tools to analyze the quality and sensitivity of the event queries themselves are required. Understanding local data sensitivity is a necessity for enabling scientists to refine query parameters as needed to produce more meaningful findings. A second aspect of query sensitivity analysis is the data's sensitivity to a query, i.e. how does the query sensitivity differ between locations and over a series of data sets. For instance, analyzing a time series of data sets with regard to a query's sensitivity enables scientists to establish trends in events shifting through parameter space over the course of the series.

Finally, as these measures are dependent on one another, they have to be applied and act in concert to aid scientists in better understanding their data sets. An integrated system containing all of the above tools and system parts is required.

In this dissertation, we present a novel approach to the challenges as aforemen-

tioned. The approach presented in this dissertation covers several fields of computer science. Thus, we find previous relevant work from a number of different fields relevant to our research. We classify and summarize these findings in the following sections, and compare our methods and tools to these approaches.

1.2 Relevant Previous Work and Classification

1.2.1 Scientific Visualization and Information Visualization

Due to traditional reasons, visualization as a general area has two distinctive roots: scientific visualization and information visualization. Although both fields are now gradually overlapping each other, each still retains their traditional identity, particularly when considering their driving application, targeted computing platform, core techniques and most pressing future research challenges.

Scientific visualization is mainly concerned with the visualization of spatiotemporal data sets common in most scientific domains, such as climate science, meteorology, medical science, and physics. Realistic renderings of such two- or three-dimensional time-varying data volumes or surfaces, including illumination and shading, are emphasized [Friendly and Denis, 2001]. More generally, scientific visualization utilizes modern computer graphics capabilities to produce visual images serving as an aid in understanding of highly complex, large scale numerical representation of scientific results [McCormick, 1988]. It is a task driven (rather than data driven) approach concerned with the extraction of patterns, features, and interdependencies from a data set.

In contrast, information visualization mainly focuses on visually representing non-numerical abstract data sets without a spatiotemporal structure, e.g. unstructured textual information, point clouds in high-dimensional space, or hierarchical classification data [Munzner, 2002,Card et al., 1999]. Information visualization utilizes graphical techniques to aid understanding, analysis, and knowledge acquisition. Typical applications

that information visualization techniques are applied to include data mining, financial data analysis, graph drawing and display of hierarchical information.

The approach presented in this dissertation is geared toward multivariate spatiotemporal data sets used in scientific visualization. With sensory and simulation data sets now commonly breaking the terabyte barrier, scientific visualization in particular requires new and more complex methods and algorithms to organize, extract, and visualize data. While our methods do not strictly require numerical data in a spatiotemporal structure, the novel capabilities, such as large scale data organization, extraction of complex temporal events with uncertainty, and analysis of parameter sensitivities within quantitative specifications, are solely within the context of scientific visualization. Typical information visualization systems of today lack comparable data intensive capabilities, and do not face the same kind of pressing challenges, such as the need of an elegant treatment of uncertainty and parameter sensitivity caused by a higher need of parallel automation.

1.2.2 Query Driven Visualization

Query-driven visualization (QDV) is still a relatively new approach in visualization research, yet much potential has already been demonstrated with this approach [Stockinger et al., 2005, Gosink et al., 2007, Rübel et al., 2008] for the intuitive interface it offers for facilitating understanding of complex multivariate datasets.

Query-driven visualization is most commonly based on the concept of compound range queries, similar to basic SQL queries in database management systems (DBMS). QDV enables the end user to perform substantial data subsetting, allowing a more flexible analyses, especially of large data sets. As demonstrated by Gosink et al. [Gosink et al., 2007], one can effectively combine correlation coefficients and distribution function using compound range query-driven visualization.

The underlying data structures, however, usually differ significantly from the indexing schemata of regular database management systems. Widely used data structures

are predominately based on the binary tree and its extensions to spatial and temporal dimensions. Examples include interval tree [Cignoni et al., 1997], k-d tree, quadtree, octree [Samet, 1990], branch-on-demand octree [Sutton and Hansen, 2000], etc. The binary tree has been favorable for reasons including: (i) simplicity and (ii) optimal asymptotic performance while balanced. Unfortunately, the storage overhead is a major technical hurdle against its use for handling large numbers of attributes. This overhead occurs whether partitions are simultaneous, as in the case of octree for three dimensions, or interleaved, as with k-d tree. In addition, with a tree of such complexity, balancing the entire tree would be a grand challenge. But without a balanced tree, optimal performance cannot be guaranteed. Thus, it is not feasible to use a binary tree based data structure in our system.

Methods that are not based on trees to accelerate searches have also been proposed, such as bitmap indexing used in [Stockinger et al., 2005, Gosink et al., 2007]. While the use of bitmap indexing in visualization has recently been extended to operate in parallel [Rübel et al., 2008], load balancing results are never discussed or focused on.

Our hybrid approach combines the efficient search algorithms of tree-based methods with the low storage overhead of a linear list of data. First, we store all data in a sorted linear list in memory. On top of the linear list, we employ a search tree structure, a complete B-tree with a fixed number of child nodes at each node and a fixed depth, thus a fixed number of leaf nodes. Each leaf node of the tree points to a data item in the sorted list. In order to avoid the high storage overhead, our tree structure indexes only 1 in 10 to 1000 data items from the list. Due to this, the overhead of the tree structure is very low, typically less than 1% of the size of the linear list that contains the actual data.

1.2.3 Data Partitioning and Load Balancing

In order to handle large datasets efficiently, parallelism is indispensable. Using an approach that allows to apply load balancing to a query-driven visualization system can be

rather difficult. This is because for general data queries, it is often difficult to predict the specific subset of data returned. Many previous researchers in parallel visualization resorted to partitioning large datasets into volume blocks or bricks; a block is the finest level of granularity of parallel I/O [Yu et al., 2004], parallel rendering [Kniss et al., 2001b], data culling [Gao et al., 2005] and multi-resolution data management [Shen et al., 1999].

Block-based data organization has already significantly improved memory efficiency when handling large data. However, block-based methods still incur overhead storage and transfer costs by returning a data block even if only single data point in a block matches a query. For large scale data sets, these costs may be significant, thus using blocks as a unit of data processing can still be too expensive. We address this issue by delivering only the data points actually matching a user-input query on the fly.

In this work, we develop and employ the concept of data items. A data item stores a single data points information including its spatial and temporal location, and all of its attributes' values. As data items are independent of each other or any other kind of structural meta information, load balancing of data items among data servers can then be employed on a level of singular data items, and queries will only return matching data items without any unneeded information attached to it. In order to achieve a near-optimal overall load balance, data items are distributed among data servers in a manner that guarantees high levels of load balance for every boolean range query.

1.2.4 Database Management in Visualization

As mentioned, the amount and size of scientific data sets is constantly growing, causing various issues for scientists seeking to extract information from large data sets. As this has been a central problem for scientific visualization and visual analytics, database management approaches to overcome this challenge have been considered and evaluated.

Musick et al. [Musick and Critchlow, 1999] evaluated both relational as well as object-relational database management systems for supporting interactive computational data analysis. They concluded that a reasonable data management approach for scientific computing must meet three prerequisites: (1) small storage costs; (2) standards-based ad hoc query support; and (3) reasonable throughput (Mb/s) performance. Combinations of two of said prerequisites are supported by current software approaches. Binary large objects ("blobs"), for example, consume very little overhead storage, have reasonable throughput performance, but do not provide SQL access to its contents. On the other hand, holding a copy of the binary file outside the database instead mitigates this disadvantage by at least doubling the storage cost. However, both ad hoc query capabilities and direct data access for visualization will see no performance degradation.

Evaluating relational database management systems (RDBMS) specifically for their suitability in Earth sciences, Kochevar [Kochevar, 1993] concludes that the relational model is inadequate as a basis for scientific data management. The relational model of an unordered set of data elements does not easily fit the kind of data found in Earth sciences, e.g. satellite imagery. Kochevar suggests to use the notion of the *fiber bundle* for scientific data, mathematically representing a space which is a Cartesian product of a base space and a fiber space. Conceptually, a copy of the fiber space is attached to each element of the base space. Both base and fiber spaces can have an arbitrary topology with or without a metric defined on it, i.e. there may not be any coordinates defined for them. Such a space can just be a finite set of elements without any implicit relationship between members of the set.

To date, a reasonable data management system for scientific computing meeting all prerequisites as outlined has not been established, forcing scientists to implement their own proprietary solutions. General data management systems are hardly used in the field of scientific computing.

1.2.5 Patterns in Volume Visualization

Many current works in volume visualization directly require that features be exactly defined before a visualization can be created. As an example, in the extraction of an isosurface, the isovalue can be considered a univariate pattern.

More sophisticated patterns are often procedurally specified, although interacting with visualizations through graphical user interfaces has been very popular in many areas. For instance, Kniss et al. proposed to use procedurally specified Gaussian functions to describe the parts of a multivariate volume that are of interest [Kniss et al., 2001a]. Univariate temporal patterns have also been specified using the procedural approach [Woodring et al., 2003].

These procedural methods are very effective in revealing features that are already understood in detail. However, the specification of high-level features using procedural methods can often become cumbersome.

Few existing works address temporal patterns in particular. Temporal patterns can be described as local events, i.e. spatially and temporally limited regions with characteristic properties in value space. In the case of scientific data sets, facile boolean range queries are not a sufficient method to describe such patterns. Our work provides a novel method of converting partially-defined queries into many specific queries that are then resolved by a range query system. To enable scientists to create such "meta queries", we define and employ a feature-oriented query language.

1.2.6 Visualization Specific Languages

The use of a mini programming language in the design of visualization user interfaces is not a novel idea. Admirable successes have already been reported by systems such as prefuse [Heer et al., 2005] and Scout [McCormick et al., 2004].

Prefuse is a typical example of modern information visualization and analytics. It provides a rich set of information visualization functionality and provides very pow-

erful graphical user interfaces. An SQL-like query language has been implemented within prefuse to enable a powerful method to formally specify interactive information queries [Heer et al., 2005].

As graphics processing unit (GPU) computing has become very popular for high-performance visualization, many current system capabilities are becoming GPU-based, whether directly or indirectly. Current GPU-based volume rendering systems often offer the ability to do run-time alternation of the shader programs through user interaction. Scout [McCormick et al., 2004] is a hallmark example of this kind.

Our intent is to be complementary to existing methods by focusing on an orthogonal direction. In this work, we develop a textual interface that allows uncertainty to be used in the specification of patterns of interest, particularly ones that are temporal and multivariate in nature.

1.2.7 Volume Visualization

Volume rendering has been widely studied, primarily targeting datasets with a single scalar variable. To date, a variety of algorithms have been proposed to render a volume, including raycasting [Levoy, 1988, Tiede et al., 1998], splatting [Huang et al., 2000, Mueller et al., 1999], 3D texture hardware based rendering [Cabral et al., 1994, Kniss et al., 2001a] and shear-warp [Lacroute and Levoy, 1994]. Among those, ray-casting, splatting and 3D texture based rendering offer the best rendering quality, with each being most efficient for specific scenarios [Meissner et al., 2000].

Splatting [Huang et al., 2000, Mueller et al., 1999] is most efficient when only a sparse subset of an entire dataset is non-transparent. This is advantageous for all applications that are primarily concerned with features composed of interval volumes as shown in [Kniss et al., 2001a]. Unfortunately, a shortcoming with previous splatting algorithms stems from its lack of well-studied methods to dynamically construct the sparse volume on the fly, which hinders its application in volume rendering systems

requiring dynamic user interactions. However, with the support of advanced query-based techniques like the ones demonstrated by Stockinger et. al. in [Stockinger et al., 2005] and our work presented herein, splatting is an ideal front-end renderer to run on client machines.

In the following chapters of this dissertation, we will present our novel approaches in detail. In Chapter 2, we propose a data structure and management system for scientific data sets that is geared toward massively parallel modern supercomputers with high-speed networks. We present a regular expression-based approach to the extraction of features in Chapter 3. Chapter 4 focuses on the analysis of the quality and sensitivity of such event queries, as well as utilizing the techniques to extract trends in events over the course of a data set series. We conclude in Chapter 5 where we evaluate the methods and tools described herein and discuss further work.

Chapter 2

Scalable Data Servers for Visualization of Large Multivariate Data

Volumetric datasets with multiple variables on each voxel over multiple time steps are often complex, especially when considering the exponentially large attribute space formed by the variables in combination with the spatial and temporal dimensions. It is intuitive, practical, and thus often desirable, to interactively select a subset of the data from within that high-dimensional value space for efficient visualization. This approach is straightforward to implement if the dataset is small enough to be stored entirely in-core. However, to handle datasets sized at hundreds of gigabytes and beyond, this simplistic approach becomes infeasible and thus, more sophisticated solutions are needed. In this work, we developed a system that supports efficient visualization of an arbitrary subset, selected by range-queries, of a large multivariate time-varying dataset. By employing specialized data structures and schemes of data distribution, our system can leverage a large number of networked computers as parallel data servers, and guarantees a near optimal load-balance. We demonstrate our system of scalable data servers using two large time-varying simulation datasets.

2.1 Introduction

Today's scientific research frequently involves large datasets that are hundreds of gigabytes or more. In order for visualization to adequately handle datasets of that scale, techniques targeting specific applications are often needed. In this work, we would like to focus on large multivariate volume visualization, in which the variables may either come directly from the simulation or be subsequently derived from the simulated variables. This application is important due to its utility to basic scientific research. It is also interesting because it brings forth a grand challenge for today's computing infrastructure.

The community already faces a growing disparity between the available bandwidth in the computing infrastructure and the total amount of data to be processed for visualization. Introducing k variables on each voxel further exacerbates the disparity by a linear factor of k. As a real-world example, a recent astrophysics simulation of a supernova explosion [Blondin et al., 2003] generates a 300 time step dataset of a spatial resolution of 864^3. This size is already considered large by current standards. However, with research questions of significance in astrophysics simultaneously involving variables such as velocity, tangential velocity, angular momentum, entropy, and various derivatives of those variables, there are indeed few existing systems that can adequately support the ensuing visualization needs.

In designing our system, we leverage two basic approaches: parallelism and data culling. To handle datasets on the scale of hundreds of gigabytes, parallelism with excellent scalability seems to be the most plausible approach. In addition, our undertaking aims to intelligently reduce the required amount of data processing and be able to do so as early during the visualization pipeline as possible, i.e. effective data culling.

Our main design concept is to develop parallel data culling on the granularity of individual voxels. The data culling process is driven by compound boolean queries. Very recently, researchers have demonstrated the great potential of this design and reported

early successes in using compound queries in the context of multivariate volume visualization [Stockinger et al., 2005]. However, they have not yet extended their system to operate in parallel and handle very large datasets.

We explore methods to use a large number of networked but independent computers as parallel data servers to support efficient visualization, with two conceptual steps: (i) a selection module in the form of compound boolean range queries to cull unnecessary data processing early in the visualization pipeline and (ii) any visualization approach to finally render the relevant voxels.

After all voxels matching specified queries have been selected through parallel data culling, there could be several potential ways to visualize or render the selected voxel data. As an example, we have implemented hardware accelerated splatting [Huang et al., 2000] with a procedural transfer function module and parallel coordinate visualization [Tory et al., 2005]. Both techniques, in their classic forms, had difficulty dealing directly with large datasets. Other uses of our scalable parallel data servers could include, for instance, support for procedurally defined high-dimensional transfer functions, first proposed by Kniss et. al. [Kniss et al., 2003]. In that context, we could easily adapt our system to handle procedural Gaussian transfer functions. All that is needed is to add a selection step to query all voxels in the rectangular domain of each high-dimensional Gaussian function and then use the Gaussian functions to assign color and opacity to voxels on the fly.

As for the actual data management, the entire dataset is distributed among all data servers according to space filling curve order in the high-dimensional attribute space. Load-balancing among data servers can be achieved, independent of the types of compound boolean range queries. Each data server leverages a data structure similar to B-tree to maintain all voxels distributed to it. As a result, we achieve a very compact structure of meta data (usually less than 0.5% in size, as compared to the size of the entire dataset). A client can query the data servers and receive queried results over the network. We refer to the selected voxels as *relevant voxels*.

Since our focus here is to handle large multivariate data, the main results presented are from visualizing the supernova simulation we described earlier. Specifically, our visualization runs involve eleven variables per voxel, with $864 \times 864 \times 864$ spatial resolution, over 6 time steps. The entire set used is over 100 GB in size. In order to demonstrate that the scalability of our system is not dataset-sensitive, we also included results obtained from a $1024 \times 1024 \times 960$ version of the Richtmyer-Mevhkov Instability (RMI) simulation [RMI, Lawrence Livermore National Laboratory,] done at Lawrence Livermore National Lab.

2.2 Our System

2.2.1 Design Considerations

Large, multivariate, time-varying datasets sized at hundreds of gigabytes are increasingly common. Such datasets are too large to be held in a single machine's main memory on most computers. Since the transfer rates offered by local disk storage are considerably slower than today's high speed networks, it is often advantageous to stream data from networked data servers which hold a portion of a large dataset in main memory for fast data retrieval.

We present a scalable system of parallel data servers that provide efficient parallel data culling on large quantities of data. In our system, the data is distributed among a system of parallel data servers, each of which holds a part of the data in main memory. The system achieves high throughput by employing specialized, efficient data structures and load-balancing schemes on the server side. The user's queries to the servers consist of compound boolean range queries where multiple ranges can be defined per data variable. Please note that time and spatial coordinates of voxels are not treated differently from other attributes.

2.2.2 Data Distribution and Preprocessing

Attribute-Space Hilbert Curve

For our system, we chose individual voxels to be the targeted level of granularity. A voxel is identified by its *attributes*. These attributes include all the *variables*, such as density or entropy, on the voxel as well as the *time step* t and *coordinates* (x,y,z) of the voxel. In any dataset, no two voxels will have all of the attributes being exactly the same, because (x,y,z,t) already uniquely identifies a voxel. However, voxels with a subset of attributes being of the same values would still be common.

All attributes are stored as unsigned shorts through a binning process, or quantization, process. This allows 2^{16} different bins for each attribute. Thus, data with a precision of 16 bits or lower can be converted without a loss of precision, while data with a greater precision will have to be scaled to match the range of the unsigned short datatype. Our system could easily switch to use unsigned integer as the precision of the binning process. With 11 attributes (i.e. 7 simulated or later derived variables, 3 spatial coordinates and 1 time step id), a voxel's information is referred to as a *data item*, which requires 22 bytes of storage.

A selection is composed of compound boolean range queries and constitutes specific criteria for data culling. Each range corresponds to a particular attribute of interest. To process a "selection", we must quickly search for voxels with the corresponding attributes within the specified ranges. To do so, all data items residing on a server are sorted by each voxels attributes, with the first attribute being the most significant value, and each following attribute being less significant than its predecessor. For instance, a data item with three attributes $(1,2,3)$ is sorted in front of a data item containing $(1,2,4)$, but behind a data item $(1,1,2)$. Upon sorting, all data items are indexed by a search data structure similar to B-tree (see Section 2.2.3).

The storage overhead of the search structure is typically only a very small fraction, e.g. 0.5%, of the entire dataset's storage space. This storage need is, however, indepen-

dent on the dataset's size. (see 2.2.3 for details). By making use of this adapted data structure we achieve very low overhead during searching and data culling, so that both querying and visualization are greatly accelerated.

A straightforward way to distribute data items to parallel data servers is to do so randomly. However, randomness does not necessarily ensure good load-balancing in all scenarios. Since a selection is most likely to query for data items with similar values, we would like the data items' distribution to satisfy the following requirement: If the attributed values of two data items do not differ significantly, they should be distributed on different servers.

This heuristic implies that data items adjacent to each other in the attribute space should be distributed to different servers. We implement this heuristic by traversing the attribute space in a locality-preserving manner, and apply a round-robin distribution scheme during the traversal. This way, we minimize the problem of data items with similar attribute values "piling up" on a few servers.

In particular, we traverse the high dimensional space in order of the discrete Hilbert space-filling curve, which comes close to achieving optimal locality [Gotsman and Lindenbaum, 1996]. A traversal along the Hilbert curve can be considered a mapping, or serialization, of k-dimensional vectors of integer values into a single integer value (Hilbert index). As demonstrated in [Pascucci and Frank, 2001], the great locality of Hilbert space-filling curves can indeed be leveraged to achieve great performance when handling a high-dimensional space in visualization.

Here k is the number of attributes per voxel (including time step, spatial dimensions, and all original and derived data attributes). Assuming 16-bit precision per attribute, such a Hilbert curve will be of an exponential length (2^{16^k}), i.e. (65536^k). For practical reasons, instead of traversing through such a long curve, we sort all data items using the Hilbert index number of each point in the high-dimensional value space. This sorting step must be implemented as an external algorithm due to the large quantity of data involved.

Because the Hilbert curve only comes close to optimal locality, it is mathematically impossible to prove that our data distribution scheme based on Hilbert curve in k-dimensional attribute space is optimal in a strict mathematical sense. However, as we will show in Table 2.1 (Section 2.3.3), our data distribution method manages to achieve a close approximation to a perfectly load-balanced system.

Data Preprocessing

In addition to computing attributes in parallel, there are more time consuming tasks to prepare a large multi-variate data visualization. Particularly, in our system it is a time-consuming process to organize data, originally indexed by temporal and spatial coordinates, according to order in attribute space. For hundreds of gigabytes of data, external sort algorithms implemented on hard-drives still can not offer sufficient performance.

In our system, we preprocess the input data utilizing a group of networked computers in parallel. In fact, it is exactly the same set of networked computers that will later be used as the parallel data servers. From this perspective, the external mergesort is computed *in situ*.

As illustrated in Figure 2.1, (1) the dataset is initially partitioned and stored on the file system as spatial blocks arranged according to time steps. (2) We then distribute the volume blocks to all networked computers in a round-robin fashion. (3) Each computer then locally computes all derived attributes (e.g. gradients) and sorts all data items it has into Hilbert curve order. At the end of this process, each computer, C_i, holds a different linear list of sorted data items.

After each data server is ready, (4) the client machine, acting as the orchestrator of an external *mergesort*, requests the first 32 MB from the linear list stored on each networked computer. (5) The client machine then starts an internal mergesort. (6) The resulting sorted list is then re-distributed in a round-robin fashion to all the networked computers. All network communication is buffered, in order to reduce overhead of small messages. There is a separate 32MB buffer on the client for each networked computer.

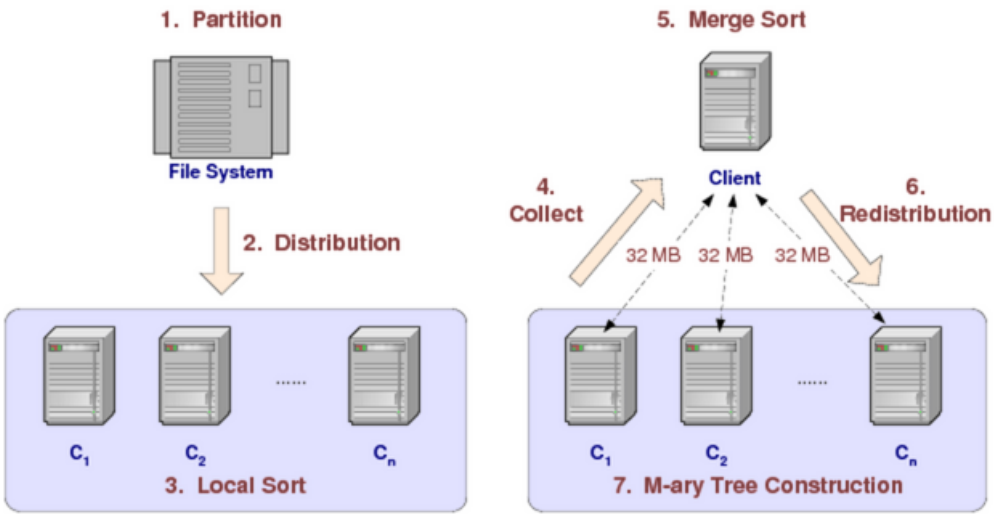


Figure 2.1: In-situ Preprocessing

Operations 4, 5 and 6 take place concurrently on all computers. After those three operations complete, each networked computer will have received an equal share of the entire dataset. (7) Each of these computers will then act as a data server. The data items are then organized locally on each server using a our special search data structure (detailed in Section 2.2.3). No additional data movement is necessary.

2.2.3 Server-side Data Structure

Design Goals

After each data server has received its portion of the data, an efficient data structure is needed to maintain those data items. A selection request, or data culling (the inverse of selection), is formally specified by

$$\{\forall x | lb(i) \leq x(i) \leq ub(i), i \in [0, k-1] \subset N\}$$

Here lb and ub are the lower and upper bounds for each attribute, respectively. k is the number of attributes in each data item. We would like to support scenarios where $k \geq 5$, using a space-efficient data structure while offering fast runtime performance. In addition, we would like the performance to be dependent on the number of relevant voxels, but independent of the size of the entire dataset. This property would be greatly advantageous since visualization algorithms usually focus on iso-ranges or intervals based on different variables. Such is the case with isosurface extraction and volume rendering.

Design of Data Structure

Our primary design concept is to use an M-ary balanced search tree of depth N. A practical combination could be $M = 256$ and $N = 3$; in all situations, M should always be significantly larger than N. This is one of the primary differences between this search

data structure and previous data structures, such as interval tree, k-d tree, quadtree and octree.

The M-ary search tree (Figure 2.2) is an adaptation of a classic concept known as B+ tree, adapted for use in multivariate visualization. In the tree, each tree node contains a list of keys and pointers to its children. From the list of keys stored in a tree node, the child containing a given target search key can be quickly identified. The M-ary tree provides meta data to guide a search process. Due to the large branching factor, M, the M-ary search tree requires little storage space. Actual data items are not stored in the tree, but in a linear list sorted by a key function, described in Section 2.2.3. The leaf nodes of the tree only store pointers to the respective records in a sorted linear list.

Such search trees have been used in other disciplines. The most noteworthy example is B trees used frequently in database systems. Our use of the M-ary search trees has several unique aspects [Weiss, 1998]. First, most disciplines use search trees in situations where records are dynamically inserted and deleted, requiring complex operations to keep the entire search tree balanced. The overhead of operations such as tree node adoption, rotation, etc., are fairly significant. Fortunately, with multivariate visualization, runtime insertion and deletion are not often necessary. Therefore, our algorithms for tree construction and maintenance are simpler and handle gigabyte sized volume data more efficiently. In particular, our search tree is always a complete M-ary search tree of depth N. Second, we have discovered that conventional methods for record access by traversing the tree are too expensive, both in terms of the large number of addressing operations and caching performance. To address this, we use a novel method to accelerate range searches in an M-ary tree for assistance (detailed in Section 2.2.4), optimized specifically for multivariate datasets. Third, our search tree does not index each individual item of the data, but rather points into the sorted linear list of data items. Thus, each leaf node indexes on the order of 10 to 1000 data items.

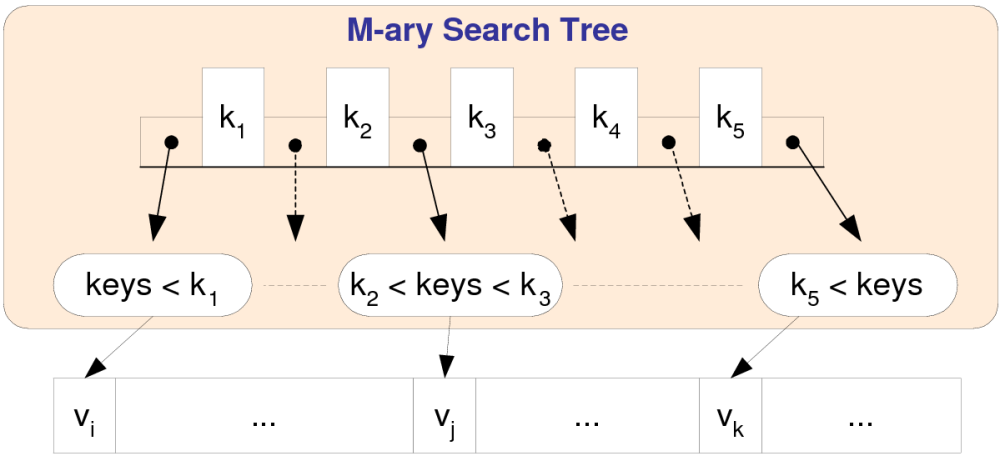


Figure 2.2: An illustrated M-ary search tree.

Tree Construction

While general methods for mapping arbitrary multi-field records to a single numeric key do not exist, it is quite straightforward when dealing with multivariate visualization data that has been quantized. We avoid using scalar valued hash functions to obtain keys; rather, we directly use attributes of each data item as the "key function". This allows a much larger range of possible keys than using only single scalar values as the key. To determine the order between two compound keys, attribute a_0 is the most significant, while a_{k-1} is the least significant.

For any given dataset, the tree is a static data structure that can be pre-computed in a one-time preprocessing step. After each data server has received all data items, we compute the key for each voxel and sort the data items into an array based on the key. On a typical 2.8 GHz P4 processor, quicksort, as part of the standard C library can finish sorting 32 million voxels within 40 seconds. This overhead as a one-time preprocessing cost is quite reasonable.

From the sorted list, we can efficiently construct the tree. Given a combination of M and N values, the total number, T_n, of leaf nodes is equal to M^N. In the case of $M = 256$, $N = 3$, T_n is 16 million. Then, the average number of voxels to be stored in each leaf node, to ensure a balanced tree, can be conveniently obtained as $1/T_n$ of the whole set of data items.

In Figure 2.2, a tree of $M = 6$, $N = 1$ is illustrated, with the sorted array of data items shown at the bottom. The array is partitioned into M^N consecutive runs. The data item separating two neighboring runs is then used as the key in the corresponding leaf node. For instance, the run started by element v_j should have elements with keys valued between k_2 and k_3. v_j should be the data item with the smallest key larger than k_2, while v_k has the smallest key larger than k_5. In a bottom-up fashion, we sequentially build the leaf nodes in left to right order and recursively fill in key values in intermediate nodes based on the range of key values within each child node.

Our experiments on the tradeoff between search performance and storage overhead

have shown that keeping the number of data items pointed to by a leaf node on the order of 10 to 1000 is close to optimal.

2.2.4 Query Handling and Complexity of Query

Server Side Query Handling

Traditionally, one accesses all entries in a search tree by first referencing the root node. While this might be sufficient for querying single items, we have found this conventional approach to be quite expensive for querying many relevant voxels from a large dataset at the same time. A large portion of the incurred overhead is due to addressing operations and cache misses during tree traversal.

To achieve maximal performance, we designed a special scheme to search through the M-ary tree. At the core of the entire process, we use a multi-digit counter with a carrying mechanism. For instance, we illustrate k attributes in (Figure 2.3). For each attribute a range is specified, shown as the non-empty region in each column. The bottom of Figure 2.3 is the same array of sorted data items in Figure 2.2. In the array, the leftmost data item has the lowest key value, and the values of compound keys monotonically increase from left to right.

In the attribute space, the query is specified by the lower bound, lb_i, and upper bound ub_i. From the first (leftmost) element of the array, we search for data items meeting the compound query. Specifically, we start by searching for the lower bound lb_i in the M-ary tree. The first set of relevant voxels should correspond to a consecutive run, e.g. run A in Figure 2.3. We traverse run A until we encounter a voxel with its compound key a_i falling out of the specified range. To determine which data item to search next, we use the method described in Figure 2.4 to increase the key a_i "back into" the specified range via a "carrying" mechanism.

We note here that the resulting key, a_i, may or may not exist in the data. But we search for it in the M-ary tree regardless. If the key a_i does exist, the M-ary tree will

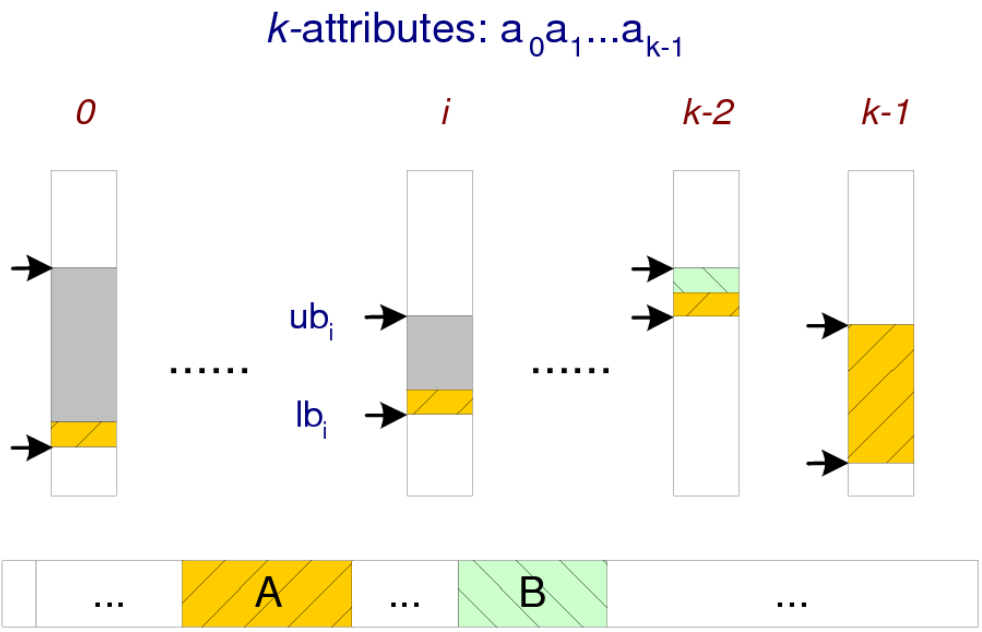


Figure 2.3: An illustration of server-side query handling using an M-ary search tree.

```
/* a[i] is the compound key */
for (i = k-1, carry = 0; i >= 0; i --)  {
     a[i] += carry;
     carry = 0;

     if (a[i] < lb[i]) {
        a[i] = lb[i];
        for (j = i; j <= k; j ++)
            a[j] = lb[j];
     }

     if (a[i] > ub[i]) {
         a[i] = lb[i];
         carry = 1;
     }
}
```

Figure 2.4: Pseudo code describing how an out-of-bound compound key is "carried" back into range.

identify its location in the array, and provide the starting point of the next consecutive run of relevant voxels, e.g. run B in Figure 2.3, meeting the boolean query. If not, the M-ary tree points to the next immediate data item with a compound key larger than a_i. We then repeat the above process to know whether or not this data item is in the bound. The entire process is iterative until the whole dataset has been searched.

In essence, our search mechanism performs a linear search on the sorted voxel list, and uses the M-ary tree only to skip voxels that are out of the specified range. While traversing a linear list in contiguous blocks is efficient, the M-ary tree also provides great cache coherence, because of its compact size and shallow tree-depth. In contrast, traditional tree traversal may incur a much higher overhead. Without knowing a priori how many data items would exist in a range, or whether such data items exist at all, one would need to make an overwhelming $\prod_{i=0}^{k-1} r_i$ number of individual queries, where r_i is the size of the range of a_i.

As a side note, the pseudo code described in Figure 2.4 can be adapted to handle floating point data. All that is needed is to set the "carry" variable to the smallest unit of floating point in the dataset. Finally, if a user is solely querying the last attribute, a_{k-1}, those runs shown in Figure 2.3 could contain as few as just one voxel. This will cause each search in the M-ary tree to generate just one data item, and thus reduce efficiency. Fortunately, by leveraging user's domain knowledge and putting attributes queried less often as the last few attributes in our system, efficiency of the system can generally be maintained.

Query Handling on Client Side

A user query consists of compound boolean range queries entered via an interactive user interface on the client side. The client then simultaneously sends request to all data servers which send the query results back to the client upon data retrieval. During the data retrieval process on the server side, the data items are further reduced in size by simply sending only data parts necessary to render the current query on the client.

For example, if the client's transfer function only uses a single value from the data item to determine color and opacity of a voxel, then only that will be sent, along with the inevitable spatial coordinates and first degree gradients. This reduces the data to be sent over the network to about 10 bytes per voxel. After the client has gathered all incoming data, it can then use any volume visualization approach, e.g. splatting, to display the result to the user. Additionally, queries will be cached locally at the client, so that recurrent requests can be dealt with locally by obtaining the data from the cache.

2.2.5 The Communication Interface

The server/ client interface has been designed to keep communication overhead low by using several different protocols depending on what information needs to be communicated.

In our system, a server opens a TCP port and starts listening for a client to connect to it. Upon establishing a connection, the first few bytes sent by the client indicate what kind of task it needs the server to perform. The server refers to its look-up table of functions to find the appropriate component and hands over control to it. From that moment on, the client communicates with the appropriate component directly. Different components implement different protocols where needed. For instance, the "QUERY" component, responsible for retrieving the data matching a client's query, sends back most of its data as a plain memory copy of what it finds in the server's data storage. There is no need for additional tags or other overhead as the client agreed upon the selected protocol by calling that particular server function.

This interface design has another advantage. As the actual core of the server simply decides what component needs to be activated, it can fork off a process to run the component and close the socket right away. This enables the server to handle multiple connections simultaneously, e.g. for the use of multi-display devices, where more than one client needs to connect to a server.

2.3 System Deployment and Results

2.3.1 Deployment / Requirements

Our system is relatively easy to deploy on networked commodity computers, whether part of a cluster or not. One computer will function as a rendering client, and the rest as data servers. The necessary number of parallel data servers depends on the size of the dataset. As long as more computers are available to serve as data servers, our system could be scaled to handle larger and larger datasets.

However, we do have a couple minimal requirements if interactive visualization is desired. First, all data servers combined must have enough main memory to hold the entire preprocessed data in-core. Our system is still operable without meeting this requirement, however, the additional out-of-core overhead incurred by each data server would hamper the query performance. Second, the client machine should be able to hold the queried data in-core locally. Our client software also includes a cache to avoiding retransmission of recurrent requests. Even with the cache resident in memory on the rendering client, most queries in our experiment still did not exceed the limit of local memory.

In all experiments, our data servers and the rendering client are commodity computers of the same configuration (Dual AMD Opteron 1.6 GHz, 2GB RAM), connected by a Quadrics Elan3 network offering a theoretical single-direction bandwidth of 400MB per second. No additional special hardware or software was used.

2.3.2 Datasets and Pre-Processing

Our system has made it feasible to examine more variables interactively within the same framework than previously possible. For experimental evaluation, we use two datasets: the TSI dataset and the RMI dataset.

The TSI dataset is the main testing dataset, since it is inherently multivariate, with time-varying density, pressure, and flow velocity simulated on each voxel. The spatial

resolution is $864 \times 864 \times 864$. In astrophysics research, density and pressure are often converted to a third variable, entropy. Besides commonly-used attributes such as gradients of various degrees, application scientists are also interested in tangential velocity and angular momentum. The angular momentum with respect to a chosen origin for a given vector field (denoting velocity and direction of velocity) is given by $L = v \times r$, at every point where r is denoting the vector from the chosen origin to the current point, and v is the vector from the velocity field.

In our experiments, there are 11 attributes visualized for the TSI dataset: time step id, entropy, magnitudes of tangential velocity, angular momentum gradient, entropy gradient, first derivative of entropy in x-, y-, and z-directions, and x, y and z coordinates. This is also the order (significance) of the attributes, when the server side M-ary tree is constructed, with entropy being the most significant. The spatial and temporal coordinates are needed, because application scientists routinely do clipped or cut-away views of their data. Our tests used six time steps of the TSI dataset totaling 105GB of data.

The most expensive overhead incurred during data preprocessing (2.2.2) is not to construct the search data structure, but actually to compute and coalesce the attributes of a volume partition into one common file, with each voxel stored in 22 contiguous bytes (11 attributes). This step is a one-time process which takes 5 to 8 hours in our experiments, since some computation needs to take place out-of-core. Nonetheless, this step could be significantly accelerated with more sophisticated parallel computing software, which is out of scope for our research.

The one-time process that distributes the whole dataset onto multiple, e.g. 20, parallel servers is the sort in Hilbert curve order and the construction of the M-ary tree (Figure 2.1), which only takes about 2 hours.

For scalability experiments, we have also used 3 time steps of the larger RMI simulation data. However, since the RMI data only contains a single variable, it is not the ideal dataset for multivariate visualization. We use this data strictly for scalability evaluation. The version of RMI data we used is of $1024 \times 1024 \times 960$ spatial resolution. Again,

we included 11 attributes in the following order: time step id, density (the original variable in the dataset), correlation between directly neighboring voxels in the +z direction, correlation between voxels that are one voxel apart in the +z direction, norm of density gradient, first derivative of density in x-, y-, and z-directions, and x, y and z coordinates.

2.3.3 Timing Results

After preprocessing, including binning and discarding uninteresting voxels such as empty space, we end up with about 20GB of data items with the TSI dataset. We experimented with the TSI dataset using configurations of 20, 30 and 40 data servers, respectively. With the 20-server configuration, each server holds roughly 980MB of data in memory. In 30- and 40-server configurations, that in-core amount of data decreases to 653MB and 490MB, accordingly.

The RMI dataset is larger and also has a much smaller portion of data constituting empty space. After preprocessing, each of 40 servers still holds about 1.6GB of data. For that reason, we did not have scalability results of the RMI data using 20- or 30-server configuration.

Running queries locally on a single data server, we are able to retrieve between 4 and 13 million data items per second. The variance is caused by the significance levels of the attributes when the M-ary tree is constructed. If a user primarily queries on the least significant (the last) attribute, 4 million data items is the maximal rate. When the most significant attributes are targeted, the rate of data retrieval increases to 13 million data items per second.

We have found in the parallel system, however, that the network is the bottleneck regardless of which attributes are being queried (see Figure 2.5). Due to the balanced workload among servers (shown in Table 2.1), the total local querying capability amounts to $80 \sim 260$, $120 \sim 390$, and $160 \sim 520$ million data items per second, with 20, 30 and 40 data servers operating in parallel, respectively. Generally, the local query time accounts

for only a fraction of the total latency, even if a server has to search through more than 1.5GB of data to find voxels matching the query.

When a query only results in a small number of relevant voxels, smaller messages are sent across the network, thus a generally lower level of performance is obtained. As the size of queries increases, the obtained query rate approaches 9 million data items per second, which corresponds to a 144 MB/sec sustained bandwidth on the network (only 16 of the 22 bytes are transmitted per data item, see 2.2.4), which is considerably less than the local query rate. Thus, the network transfer time accounts for most of the total latency.

In a separate experiment, we have run the same test using a standard Gigabit Ethernet as the interconnect. There, the obtained maximal query rate in our system is 4.2 million data items per second (roughly a 536Mbits per second sustained bandwidth).

From Figure 2.5, we see that the system's performance is rather stable even with differing numbers of servers. We attribute the relatively lower throughput when using 40 servers to the fact that more concurrent TCP streams are on the network, with smaller messages being sent. While the RMI dataset has almost double the amount of data for the system to handle, the achieved throughput was not reduced significantly.

In Table 2.1, we recorded the exact number of data items found to be relevant for queries resulting in varying number of relevant voxels. Although perfect load-balancing between data servers cannot be theoretically proven, we find the measured load imbalance negligible. The load imbalance ranged between a mere 0.012% to 0.155% for medium and large queries. For very small queries $(10-10,000)$ the load imbalance increased to a reasonable 4%, since load-balancing tends to be more important for large scale queries. When more data servers are used, e.g. 30 or 40, the relative levels of load imbalance is similar to those shown in Table 2.1.

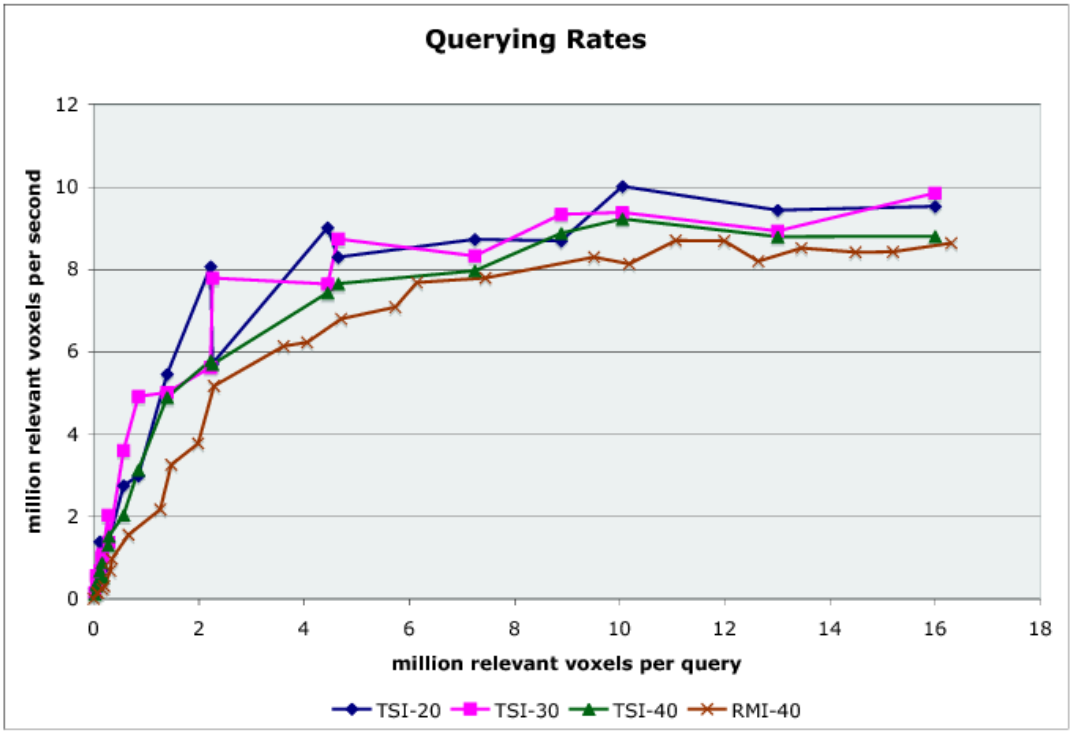


Figure 2.5: Measured rates of query using queries of different numbers of relevant voxels.

Table 2.1: Load imbalance for different queries from TSI data (6 time steps, $864 \times 864 \times 864$, 11 attributes per voxel) in a configuration with 20 data servers.

voxels per query	voxels per server	load imbalance abs.	load imbalance relative
228,838	$11,464 \pm 178$	356	0.155%
1,335,685	$66,968 \pm 533$	1,066	0.079%
6,099,624	$304,957 \pm 825$	1,650	0.027%
7,737,111	$386,964 \pm 863$	1,726	0.022%
9,688,584	$484,541 \pm 936$	1,872	0.019%
10,989,382	$549,588 \pm 1,100$	2,200	0.020%
12,837,754	$642,000 \pm 1,361$	2,722	0.021%
17,700,906	$885,162 \pm 1,711$	3,422	0.019%
19,241,715	$962,156 \pm 1,713$	3,426	0.017%
20,304,821	$1,015,461 \pm 1,549$	3,098	0.015%
22,330,131	$1,116,666 \pm 1,614$	3,228	0.014%
23,565,859	$1,178,372 \pm 1,804$	3,608	0.015%
24,467,368	$1,223,571 \pm 1,416$	2,832	0.011%
28,021,080	$1,401,034 \pm 1,749$	3,498	0.012%
30,085,660	$1,504,109 \pm 1,932$	3,864	0.012%

Chapter 3

Visualizing Temporal Patterns using Textual Pattern Matching

Extracting and visualizing temporal patterns in large scientific data is an open problem in visualization research. First, there are few proven methods to flexibly and concisely define general temporal patterns for visualization. Second, with large time-dependent data sets, as typical with today's large-scale simulations, scalable and general solutions for handling the data are still not widely available. We have developed a textual pattern matching approach for specifying and identifying general temporal patterns. Besides defining the formalism of the language, we also provide a working implementation with sufficient efficiency and scalability to handle large data sets. Using recent large-scale simulation data from multiple application domains, we demonstrate that our visualization approach is one of the first to empower a concept driven exploration of large-scale time-varying multivariate data.

3.1 Introduction

Recent advances in the field of query-based visualization have enabled a powerful technique for large data visualization. Using query-based visualization, users can quickly

and intuitively explore very large, highly complex multivariate data. However, before query-based visualization can adequately address the needs of exploratory computational science research, several important challenges must be resolved.

First, although human-domain knowledge is always a driving factor, query-based visualization is commonly guided by trial and error and is often ad hoc. In addition, the approach becomes extremely time consuming when the exponentially large growth of multivariate time-varying simulation data is considered. In this domain, more efficient and versatile user tools are necessary.

Second, data set sizes and the unstructured data access patterns required in an exploratory usage mode put a strain on today's visualization architectures. Sufficient parallel visualization infrastructure must be developed in a generic and yet optimized manner.

Third, qualitative concepts involve an innate degree of uncertainty. Visualization methods must be developed to adequately and properly display the uncertainty inherent in the understanding of these scientific phenomena.

In this work, we use general time-varying multivariate simulation data as the driving applications to investigate ways to address these bottlenecks. Specifically, we aim to discover temporal connections between multivariate patterns of concern. Our test data sets come from two simulations: global climate modeling research that aims at understanding long-term climate change and combustion research that aims at developing breakthroughs to advance fuel efficiency.

Motivated by the elegance and power of regular expressions and globbing in text string searching, we have developed a text search language using *flex* and *bison* for concisely specifying temporal patterns. We borrow heavily from the richness and ease of regular expressions and globbing. Temporal patterns are sequential by nature and can include an arbitrary number of variables in their specification.

In our system, a user hypothesis needs to be only partially defined, similar to how `glob()` is used in UNIX. `glob()` is a classic tool that expands searches for file names

according to partially defined patterns (e.g. `file??.pdf`). Regular expressions allow for partial definitions as well (e.g. `file.*\.pdf`). Our system accepts a kind of qualitative query that is similar to both globbing and regular expressions. In these queries, wildcards provide a powerful way to represent the existence of non-topical events, whether temporal or multivariate. Queries containing wildcards are then expanded to a number of actual range queries that can be extracted from the data set.

We find this type of querying capability to be very powerful for visualizing temporal patterns in large simulation data, because it elegantly allows the specification of uncertainty in the time separation within features. Due to the allowed uncertainty and the ensuing overwhelming complexity, a sufficient way to automate the overall query-driven visualization is necessary. By using a formal language approach, we blend rigor with expressiveness and flexibility in query-driven visualization. Queries containing wildcards are then expanded to a number of actual range queries that can be extracted from the dataset.

As common with all formal language approaches, we can make incisive distinctions in temporal data in an unconstrained manner. The more traditional visual subsetting approaches, such as brushing in scatterplots and selecting contiguous ranges in parallel coordinate plots, are naturally intuitive but are often manual and hard to automate. Especially in the case of time-varying multivariate data, manual data selection can be cumbersome and in many cases infeasible.

To application scientists, this method of specifying temporal patterns to visualize is extremely useful. Often domain knowledge is expressed in a qualitative manner, and domain scientists can be hard pressed to define discrete data queries to extract meaningful subsets of the data. Our approach supports hypotheses that may only be partially defined. Qualitative patterns containing wildcard characters can be entered and are expanded to a set of discrete data queries that are then extracted from the data set. For a scientist, this offers a more natural way of entering qualitative domain knowledge and avoids a potentially lengthy search process guided by trial and error. In addition, we

provide ways to compare and relate results from multiple queries in search of broader multivariate patterns.

3.2 Overview

Our system provides analytic tools to aid scientists in understanding inter-variable relationships and their development over time.

For instance, in combustion research, scientists are often interested in "precursor" events preceding an event of reignition. We enable the scientist to describe these precursor events as well as the reignition event by using wildcards for the values of multiple variables over time. This allows for a single event description to match several actual events of the same nature that are not identical. For instance, a reignition process as described by the scientist could last only a few time steps or continue to go on for many subsequent time steps, with slightly different values of the given variables. Occurrences of precursor events could then lead to discovery of time-lagged correlations with high statistical reliability.

In the following sections we will describe our data organization, infrastructure, pattern matching language, and data retrieval in detail.

3.2.1 Data Organization

In order to query directly for temporal changes in the data set, we precompute the relative change for each variable over time and store it along with the original data value. The pattern matching language described in section 3.2.2 enables us to query for data locations matching a user-defined data value or range and a user-defined relative temporal change over time. Multivariate queries specifying any number of data values are supported directly.

Our approach uses an item-based data format in which each item holds a complete set of values, including its three coordinates, time index, and values of all its variables.

A data item can be uniquely identified by its spatial coordinates and time index when using a time-varying data set. We do not require specific connectivity requirements, and are thus freed from mesh restrictions. However, the requirement of explicit coordinate fields can increase the size of data sets where coordinate locations are natively implicit. For example, a structured three-dimensional time-varying data set containing a single variable will be increased by a factor of 5. This penalty decreases dramatically as more variables are included, as is frequently the case with typical scientific data sets. A typical two-dimensional climate data set may contain 70 or more variables, for instance, bringing the increase in data size down to less than 5%.

Further optimizations counterbalance the increase of data set size. First, since our data is no longer grid based, we do not necessarily need to store it entirely, but can select parts of it based solely on item values. We thus allow sparse data sets to be represented very efficiently. As many scientific simulation data sets only use a part of the entire grid, we typically find many elements at the outer boundary of the grid that contain fill values only and no valuable data, and we simply discard these data items. In our experience, this optimization alone often counterbalances the increase in size with many scientific data sets.

The second optimization we employ is compression. We extended the approach described in chapter 2 by additionally compressing data items in memory on each data server. Blocks of data items are decompressed when an item inside a data block is accessed. An LRU cache of decompressed data blocks accelerates subsequent reads to the same block. The compression rates vary from data set to data set. For typical data sets, we saw compression ratios of 20:1, while on highly noisy data sets, we could obtain as low as 4:1 compression.

The way we arrange the values of individual variables in the data items is directly related to the kind of queries we can make. We allow querying for temporal patterns whose location in time and discrete data values are not known a priori. To do so, we store in each data item a sequence of values describing the relative change for each

variable from one time step to the next along with its coordinates. To this end, spatial coordinates are the only identifiers of each voxel. On every voxel, all temporal changes and inter-variable differences are treated equally as parts of the multivariate data.

In addition to the relative change for each item and timestep, we also store the original data values in a separate set of data items. This is required to give our temporal "deltas" a base or absolute reference from which to work. Being able to query both sets of data items – relative change and absolute data values – and then combine the results allows for more precise and expressive queries.

3.2.2 Pattern Matching and Syntax

For querying, we employ range queries, a widely used method for selecting subsets of multivariate data. Even though range queries are usually discrete, our system accepts quantitative queries that match and support a user's unclear or imprecise understanding of data. Thus, we allow a user to issue "fuzzy queries" in order to explore a data set he is not highly familiar with. Range queries may contain wildcard characters such as `*` and `?` that are expanded to generate actual range queries, much like the UNIX function `glob()` expands wildcards in `file??.pdf` and regular expressions expand patterns such as `file.*\.pdf`.

However, our language has a few non-traditional elements. The first one is `T`, the temporal mark. A search of `[4]*T[5]*` means that we are looking for patterns where an attribute is valued at 4 for zero or more timesteps and then changes to value 5. We use braces around all ranges to avoid ambigious cases such as 45* (meaning either [45]* or [4][5]*) The temporal mark is the time of this event's occurrence, and in this case it is chosen to be the instant of the first timestep of the value 5. Our parser extracts the `T` from the expression and then generates all discrete patterns of interest. The location of `T` is recorded so as to indicate the precise time of the event's occurrence in further visualizations or data explorations. Multiple `T`'s are ignored. Only the first `T` in a query is chosen as the temporal mark.

The second extension in our system is to compute a matching score for results returned by each of the alternative searches. When a user specifies `[4]*[5]*` as the pattern, a user may also specify an ideal template to match with. For instance, a user may be specifically interested in `[4][4][4][5][5]`. A distance is computed between `[4][4][4][5][5]` and each case matching `[4]*[5]*`. This technique enables a user to specify a controllable level of uncertainty and yet can still focus on a very specific target. We use a standard normalized euclidean distance for the data sets used herein. A different distance metric may be supplied by the user, and as such, a metric heavily depends on the data values and their relationship to each other.

We have implemented our pattern matching language using *flex*[1] and *bison*[2]. The main elements of our query language can be found in Figures 3.1 and 3.2.

The following list of examples demonstrates accepted wildcards, special characters, and valid data ranges:

- `[-1e15 - 1025.7]` – data ranges for a single timestep. This range contains a from-value and a to-value. `[74.2]` has both values being identical (same as `[74.2 - 74.2]`).
- `[0.3 - 10e9]*` – a data range applied to zero or more sequential timesteps.
- `?`, `?*` – wildcard ranges. The first represents the entire range of data values for a single timestep. The second represents the entire range of data values for zero or more timesteps.
- `T` – the (optional) temporal mark. It marks the time index at which the event that is subject of the query occurs.
- `[1 - 5]*[8]` – a query without a temporal mark is also valid. This query adresses all items which values in all timesteps are between 1 and 5, and a value of 8 in the last timestep.

[1] http://flex.sourceforge.net/
[2] http://www.gnu.org/software/bison/

```
/* floating point numbers */
\-?[0-9]+
| \-?"."([0-9])+
| \-?([0-9])+"."([0-9])*
| \-?([0-9])+\.?([0-9])*[eE]\-?([0-9])+
      return VALUE;

\[    return RBEGIN;
\]    return REND;
\?    return RWILDCARD;
\*    return MREXPANDER;
\-    return RTO;
T     return TIMEMARKER;

[ \t\n]   ;

.         ;
```

Figure 3.1: Tokens defined for lexical analyzer (*flex*)

```
statement:
       | statement range
       | statement multirange
       | statement TIMEMARKER range
       | statement TIMEMARKER multirange
       ;

range: RBEGIN values REND
   | RWILDCARD
   ;

values: VALUE RTO VALUE
        | VALUE
        | RWILDCARD
        ;

multirange: range MREXPANDER
        ;
```

Figure 3.2: Grammar rules (*bison* grammar file)

Data ranges that are appended with a `*` are called multiranges, as their values can span over multiple timesteps. All other ranges, including `?`, are ordinary ranges.

3.2.3 Matching and Querying

In order to query for data elements matching range queries as defined in section 3.2.2, the queries are expanded to discrete queries. As we know the number of timesteps in the data set a priori, we must expand a user query to define a range for each timestep in order to construct a complete range query.

For an example data set with 5 timesteps, a query `[2][3]*[6]T[4]*[5]` expands to 3 discrete range queries `[2][6]T[4][4][5]`, `[2][3][6]T[4][5]`, and `[2][3][3][6]T[5]`.

Expansion of general range queries to construct discrete range queries may explode the number of actual queries being issued to the data servers. The number of discrete range queries after expansion can be determined as follows. Let t be the number of timesteps in the data set, let m be the number of ordinary data ranges, and let n be the number of multiranges. In our above example, t equals 5, there are $m = 3$ ordinary ranges (`[2]`, `[6]`, and `[5]`), and $n = 2$ multiranges (`[3]*` and `[4]*`). It is clear that the 2 multiranges need to fill up the remaining $k = t - m = 5 - 3 = 2$ data ranges to construct a complete set of 5 data ranges for a discrete range query. Mathematically, this is a multiset of cardinality k, with elements taken from a set of cardinality n. The corresponding multiset coefficient and therefore the number of discrete range queries after expansion is defined as

$$\left\langle {n \atop k} \right\rangle = \binom{n+k-1}{n-1} = \binom{n+k-1}{k}. \tag{3.1}$$

This number can potentially get very large, even for low n and k. Our parallel data servers are able to handle a large number of consecutive queries efficiently, and only data items matching one or more queries in the set are retrieved. Each data item is retrieved only once. The position of the temporal mark `T` for each query is added onto every data item. It marks the timestep in which the event that was subject of the query occurs. If a

data item matches more than one of the discrete queries, it contains the data values for the earliest temporal mark at which it occurred (i.e. we are marking only the first occurrence of the user-defined event in the time sequence). In the example, `T` marks the first timestep with a value of 4 directly following a value of 6, and the temporal mark would be at timestep 2, 3, and 4 for the three given discrete queries. All items coming from one discrete query are given the same time index (e.g. an item returned by the second discrete query of the example gives its spatial location a time index of 3).

For multivariate queries, the query string for each variable may be specified separately, and the results are combined upon data retrieval for data items matching all (logical AND) of the multivariate queries at each individual spatial location. This ensures that, upon combining queries, all data items have a full set of data values from the query. Temporal marks in multiple queries can either be included in the intersection (like the spatial location), or a temporal mark from one of the queries can be selected as the temporal mark of the intersecting data items.

3.3 Results

We evaluate our newly developed method using two simulation data sets from different domains. The first data set consists of 71 variables output from a global land surface model running within a general circulation model performing a projection of Earth's climate from the year 2000 to 2099. We are using monthly averaged data with 12 timesteps per year, comprising 1200 timesteps in total on a grid of size 256×128. Alongside each variable we store the relative temporal change between consecutive months. This leads to a total data set size of 21 gigabytes.

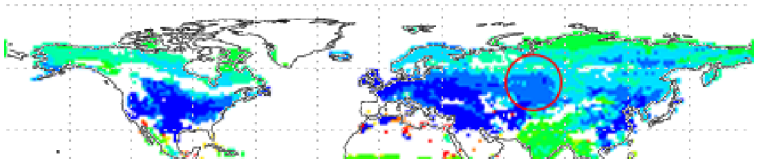
The second data set comes from combustion simulation, has a grid size of $480 \times 720 \times 120$ and 5 variables over 122 timesteps. Once again, we also store the relative temporal change between consecutive timesteps together with the original data values for each variable, resulting in a data set of 188 gigabytes.

3.3.1 Global Climate Modeling Simulation

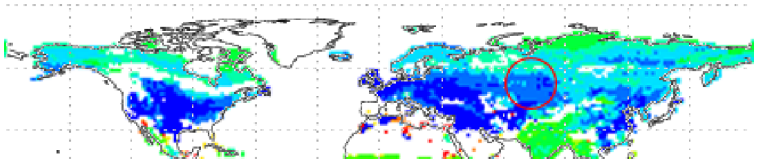
Using the global climate modeling data set, we establish "events" that are defined by one or more variables changing over time at different spatial locations. We mark the time of such events at each spatial location with the temporal mark `T`, and color individual pixels accordingly (see Figure 3.5). As we are considering temporal change between consecutive months, colors indicate points in time that are between the months in which the event occurred (i.e. a blue pixel depicts that the event happened in between the January average and the February average). Even though the low temporal resolution of the data is a disadvantage when trying to search for the exact point in time an event occurred, we were able to extract several events of significance.

In Figures 3.3(a) through 3.3(c), we display the temporal change of the variable ELAI (Exposed one-sided leaf area index in units of m^2/m^2) over the course of one year. We queried for the following percentage change pattern: `[-.4-.4] *T[.4-max]?*`, meaning that we are looking for data locations that experience a positive relative change of more than 40% after going through a period of zero or more timesteps where the relative change was low (between -40% and $+40\%$). The purpose of this query is to find the beginning of the growing season in the Northern Hemisphere. We mark the first timestep in which the larger change occurs and color the pixels matching this query according to the temporal mark `T`. We chose 40% as the threshold after consulting with a climate scientist as to what rise of ELAI would best represent the event we are looking for as described in the following paragraphs. Please note that the values *min* and *max* in the queries are simply used for ease of readability, where *min* represents the smallest and *max* the largest floating point number the system can represent.

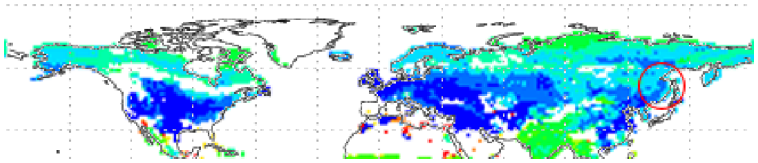
We can immediately see how the event we queried for follows a certain path as the year progresses. It begins in the Southern parts of North and Central America and generally advances North month by month. On the Eurasian landmass, a similar progression to the North is visible, as well as a progresssion from Central Europe towards Central Russia and Siberia.



(a) Year 2030 of ELAI



(b) Year 2050 of ELAI



(c) Year 2090 of ELAI

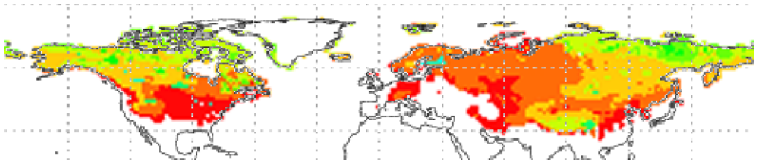
Figure 3.3: Northern hemisphere colored by temporal mark `T` representing the month as labeled in Figure 3.5. Variation between years highlighted as red circles.

What we have established here is the progress of spring or rather the beginning of the growing season ("green-up") in the Northern Hemisphere. The leaf area index – which is close to 0 at the beginning of the year in the Northern Hemisphere as there are virtually no leaves exposed in the winter – rises sharply as vegetation starts to flourish, creating leaves and blossoms. As we expect, this event progresses to the North, except where the beginning of spring is delayed (e.g. because of high altitude). One is clearly able to make out the area surrounding the Himalayas as an area of delayed spring.

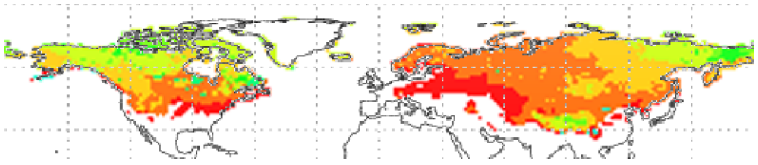
Another interesting feature of this query is the white band connecting the East and West coast in Southern Canada. It appears that spring "skips" over said area. This is due to the fact that this part of Canada – the Northern Boreal Evergreen Forest biome – is home to a different kind of vegetation, dominated by evergreen vegatation that does not significantly change in leaf area index seasonally. The data values of temporal change reveal a rise in the leaf area index in this area, but the index never experiences a change above 40% in a single month. Since our query is explicitly looking for this, we do not report a point in time for these spatial locations.

We also noticed that the differences between years are minor and almost imperceptible. The temporal change of the variable ELAI over the course of each year appears to be very stable.

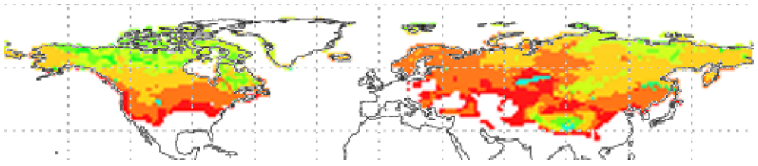
Figure 3.4 displays a query that aims to identify the point in time when the first large snowfall between May and December occurs. The query we are using is `???[`*`min`*`-.07]*T[.07-`*`max`*`]?*` for the change in variable FSNO (Fraction of ground covered by snow). The leading `???` ensures that we query for the full range of values for the relative change between January and April. This way we are sure not to query for the first large snowfall in the first four months of the year, as we want to see how a snow cover progresses over the course of fall and winter in the Northern Hemisphere. The query then considers only data locations whose snow cover is either reduced or increased by no more than 7%. When we encounter the first temporal change of the snow fraction larger than 7%, we consider it the first large snowfall and mark the timestep with the temporal mark `T`.



(a) Year 2050 of FSNO



(b) Year 2051 of FSNO



(c) Year 2052 of FSNO

Figure 3.4: Northern hemisphere colored by temporal mark `T`

We chose 7% as it gives us a good threshold that makes our query resistant to minor changes of the snow fraction which would generate a false and potentially too early temporal mark.

Again, we can clearly make out the underlying pattern. The snow cover first grows larger by more than 7% in Northern Canada and Siberia, as well as the Himalayas, and then progresses into the warmer regions to the South and, in the case of the Eurasian landmass, to the West. One can also recognize the Rocky Mountains on the West Coast as an area of early snowfall. Most of the U.S., however, never experiences a heavy increase of snow cover by more than 7% in one month, just like Southern Europe and Northern Africa. FSNO is more variable than ELAI, as can be seen in the differences between the three years shown.

Figures 3.6 through 3.8 display a different kind of visualization for multivariate queries. We depict two-bit correlations between variables over the course of one or many months. Specifically, two-bit correlation shows 4 combinatorial scenarios: two variables both changing positively, negatively, or in different directions. The term "two-bit" are derived from the fact that we can use two bits to represent each case.

In Figure 3.6 we display variables TSA (air temperature 2 meters above ground) and TV (vegetation temperature). For each variable, we issue 2 queries of the kinds `??[0-max][0-max][0-max]?*` (hereafter referred to as *TSA+* and *TV+*) and `??[min-0][min-0][min-0]?*` (referred to as *TSA-* and *TV-*), meaning that we query for all data items that experience either a constant increase or constant decrease in value between March and June. We then compute the intersection based on spatial location for the pairs *TSA+* and *TV+* (set "++"), *TSA+* and *TV-* (set "+-"), *TSA-* and *TV+* (set "-+"), and *TSA-* and *TV-* (set "--"). As the original sets with prefixes "-" and "+" for each variable are mutually exclusive, all four intersections are likewise mutually exclusive. Thus we are able to combine them into a single image using a simple color scheme (see legends in Figures 3.6 through 3.8). We compute a score based on the normalized euklidean distance of the temporal change of the values TSA, FCEV, and RAIN, to their respective maximum

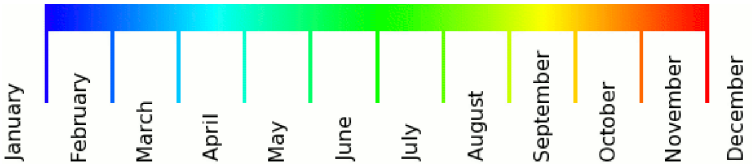


Figure 3.5: Legend for 11 month-to-month changes (Figures 3.3, 3.4, and 3.12)

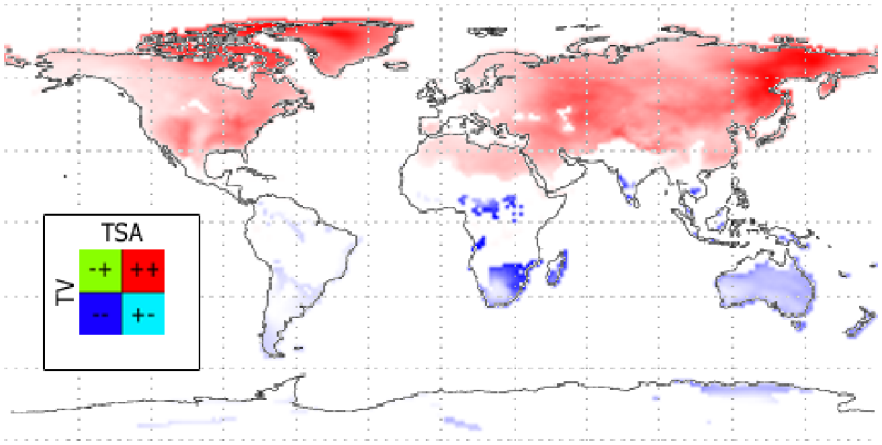


Figure 3.6: Two-bit correlation of TSA and TV

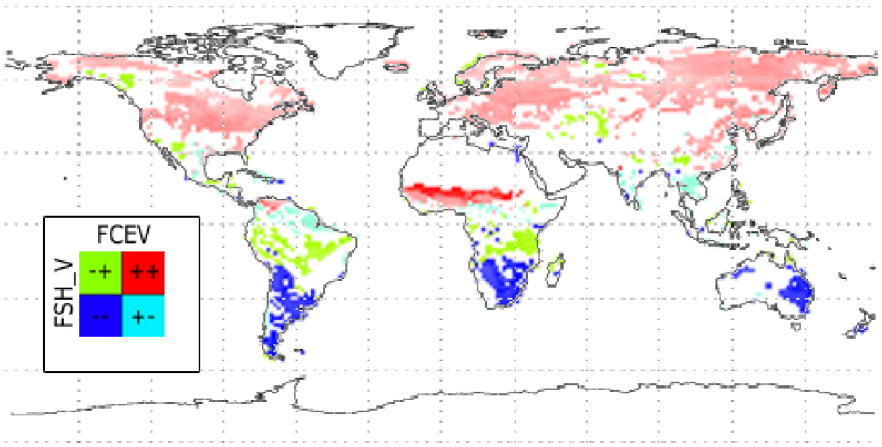


Figure 3.7: Two-bit correlation of FCEV and FSH_V

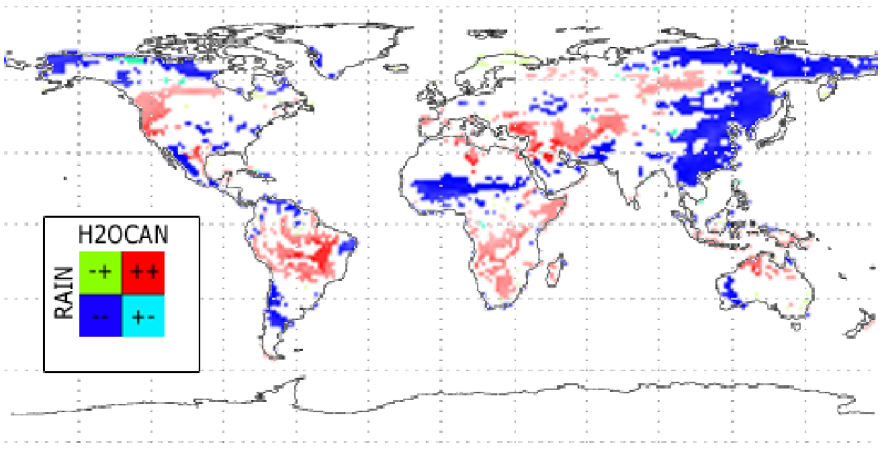


Figure 3.8: Two-bit correlation of H2OCAN and RAIN

change or, in case of the "--" set, to their minimum change. We display the resulting score by opacity. Spatial locations with a higher score are more opaque.

The resulting images can be easily interpreted. Red and blue areas denote areas where both variables' temporal change over one or many months – in the case TSA vs. TV, three months – simultaneously and constantly increased and decreased, respectively. Areas of light green or turquoise signify areas where one variable constantly increases for the duration of the query while the other variable decreased, and vice versa.

Notably, Figure 3.6 does not contain any areas of light green or turquoise color at all. We can conclude that whenever variable TSA constantly rose from March to June, so did TV (red areas); likewise, a constant decrease in value of TSA in said period is always mirrored by TV (blue areas). This signifies that the variables are strongly positively correlated, as one would expect for the air temperature 2 meters above ground and the vegetation temperature.

In Figure 3.7, we correlate variables FCEV (canopy evaporation) and FSH_V (sensible heat from vegetation) for the duration of 3 months (March to May). Clearly, we can identify areas of positive correlation, where both variables increase or decrease simultaneously. Simultaneous increase appears almost exclusively in the Northern Hemisphere, while simultaneous decrease is restricted to areas in the Southern Hemisphere. Smaller areas of negative correlation appear all over the map, but the largest areas can be found in or near tropical rainforests in South America, Africa, and South East Asia. This is counter-intuitive for regions where clouds and active precipitation are dominant.

Figure 3.8 depicts correlation between variables H2OCAN (intercepted water) and RAIN (atmospheric rain) from August to October. It is noticable that, while we retrieve less data points with a constantly increasing or descreasing value in total than with Figures 3.6 and 3.7, positive correlation by far outweighs negative correlation. The negative correlation visible at very high latitudes in Europe is probably caused by the fact that these areas are dominated by grassland, and the concept of "canopy" is not applicable here.

Our approach also allows the correlation of temporal changes in variables at different times and for different durations. While it appears to make more sense to correlate queries of identical event length and event time, our system is not limited to this.

3.3.2 Turbulent Combustion Simulation

In the combustion simulation data set, scientists are interested in finding and tracing areas of interest over time. One example of such an area of interest in nonpremixed combustion is extinction of the flame in areas where fuel and oxidizer are in stoichiometric proportions. Under normal circumstances, the spatial locations with a stoichiometric mixture fraction of 0.42 are a good representation of a fully burning flame. However, local extinction can occur in areas where other chemical effects come into play (e.g. local turbulence). One effects that can cause this behavior is a local value of the mass fraction of the hydroxyl radical (OH) below 0.0007 and a high value of the scalar dissipation rate χ. Areas that maintain these characteristics are temporarily extinct, as they do not support the chemical conditions to maintain a burning flame.

Based on this knowledge, we issue two queries for data items of high χ values (`chi:[18000-33000]*`) and data items of a low reaction of hydroxyl radicals (`Y_OH:*?T[0-.0007]*?*`). We use the temporal mark `T` to mark the timestep at which each variable enters the range favorable for an extinct flame. A third query containing the original values of the stoichiometric mixture fraction (`mixfrac:[.40-.44]*`) retrieves all voxels with a fraction of around 0.42, the perfect mixture fraction for a fully burning flame.

Figures 3.9 through 3.11 display the spatial locations of intersection of the queries colored by the temporal mark. In this visualization of a sparse data volume, we are unable to use lighting based on surface normals. Figure 3.9 is rendered using a typical splatting renderer (without shading). To better convey three-dimensional relationships, Figures 3.10 and 3.11 are generated by rendering a lit sphere for each data item. Additionally, we also rendered the spatial locations of the perfect stoichiometric mixture

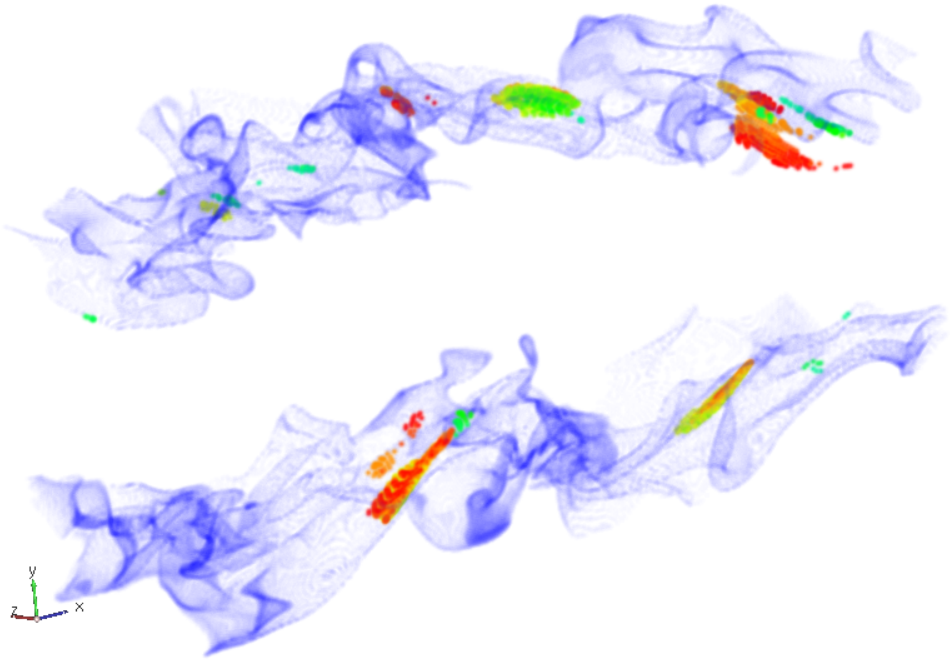


Figure 3.9: Image generated using a splatting renderer (with shading disabled).

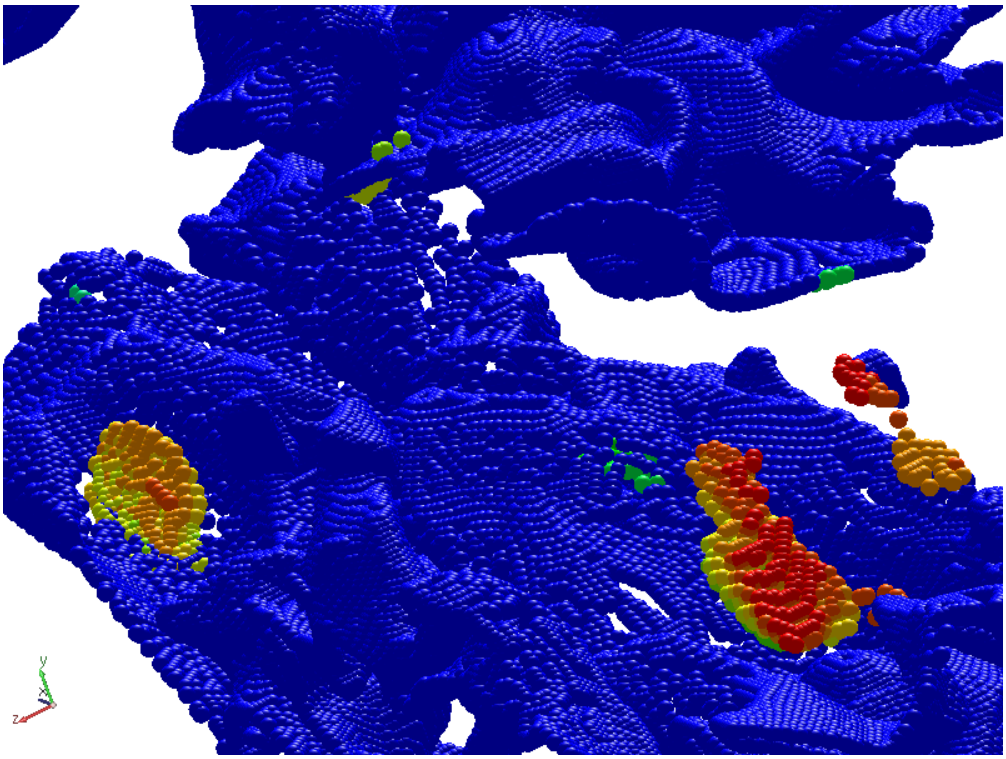


Figure 3.10: A lit sphere rendered for each individual data item better conveys the three-dimensional relationships.

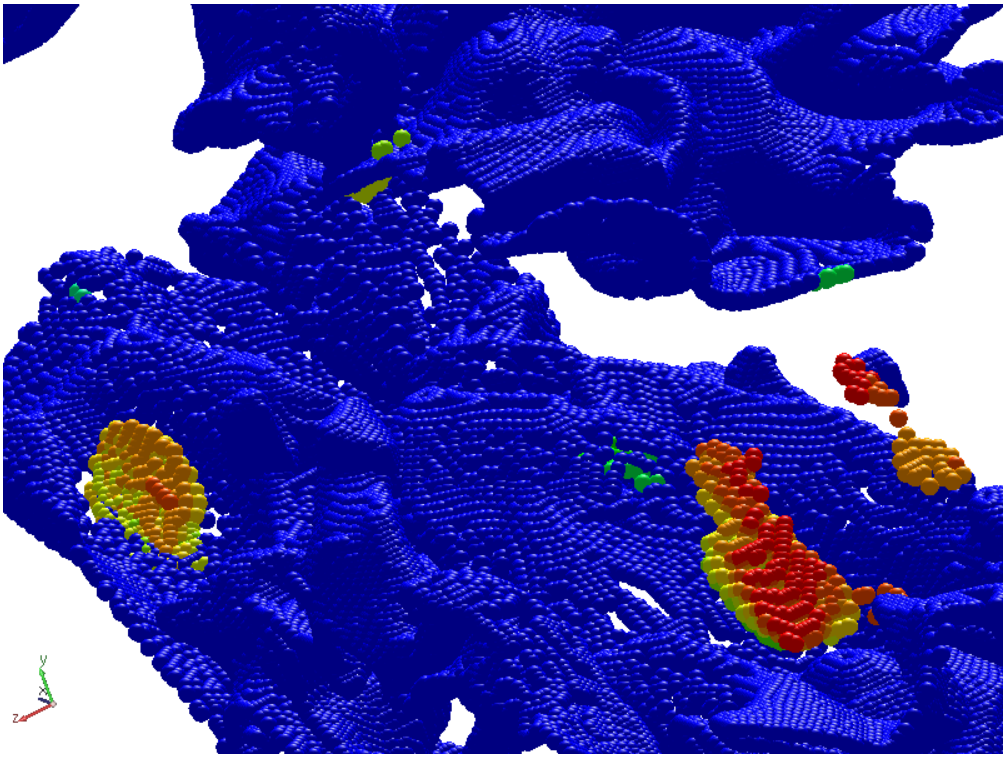


Figure 3.11: Close-up of an extinction region.

fraction (0.42) in blue for clarity.

The spatial location and temporal change of these extinction regions can be analysed in a single 3D visualization. Extinct regions can be found strictly along the outside surface of the blue region of perfect stoichiometric mixture. Also, over time, these regions do not move far away from their origin and generally stay near the blue surface region. We are able to present a single 3D visualization that conveys the formation and movement of locally extinct regions over time.

3.3.3 Performance

All timing results have been collected using an AMD 2.6 GHz Opteron cluster connected by InfiniBand DDR 4X network and the climate data set from section 3.3.1. Measured performance metrics of our system are highly dependent on the data and the query. With most tests that we have run using 20 compute nodes as data servers, the query performance is acceptable for interactive use.

In Table 3.1, we provide a list of qualitatively different queries and show the number of data blocks accessed on each data server node. Due to the great scalability and load balancing using the methods introduced in chapter 2, each server node loads in approximately the same number of blocks during our tests. On each server node, the cache for

Table 3.1: Performance results on 20 compute servers. "Running time" is the wall clock time between query invocation and receipt of all voxels. "# Voxels considered" is the average number of matching voxels returned by each server. "# Queries generated" is the number of specific queries generated by parsing and interpreting the query specified.

ID	Query	# Queries generated	# Data blocks accessed	# Voxels returned	Running time (secs)
1	`[0-10][0]?[0-1e10]*`	1	5	3	3.168
2	`[0-99][0]?[0-1e10]*`	1	41	3	26.522
3	`[40-60]??[0-1e10]*`	1	10	342	6.067
4	`[50]??*?*`	71	1	3,615,888	7.454
5	`[70]?[-5-1e10]*[5--1e10]*`	71	1	6,584	7.485
6	`[0-20]??[0-1e10]*?*`	71	10	3,593,696	159.97
7	`[80]?[-100-1e10]*[100--1e10]*[-100-1e10]*`	2,556	1	16,994,091	248.87
8	`[80-82]?[-100-1e10]*[100--1e10]*[-100-1e10]*`	2,556	3	50,565,859	757
9	`[0-99]?[-100-1e10]*[100--1e10]*[-100-1e10]*`	2,556	$\approx 117{,}500$	$\approx$ 1,685,529,000	$\approx 72{,}500$

uncompressed data blocks is sized at 800 megabytes, enough to hold 25 uncompressed blocks.

These tests show a spectrum of different cases. The first three queries generate only one specific query each. Starting with a cold cache, the overhead of decompressing data on the fly is paid for each block on each server. The subsequent three queries each generate 71 specific queries. With the data blocks loaded on each server being less than the size of the cache, we can tell that excellent cache reuse has been achieved. Similar scenarios can be observed with queries 7 and 8. Query 9 is a case that demonstrates our caching algorithm performing in a less optimal way. The values of query 9 are estimated from running about 2% of the query and extrapolating the values. As the time per single query and the number of data blocks accessed per query are almost constant, one can reasonably extrapolate the values.

From this we draw the following conclusion: the conventional LRU caching algorithm we're using is sufficient for many common cases. However, to truly achieve optimal results in general, caching algorithms need to be customized or fine tuned using heuristics that can be derived from domain and knowledge about patterns that typical users would investigate.

3.4 Discussion and Evaluation

3.4.1 Integration of Data Management

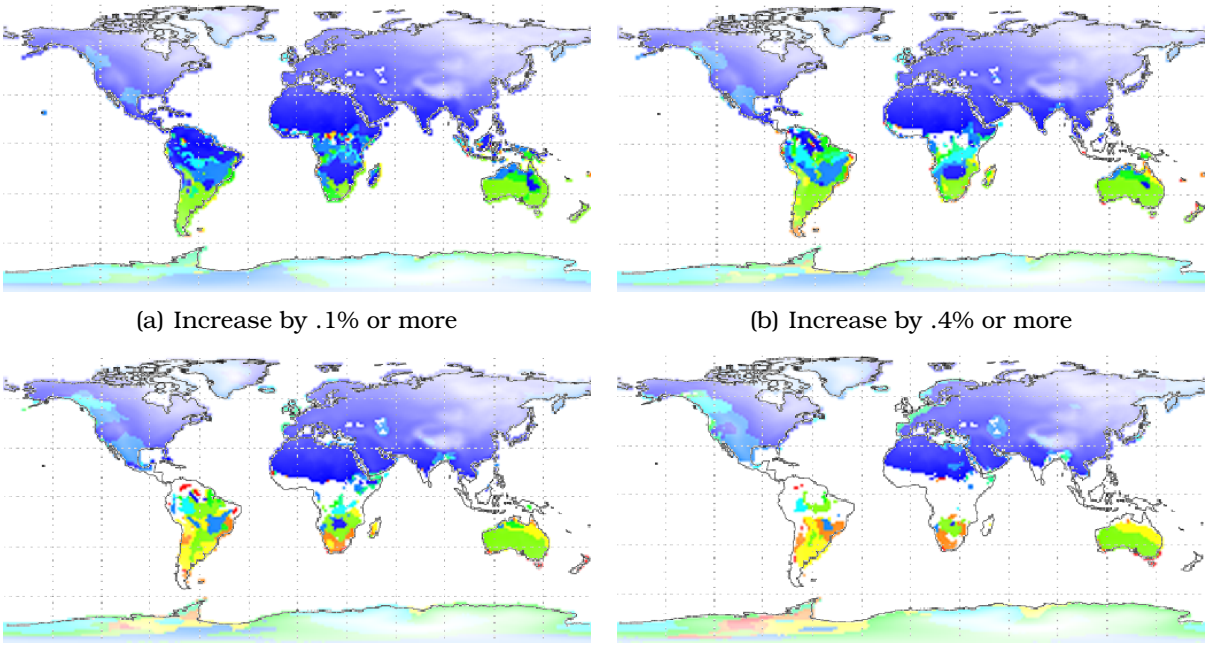
Our new approach provides a way to fundamentally use time to describe and extract temporal features. More generally, we aim at offering a more functional and more flexible technique of data selection and reduction. At the systems level, our approach only requires a set of servers that support compound range queries. In this regard, any base line implementations able to extract subsets defined by ranges is sufficient to provide the functionality. However, for large data sets a more sophisticated solution is necessary.

Efficiency demands a low-latency system that can run several concurrent queries in a combinatorial problem space in parallel on load-balanced data servers. Our implementation uses the scalable data servers described in chapter 2. Recent development in other systems (e.g. bitmap indexing [Stockinger et al., 2005]) could also meet these efficiency requirements. In particular, bitmap indexing also extracts subsets of data based on boolean range queries, and can therefore be easily adapted to work with our system. Changing to use one system versus another is very straightforward and should make no differences in functionality.

3.4.2 Query Generation

Our approach requires the user to express selections in terms of a regular expression. For many users, this may seem more complicated – albeit more flexible – than querying data in a visual (manual) setting. A best practice to formulate queries is hard to generalize, as useful queries are likely data and application dependent. This is quite similar to the practice of transfer function design. As with transfer functions, a practical way is through progressive refinement.

In the following example (Figure 3.12), we would like to demonstrate how progressive refinement of user queries can lead to the desired results. The user examines the variable TSA (air temperature at ground level in Kelvin) to identify the month when the temperature first rises over the course of a year. The first query a user might try is `?*T[.001-max]?*`, indicating that she is interested in locations that exhibit the following behavior: A rise in temperature as indicated (.1% or more) after a period of zero or more months of any behavior. Figure 3.12(a) is color-coded by the month in which such a rise in temperature can be observed. As one would expect, such a query is more geared to the Southern Hemisphere, as temperatures in the Northern Hemisphere will generally rise between January and February, marking almost the entire North in the respective color. Only the tips of Africa and South America as well as areas occupied by Rain Forrest exhibit a different behavior. In Figures 3.12(b), 3.12(c), and 3.12(d), the



(a) Increase by .1% or more

(b) Increase by .4% or more

(c) Increase by .8% or more

(d) Increase by 1.2% or more

Figure 3.12: 2m air temperature data of the year 2050, colored by temporal mark `T` representing the month as labeled in Figure 3.5

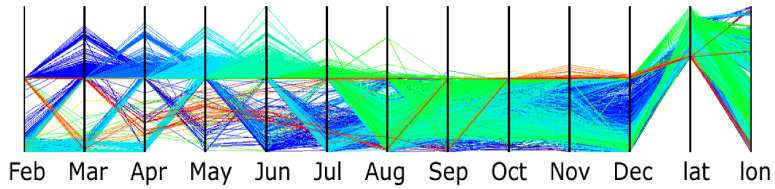
user adjusts the target value to .4%, .8%, and 1.2%, respectively, and a different and potentially more revealing distribution of timemarks can be observed. As exemplarily shown in this example, a user may approach the desired result by progressively refining the query.

3.4.3 Temporal Pattern Matching vs. Traditional Techniques

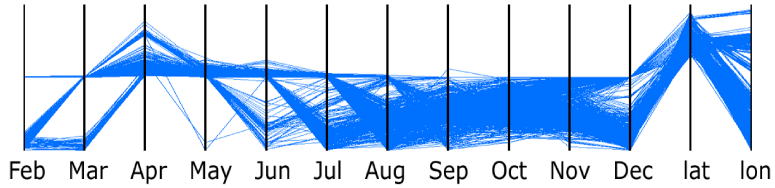
Many existing approaches have been developed to describe and extract subsets of data sets (e.g. scatterplots, histograms, and parallel coordinates). Using a parallel coordinates display, a user can manipulate ranges for one or many variables at a time, extracting a single range query. Our technique generates from every expression a number of range queries that are extracted and displayed simultaneously. For example, Figure 3.13 displays a number of traditional parallel coordinates plots with the data of the query `[-.4-.4]*T[.4-max]?*`, as displayed in Figure 3.3. This query expands into the following set of discrete queries: `T[.4-max]??????????`, `[-.4-.4]T[.4-max]?????????`, `[-.4-.4][-.4-.4]T[.4-max]????????`, `...`(11 queries in total). Data for all these queries is extracted, ignoring duplicate items as described in section 3.2.3. All queries are displayed in Figure 3.13(a), and selected queries 3 and 6 in Figures 3.13(b), and 3.13(c).

It is clear that using traditional parallel coordinates and discrete range queries, these results could not have been obtained directly. In this case with 11 queries, one would have to define 11 independent queries and combine the results. With only 11 queries, this can be achieved in a manual process. With a growing number of queries, this approach becomes infeasible. Additionally, independent queries might – and in the case shown here will – return duplicate items that would have to be resolved before being able to display the data as shown in Figure 3.3.

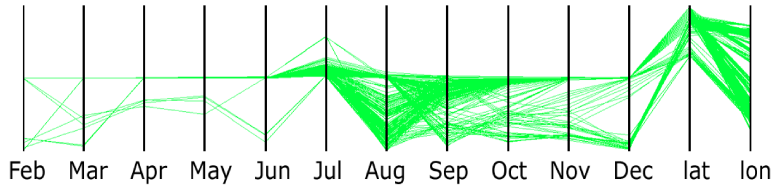
Scatterplots with a brushing interface allow for a less rigid way of data selection, albeit in a restricted number of dimensions. In all traditional techniques mentioned here, general temporal patterns cannot be extracted.



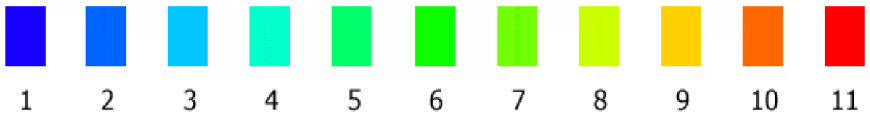
(a) Year 2050 of ELAI, all discrete queries



(b) Year 2050 of ELAI, discrete query #3



(c) Year 2050 of ELAI, discrete query #6



(d) Legend of discrete query numbers

Figure 3.13: Traditional parallel coordinates plot of the data shown in Figure 3.3(a), colored by the query number as labeled in Figure 3.13(d)

More comprehensive visualization packages such as Paraview have also addressed time-varying data visualization in a significant way. The most recent example of such effort is [Biddiscombe et al., 2007]. Previous work in the Paraview domain has not focused on any specific search structures for implementing functions such as to ask for what parts of a temperature field increases by at least 150% between consecutive time steps t_0 and t_1. Although one can write a filter to manually extract time step t_0 and t_1, find where the pressure field makes the requested change, and iterate to the next pair of consecutive time steps, this process could quickly become cumbersome and eventually intractable. In this context, Paraview can directly benefit from our work's automation and scalability (our search data structure and automated approach to expand partially defined query patterns).

Chapter 4

Event Sensitivity Driven Visualization of Scientific Data

Event-driven visualization is a novel approach to extract data from large-scale time-varying data sets prevalent in scientific simulations. Scientific data sets often contain complex elements of uncertainty and variability making them ill-suited for conventional data-oriented exploration. We extend the approach of temporal pattern extraction by introducing an event defining parameter λ that enables scientists to analyze the query's sensitivity for parameter change in detail. One aspect of this analysis is the difference in sensitivity between spatially distinct locations over time. Our approach establishes trends in query events by analyzing the shift through parameter space over the course of a time series, utilizing statistical methods to summarize the enormous amount of generated results. As the query parameter λ adds another degree of freedom and thus complexity, we gear our approach to perform best on massively parallel machines. Using recent large-scale climate modeling, satellite data, as well as a combustion simulation data set, we demonstrate that our approach is able to extract short-term and long-term trends from various data sets.

4.1 Introduction

Scientific data sets often contain complex elements of uncertainty and variability. Extracting subsets by individually restricting attribute ranges, i.e. using data-driven range queries, does not match the users' understanding of the data and subsets they wish to query. Hence, conventional data-oriented exploration tools are not necessarily applicable or well-suited to extract findings from complex large scale data sets.

Some existing techniques allow specifying queries that are not strictly data-driven, e.g. brushing of attribute clouds [Jänicke et al., 2008]. Recent advances in event-driven visualization have also created new ways to describe and extract data from large-scale scientific data sets with a pattern matching approach using regular expressions. This method is particularly well-suited for automating the process to extract temporal events from large scale data sets.

In previous approaches to event-driven visualization, the key parameters to define an event were often estimated by domain scientists. The values used are often refined iteratively to increase accuracy. Also, while this approach works best on events based on relative change of attribute values, applying a singular query with *absolute* attribute values to an entire data set does not generally lead to the intended results. In order to extract findings using query-driven visualization techniques, the event queries themselves, their parameters, and their sensitivity need to be quantified and understood.

In this work, we have extended the temporal pattern extraction approach to include an additional event-defining parameter λ. One aspect of our approach is to enable scientists to analyze the sensitivity of event queries by quantifying how sensitive an event is to a change of the parameter's value for a spatial location or region. Understanding event sensitivity is a necessity for enabling scientists to refine query parameters as needed to achieve more precise results.

For instance, for many variables in climate modeling, events based on absolute values oftentimes are only valid within a subset or region of the data set. Trying to define the

beginning of the growing season by using absolute values of the vegetation index has to be limited to a region of relatively homogeneous growth. To determine what absolute value could be used to define a meaningful event at a given location, one needs to analyze the sensitivity of the parameters defining the event.

A second aspect of event sensitivity analysis is the data's sensitivity to such a query, i.e. how event sensitivity differs between locations and over time. By analyzing event sensitivity with regard to a given event query over an entire time-varying data set, we enable the scientist to establish trends in events to shift through parameter space over time. For instance, in climate modeling, existence of such trends has proven to be difficult to verify, as such trends may be obscured by short-term climatic changes. In this work, we demonstrate that our approach is robust and able to recognize such trends.

Using a pattern matching approach for event extraction (see Chapter 3) leads to an already complex event-driven query system. Including an event-defining parameter λ adds another degree of freedom to it and makes the problem space even larger. Single user queries to analyze sensitivity may lead to a vast number of actual range queries on the order of thousands or more per data point, as shown by our results. Not only does this lead to the necessity of using massively parallel machines to extract information from large-scale data sets, it also produces a large amount of information for each individual data point. We have successfully employed statistics to summarize the enormous amount of event-driven query results to establish these trends in events over time.

Finally, we enhanced the scalable data server concept presented in Chapter 2 to support massively parallel machines ($\geq$ 10,000 processors) better by reducing preprocessing to a minimum. Scalability and massive parallelism are imperative for this approach, as analysis of the generated excessive problem space demands a growing number of data queries to the data servers.

4.2 Our Approach

4.2.1 Overview

Event-driven visualization techniques (see Chapter 3) have led to novel ways of describing and extracting subsets from large-scale scientific data sets. While this approach works well on relative or normalized data, it is unclear whether their method performs well with most other (non-normalized, absolute) scientific data sets.

The reason lies in the fact that one applies a single query with absolute attribute values to an entire data set that is most likely not homogeneous, i.e. events based on absolute values are valid only within a subset or region of the data set. Consider the case where the intensity of an event varies throughout the data set, and an event query that establishes a fixed absolute number as a threshold for the event. In this case, the used query will not detect an event that is similar in nature but dissimilar in absolute value space.

To overcome the shortcomings of this approach, we aim at providing methods and tools to analyze results of event queries locally in more detail. In order to determine an absolute value that results in meaningful event queries at a given location, one first needs to analyze and understand the sensitivity of the parameters defining the event.

We have extended the pattern matching approach from Chapter 3 to include an event-defining parameter λ. This event sensitivity – i.e. the impact of a change of the parameter's value on the result – is subsequently statistically analyzed. This analysis enables scientists to quantify the parameter's sensitivity for any given spatial location or region.

The second aspect of the event sensitivity analysis is the data's sensitivity itself to a given event query, i.e. how does event sensitivity differ between spatial locations and over time. This is of interest for many scientific applications since determining how an event query has to be adjusted over time to account for changes in the data set establishes *trends* in events over time.

Sections 4.2.2 and 4.2.3 delineate the necessary adaptations to the approach presented in Chapter 3, and focus on the changes made to the system necessary to support more sophisticated event-driven queries. The following Sections 4.2.4 and 4.2.5 describe our novel approach to analyze event sensitivity and event trends in detail.

4.2.2 Syntax of Query Language

As we are extending the querying concept introduced in Chapter 3, we employ range queries to select subsets of data sets. These range queries can contain wildcard characters like `*` and `?` as well as the temporal mark `T` to allow the definition of temporal events. Apart from the elements introduced in Chapter 3, we introduce the parameter λ that can be used instead of absolute values in a query. The range and stepping for the λ is to be provided separately.

We have used *flex* and *bison* to implement the query language's syntax. We have provided Figures 4.1 and 4.2 for a more detailed overview of the language's elements and highlighted the new elements concerning λ.

4.2.3 System Design

To solve problems of this complexity that generate billions of queries, a scalable system must be present. Our solution is geared towards massively parallel modern supercomputers. By utilizing these resources, we harness parallel file systems and high-speed networks to reduce I/O costs and interactively query large data. We use methods to process and later query data sets in netCDF format since our collaborators output their data in this format.

Our system allows the scientist to specify which variables in their data they are interested in with an XML configuration file. To further enhance the I/O time and resulting query performance, restrictions on the variables and dimensions may be specified to reduce the amount of data that is read in. To make use of parallel file systems and the

```
/* floating point numbers */
\-?[0-9]+
| \-?"."([0-9])+
| \-?([0-9])+"."([0-9])*
| \-?([0-9])+\.?([0-9])*[eE]\-?([0-9])+
      return VALUE;

\[    return RBEGIN;
\]    return REND;
\?    return RWILDCARD;
\*    return MREXPANDER;
\-    return RTO;
\T    return TIMEMARKER;

\L    return LAMBDA;

[ \t\n]   ;

.         ;
```

Figure 4.1: Tokens defined for lexical analyzer (*flex*), with elements concerning λ highlighted.

```
statement:
         | statement range
         | statement multirange
         | statement TIMEMARKER range
         | statement TIMEMARKER multirange
         ;

range: RBEGIN values REND
     | RWILDCARD
     ;

values: VALUE RTO VALUE
      | VALUE RTO LAMBDA
      | LAMBDA RTO VALUE
      | LAMBDA RTO LAMBDA
      | LAMBDA
      | VALUE
      | RWILDCARD
      ;

multirange: range MREXPANDER
          ;
```

Figure 4.2: Grammar rules (*bison* grammar file), with elements concerning λ highlighted.

netCDF file format, we chose to use the Parallel netCDF [Li et al., 2003] library for high performance I/O. This library is built on top of MPI-2 [MPI Forum, 1997], allowing it to be executed on wide variety of parallel file systems. Since it is common for our collaborators to store time steps of their simulation data in separate files, we use a greedy algorithm that assigns files to processors for I/O in such a way to increase the overall I/O performance.

After reading data into memory, the data points are then converted to an item-based format. These items are then randomly distributed to all the processors for load balancing the resulting queries. Each processor locally sorts its data and then builds an M-ary search tree structure on top of it for subsequent querying.

4.2.4 Event Sensitivity Analysis

In the previous Sections, we have extended the pattern matching approach to include an event-defining parameter λ and described a parallel approach geared at massively parallel systems that can handle the extreme number of queries necessary for sensitivity analysis. In this Section, we focus on the mathematical foundation for our approach and how query results can be visualized and interpreted.

The parameter for event sensitivity analysis λ obviously has an impact on the user-specified event. Its value may change the time step at which the user event is registering. For instance, if we are looking for the simple event of a variable growing beyond a given threshold, a sample query could be

```
[0.0 - 4.0]*T[4.0 - 1e8][?]*,
```

i.e. zero or more time steps of `[0.0 - 4.0]` (below the threshold of 4.0) followed by a time step of `[4.0 - 1e8]` (beyond the threshold value). The final `[?]*` indicates that after that initial event any value is accepted for the purpose of this query. The `T` marks the time step at which we register the event, and it is the resulting time step at which the value grows beyond the threshold value for the first time. The threshold value 4.0

in this example can be adjusted, and doing so may lead to a different event time. To automate this process, we allow for the query to be rewritten as

```
[0.0 - λ]*T[λ - 1e8][?]*.
```

A value range and stepping for the sensitivity analysis has to be provided, e.g. $\Lambda = (3.0, 5.0)$ with a stepping of $\Delta\lambda = 0.1$. This range and stepping may be subsequently refined if needed. For instance, in this example, we may find that if we choose a threshold value of 3.7 or below, the user event registers at time step 12, while it registers at time step 13 for a higher λ. To find this result, we ran 21 queries with λ being 3.0, 3.1, 3.2, ..., 5.0. All of these 21 queries still contain wildcard characters ("$*$") and are expanded into a number of discrete queries as described in Section 3.2.3.

Analyzing the results gathered from this event sensitivity analysis enables users to specify more accurate event queries. For the query discussed above, a change of the threshold value from 3.7 to 3.8 changes the outcome of the event query at that location while a change from 3.8 to 3.9 has no impact. If a scientist is trying to find locations that experience a similar event, a threshold value of 3.7 would be a better choice than 4.0. Our approach helps scientists to become aware of such sensitivities in their event queries.

Further extending this concept, we run the same queries on many series of a data set. A series may be one of several things: (1) For data sets from the earth sciences, we can consider a single year a series, i.e. a climate modeling data set running for 100 years consists of 100 series, or (2) for other climate modeling data sets, "suites" of slightly different climate models may be considered to consist of several series, or (3) for other data sets we may create different series by restricting a secondary variable – referred to as secondary λ – to a number of different value ranges.

For the following mathematical description, case (1) is assumed, other cases analogous. Given an event description or query d_λ with λ as its parameter for event sensitivity analysis and a step size $\Delta\lambda$, t time steps, a series, we define the event time as $event(d_\lambda, a) = t_{\lambda,a}$. We assume

$$\forall a,t \quad \exists \lambda_{t,\Delta\lambda,a}: \quad t_{\lambda,a}+1 \leq t_{\lambda+\Delta\lambda,a}. \tag{4.1}$$

Ignoring the step size $\Delta\lambda$ for now, we may phrase this as

> In series (year) a, the event transition from time step t to the next larger t is detected by using $\lambda_{t,a}$ for λ in d_λ.

In other words, for the given series a, the value of $\lambda_{t,a}$ describes the value at which the event time t changes. For the example in the second paragraph of this Section, $\lambda_{t,a} = 3.7$.

In this example, we chose a step size of $\Delta\lambda = 0.1$. The step size defines the granularity of the obtained results, i.e. had we chosen a step size of $\Delta\lambda = 0.01$, we may have found a better value for $\lambda_{t,a}$. For instance, in the above example, $\lambda_{t,a}$ may be 3.74. However, instead of 21 queries, a smaller $\Delta\lambda$ value of 0.01 results in 201 queries. We thus have to trade query speed for accuracy. Technically, every $\lambda_{t,a}$ has to be written as $\lambda_{t,\Delta\lambda,a}$, as it is also dependent of the step size $\Delta\lambda$.

We can reduce the cost of greater accuracy by recursively refining the step size $\Delta\lambda$ and the value range Λ. In the last paragraph, we had to run 201 queries to obtain the result 3.74 with an error of ± 0.01. If we run the query $\Lambda = (3.0, 5.0), \Delta\lambda = 1.0$ (3 queries), we find $\lambda_{t,\Delta\lambda,a} = 3.0$. We can now recursively refine the range $\Lambda = (3.0, 4.0)$ since we already know that a more accurate value for $\lambda_{t,a}$ has to be contained in this range. With $\Lambda = (3.0, 4.0), \Delta\lambda = 0.1$ (9 extra queries) we find $\lambda_{t,\Delta\lambda,a} = 3.7$, and with $\Lambda = (3.7, 3.8), \Delta\lambda = 0.01$ (9 extra queries) we find $\lambda_{t,\Delta\lambda,a} = 3.74$. Note that we do not need to repeat the queries for values 3.0 and 4.0 for the second round, and neither 3.7 and 3.8 in the last round of queries. Thus, with $21 (= 9+9+3)$ queries instead of the original 201 we are able to achieve the same level of accuracy.

If such an analysis of event sensitivity is executed at every data location, a scientist will obtain a vast amount of additional information for every single data location. In order to reduce this information to a comprehensible depiction, we apply statistics before

visualizing the findings. In the following Section we demonstrate how statistics can be applied to a time series of data sets to extract trends in queried events over time.

4.2.5 Event Trend Sensitivity over Time Series

Since the series is an ordered set, we can compute a linear fit $\overline{\lambda_t}(a) = s_t a + i_t$ for each t. The slope of this linear fit s_t indicates how much the parameter λ for event d_λ needs to be adjusted on average per series unit a to account for the shift in event time t over the course of the entire series a. It therefore describes the *trend* of event d to shift in t over the series a.

Exemplary, from a recent climate modeling data set (see Section 4.3), we are using the variable ELAI (exposed one-sided leaf area index) to find the beginning of the growing season, defining this event as an absolute value of greater than λ after a period of zero or more time steps where the absolute value was lower. For a given location, we can then run this query for every year and every month of the 100-year series as well as using a range of lambdas (threshold values). Figure 4.3 depicts this parameter analysis for a single location, colored by the time step in which the given event has been found. We can refine the range of lambda values Λ as well as the step size $\Delta\lambda$ for a transition to get a more detailed transition, as on the right of the figure. In every year, the event transition from one time step to the next larger time step can be detected by using $\lambda_{t,a}$ in the event description that marks the boundary between time steps. For each transition, we can compute a linear fit of these values over the entire series. Its slope indicates how much the parameter λ for an event needs to be adjusted on average over the course of the series to account for the shift in the event time. It therefore describes the trend of an event to shift in time over the course of the series. Here, we would have to increase λ over the course of 100 years to obtain the same time step value for this event. Thus, the beginning of the growing season, in this example, occurs earlier within a year over the course of the series of 100 years.

Interannual variability is clearly visible in Figure 4.3, especially for refined $\Delta\lambda$ (right

diagram). By applying statistics like linear regression, we are able to establish a clear trend from this. Other statistics such as higher level regressions, variance, etc. may be used for other data sets and different events. In Section 4.3.1, we present another method to reduce the obtained results statistically.

Please note that for every point in Figure 4.3, up to 12 queries (one query per month) have to be run to determine its event time and hence its color. For a series containing 100 years and a λ refinement containing about 200 steps, a total of about 240,000 queries *per data point* may be necessary. We facilitate such great query demands by leveraging massively parallel machines and a data management system as described in Section 4.2.3.

With the methods established in Section 4.2, we are able to analyze event sensitivity and establish trends in events over time in scientific data sets.

4.3 Results

We evaluate our approach using data sets from various domains.

The first one is a scientific simulation data set, a combustion simulation with a grid size of $480 \times 720 \times 120$ and 5 variables over 122 time steps.

In addition, we also used a climate modeling data set consisting of 71 variables (hereinafter referred to as IPCC_A2) output from a global land surface model running within a general circulation model performing a projection of Earth's climate from the year 2000 to 2099. We are using monthly averaged data with 12 time steps per year, comprising 1200 time steps in total on a grid of size 256×128. This data set is used for timing results in Section 4.3.3.

We are also using 150 years of data each (1850 - 2000) from a suite of climate modeling data sets from the "Carbon-Land Model Intercomparison Project" (C-LAMP) [Hoffman et al., 2007]. The purpose of this model-measurement inter comparison simulation is

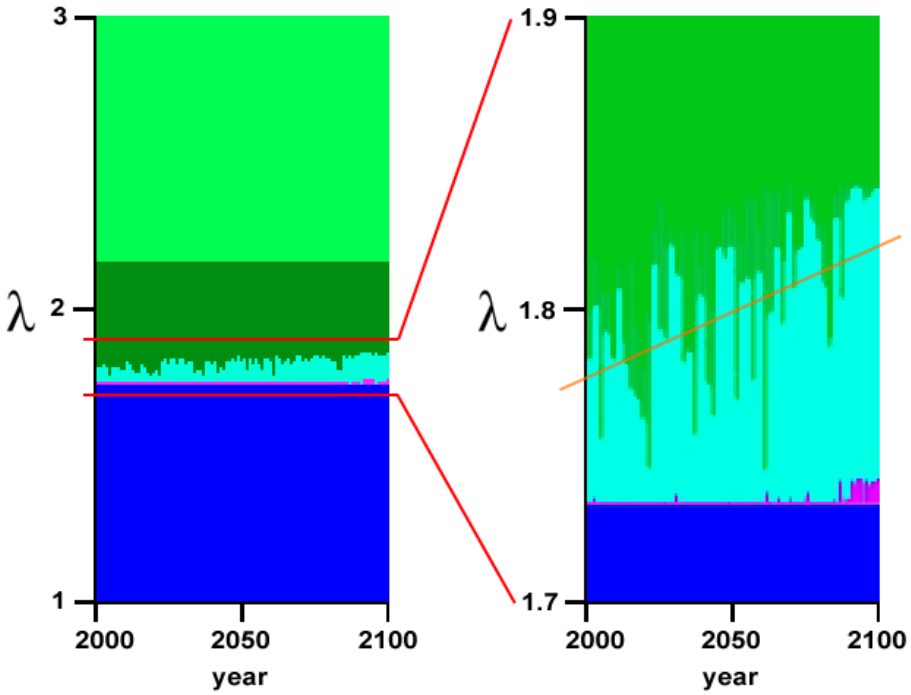


Figure 4.3: An example parameter analysis for a single location, colored by the time step in which the given event has been found. A refined $\Delta\lambda$ leads to more accurate results (right figure).

to allow the international scientific community to evaluate the performance of biogeochemical models normally coupled to general circulation models (GCMs). In the C-LAMP project, the objective is to examine the ability of the models to reproduce surface carbon and energy fluxes at multiple sites, and to examine the influence of climate variability, prescribed atmospheric CO_2 concentrations, and land cover change on terrestrial carbon fluxes during the 20th century. As typical with any climate modeling efforts, the same models would be run under different conditions. To demonstrate our approach and typical use cases, we show two different types of runs: (i) control run, and (ii) full run (varying climate, CO_2 and N deposition transient run). Under each type of run, there are two models from NCAR Community Climate System Model Version 3 (CCSM3), specifically CCSM3-CASA and CCSM3-CN, leading to a total of four run named "Control(CASA)", "Full(CASA)", "Control(CN)" and "Full(CN)". For these models, we are particularly interested to see how model differences translate into different trends and their intensities.

Finally, we have applied our approach to data from NASA's Moderate Resolution Imaging Spectroradiometer (MODIS) database containing years worth of high-resolution satellite data sets of Earth. In particular, we have computed the "Normalized Difference Vegetation Index" (NDVI) for the years 2001 through 2007 from it using 2 bands from the satellite data – near-infrared (NIR) and red (RED) – and the following formula:

$$\mathrm{NDVI} = \frac{(\mathrm{NIR} - \mathrm{RED})}{(\mathrm{NIR} + \mathrm{RED})} \tag{4.2}$$

It has been shown that NDVI is directly related to the photosynthetic capacity. It is in many ways similar to ELAI which is used to predict photosynthetic primary production.

4.3.1 Visualizing Event Sensitivity in Scientific Data Sets

Unlike climate modeling simulations and satellite data, most scientific data sets do not consist of a series of independent individual runs or years. To fully utilize the capabilities

of our approach with such data sets, we allow scientists to define and vary a second variable, as described in Section 4.2.4.

We have chosen a well understood simulation from the field of combustion science to demonstrate the utility of our approach for such a data set. Amongst others, it contains the variables "mixture fraction" (mixfrac), "mass fraction of hydroxyl radical" (OH), and "scalar dissipation χ" (chi). In this combustion simulation data set, scientists are interested in finding and tracing areas of interest over time. One example of such an area of interest in non-premixed combustion is extinction of the flame in areas where fuel and oxidizer are in stoichiometric proportions [Akiba et al., 2007]. Under normal circumstances, the spatial locations with a stoichiometric mixture fraction of 0.42 are a good representation of a fully burning flame. However, local extinction can occur in areas where other chemical effects come into play (e.g. local turbulence). One effect that can cause this behavior is a local value of the mass fraction of the hydroxyl radical below 0.0007 and a high value of the scalar dissipation rate χ. Areas that maintain these characteristics are temporarily extinguished, as they do not support the chemical conditions to maintain a burning flame.

We analyzed whether the values of 0.42 for mixture fraction and an upper limit of 0.0007 for the mass fraction of the hydroxyl radical are sensitive to small changes, i.e. whether the event defined as "extinction" is stable with regard to mixfrac and OH. The exact event we are looking for is the value of OH dropping below a threshold value λ, and mixfrac being contained in a varying size range centered around 0.42. As for χ, we will later on filter the query results according to a separate threshold.

Figure 4.4 depicts – for a single data point – when the event as described first happens for different threshold values of mixfrac and OH. We notice how an increase of the range of mixfrac does not change the event time up to a range of 0.10. At that point the valid range for mixfrac is defined as $0.32 - 0.52$ which is well beyond stoichiometric proportions [Akiba et al., 2007], i.e. not relevant to define extinct areas. For the range from 0.00 to 0.10, however, the event time behavior is very stable, with only two different event

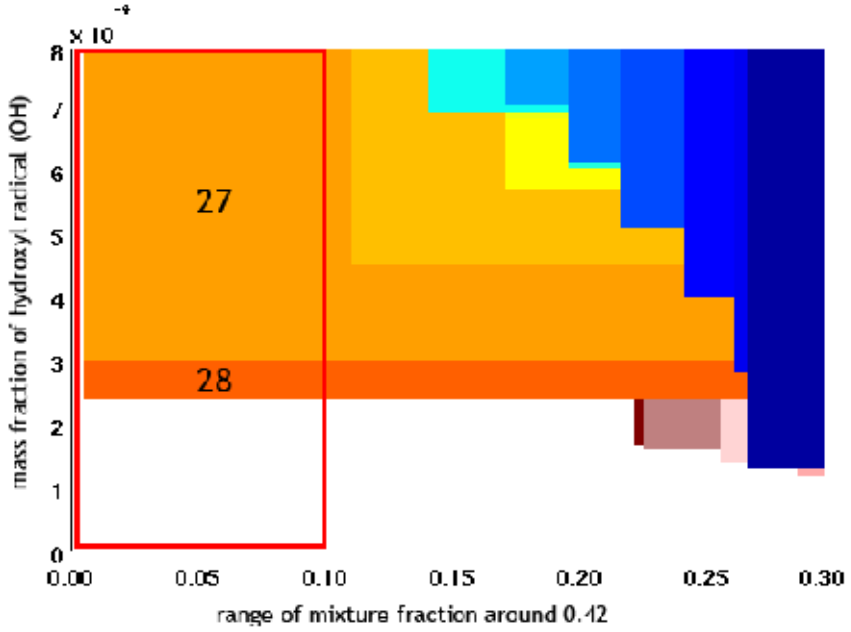


Figure 4.4: Local extinction event for a single data location and varying values of OH and mixture fraction, colored by time step.

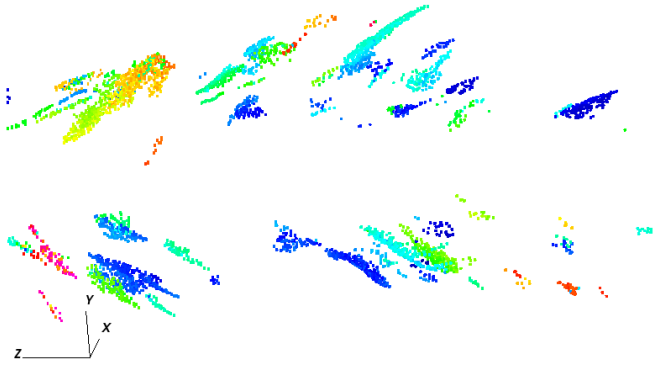
times (27 and 28), depending on the threshold value for OH. If OH has to drop below 0.0003 to define this event, it first happens in time step 28, with any larger threshold values in time step 27. Note that for a threshold value below 0.00026 the event never occurs at that particular location.

The statistics we apply in this case to reduce the obtained results to be able to display them differ somewhat from the approach discussed in Section 4.2.5. Here, we chose to visualize each data point based on how the event time differs for mixfrac ranges up to 0.10 (see red box in Figure 4.4). We extract the "first hit" (27) – the earliest event time – and the range of different event times ($28 - 27 = 1$).

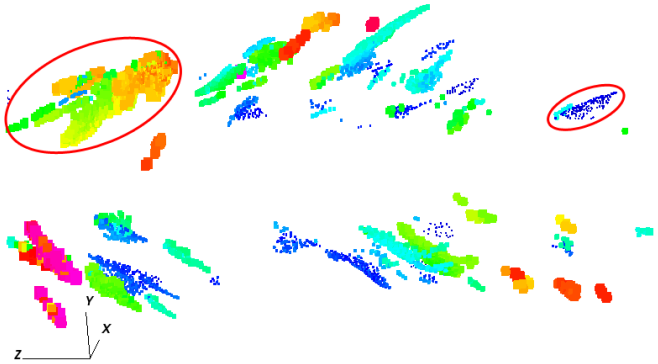
In Figure 4.5(b), we have depicted all the data points that exhibit such an extinction feature, with a χ value of greater than $18,000$. We color each data point by the time step of its "first hit", and scale its size by the range of different event times. The image clearly shows how extinction areas develop at different time steps and locations, and how these areas seem to move in time and space. In addition, many of these data points depend on the threshold value of OH, as can be seen by the size of each point. For instance, in the top left area of the image, many data points have a wider range of different event times, while event points in the top right corner seem to not depend on the exact value of OH.

In Figure 4.5(c), we display all such points with a χ value of greater than $6,000$. While the event not includes more data points, clear structures emerge from it as well, depicting how event points move in time and location.

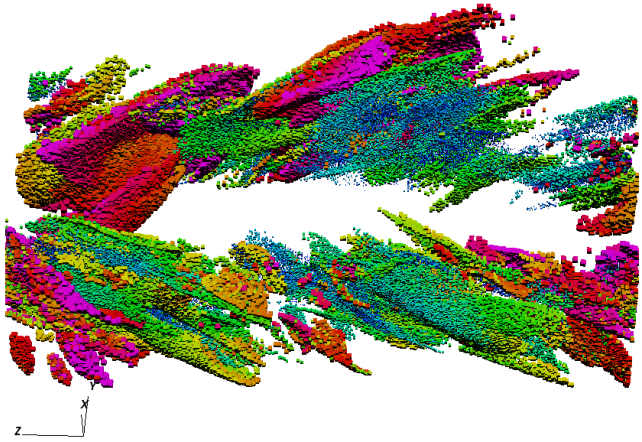
Ultimately, these results may lead to a better understanding of how sensitive the observed event (here: extinction areas) is to slight changes in threshold values. In this example, we used a rather simple statistical measure to depict a data point's attributes. More sophisticated approaches, especially when created by or in cooperation with domain scientists, are bound to expose more and more meaningful relationships between an event and the sensitivity of the variables involved.



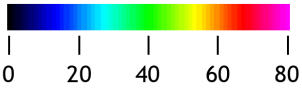
(a) χ value of 18,000 or more



(b) Same locations than Figure 4.5(a), scaled by range of event times



(c) χ value of 6,000 or more



(d) Legend of time steps

Figure 4.5: Data locations exhibiting extinction characteristics and specified χ values, colored by time step of event occurrence.

4.3.2 Visualizing Temporal Trends in Climate Data Sets and Sensory Data

C-LAMP Climate Modeling Data Set Suite

As described in Section 4.2.5, we are using the variable ELAI (exposed one-sided leaf area index) to find the beginning of the growing season ("green-up"), defining this event as an absolute value of greater than λ after a period of zero or more time steps where the absolute value was lower. Figure 4.3 displays the resulting parameter analysis for a single location of the IPCC_A2 data set, colored by the time step in which the given event has been found. As described in more detail in Section 4.2.5, the slope of an event transition indicates how much the parameter λ for an event needs to be adjusted on average over the course of the series to account for the shift in event time and therefore describes the trend of this event to shift in time.

Due to the limitation of screen space, we opt to display the resulting data set by coloring each spatial location according to the slope (measured as $\Delta\lambda$ per annum) of a selected transition. In Figures 4.6 and 4.7, we chose to analyze the transition between the months of April and May in the selected suite of C-LAMP climate modeling runs, to compare how the event based on ELAI as described above changes over time at each location. Even though each pixel's coloration is based solely on analysis of its own values over time, underlying patterns can be recognized quite easily in these images. In Figure

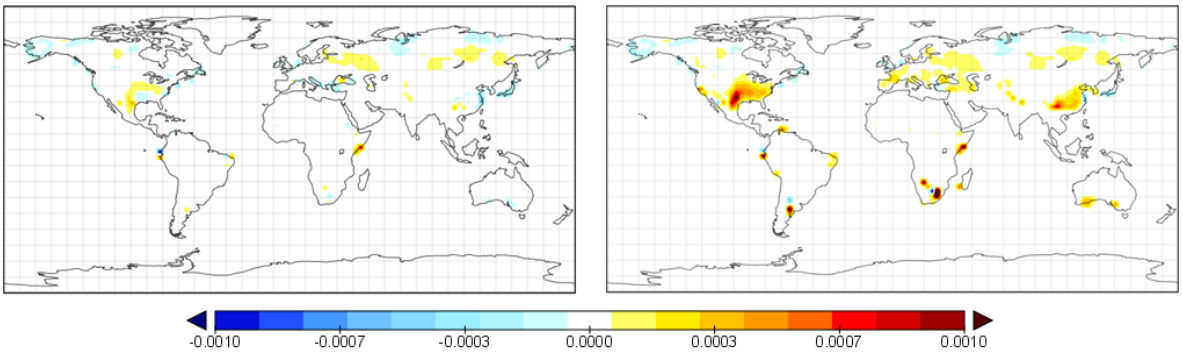


Figure 4.6: Analysis of the transition between months April and May for "Control(CASA)" and "Full(CASA)". Measurements are displayed as $\Delta\lambda$ per annum.

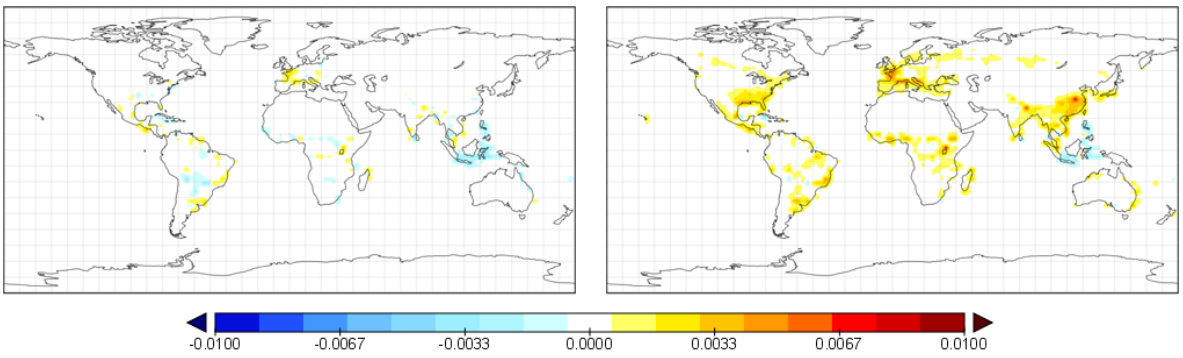


Figure 4.7: Analysis of the transition between months April and May for "Control(CN)" and "Full(CN)". Measurements are displayed as $\Delta\lambda$ per annum.

4.6, we notice that while the basic trends in certain areas of the globe are very similar between models "Control(CASA)" and "Full(CASA)", their intensity is very different. For the South-Eastern U.S. as well as Eastern Europe, the "green-up" happens earlier and earlier on average over the course of 150 years, according to these two models. Parts of China exhibit a clear trend to earlier "green-up" events in the "Full(CASA)" model, and almost no trend in the "Control(CASA)" model. Figure 4.7 depicts the differences between models "Full(CN)" and "Control(CN)" in the same manner with more intense temporal trends showing in the "Full(CN)" run as well. The intensity of the events of the CASA runs versus the CN runs differs by about one order of magnitude, as their respective legends depict.

NASA MODIS Satellite Data Set

Taking advantage of our approach's λ parameter, we can use the "green-up" event definition based on ELAI as before to define the same event based on the Normalized Difference Vegetation Index (NDVI). In this case, however, we have 46 time steps per year – one every 8 days – but only 7 years of data. Temporal trends in satellite data sets are particularly hard to detect due to the inherent noise and inter-annual variability of sensory climate data. In addition, any trend detected over the course of 7 years is inclined to be overshadowed by annual variability. Again, we color each spatial location according to the slope of a selected transition.

Keeping in mind that each pixel's coloration is based solely on analysis of its own values over the 7 years used here, the statistical confidence of this result is naturally lower than in Figures 4.6 and 4.7, where 150 years were used to establish a trend. Nonetheless, clear structures can be recognized around the Great Lakes in Figure 4.8. Areas of the "green-up" event shifting in time forward or backward are distinctly clustered with clear borders. The structure marked with a red circle actually outlines New York's Adirondack Park Reserve very precisely, indicating that the vegetation, land use, and altitude of the Park Reserve and the surrounding area has a strong impact on the

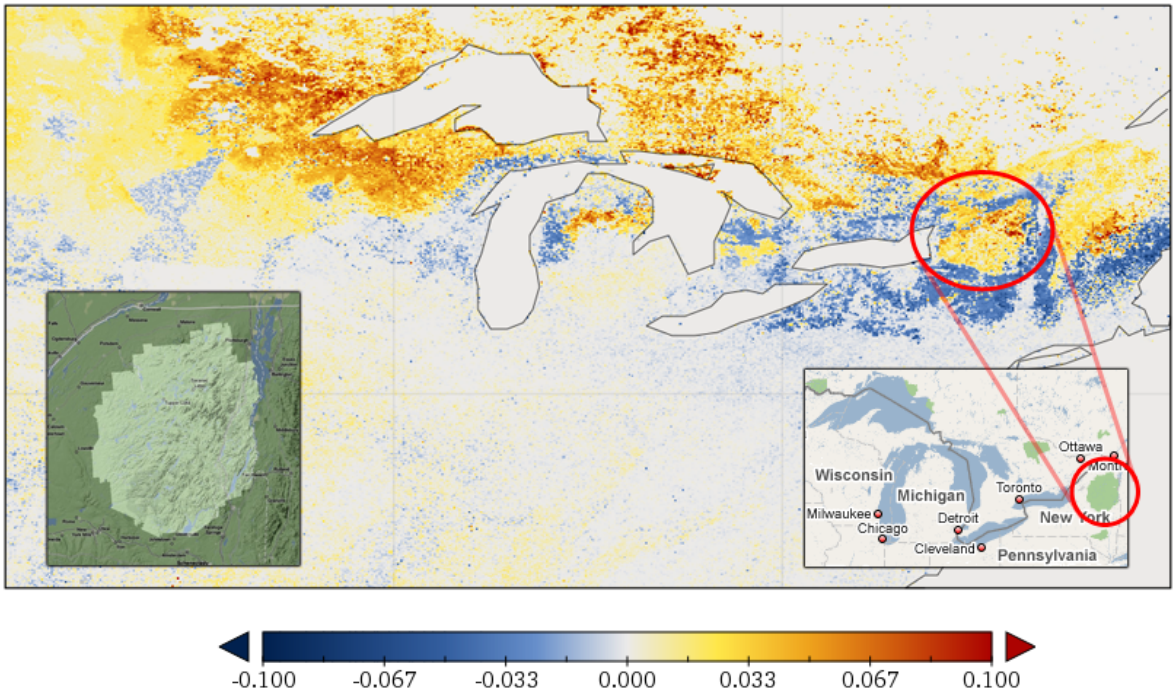


Figure 4.8: Temporal trends in MODIS satellite data for early March exhibit clear structures of events shifting in time. The red/yellow cluster marked outlines New York's Adirondack National Park Reserve. Measurements are displayed as $\Delta\lambda$ per annum.

trend of the "green-up" event in that area. By visually comparing the terrain map in the lower left corner of the image to the results of the event trend analysis, it is noticeable that the highest positive shift (i.e. earlier "green-up") is visible in areas of high altitude.

4.3.3 Timing and Scalability Results

Since independent points of data are being queried and analyzed one by one, the problem is embarrassingly parallel and highly scalable. In our implementation, we load multiple copies of the data set among groups of processors. Instead of having each group perform I/O and preprocessing on the data set, only one group does I/O and preprocessing and simply broadcasts the resulting data to the other groups. The data points to be queried are then assigned to the groups in a round robin fashion for load balancing. When all of the groups are finished querying and analyzing their set of points, the entire set of results for all the points is written in parallel. We exploited the parallelism of our problem and conducted experiments on the Jaguar system at Oak Ridge National Laboratories. Jaguar is a Cray XT4 system with 7,832 quad-core 2.1 GHz AMD Opteron processors, totaling to 31,328 cores. In some experiments, we used as many as 16,384 cores because of the immense problem space.

To test the scalability of the system, experiments with an increasing number of query groups were conducted on the leaf area index variable of the IPCC_A2 climate modeling data set. Each month was used as a temporal mark per 100 years in the conceptual queries issued. This means that 12 queries per point per 100 years were generated, totaling to a maximum of 1,200 queries per point per lambda. All 1,200 queries may not have to be issued per point to find an event, however, this seemingly small resolution experiment shows the extravagant problem space that is generated. Results for the scalability experiment are shown in Figure 4.9. The total aggregate number of queries issued was roughly 6.8 million. Because of the parallel nature of the problem, the total query time scales near-optimally with the number of processors. The reading time of the data set does not scale, and this is because only one group does I/O and prepro-

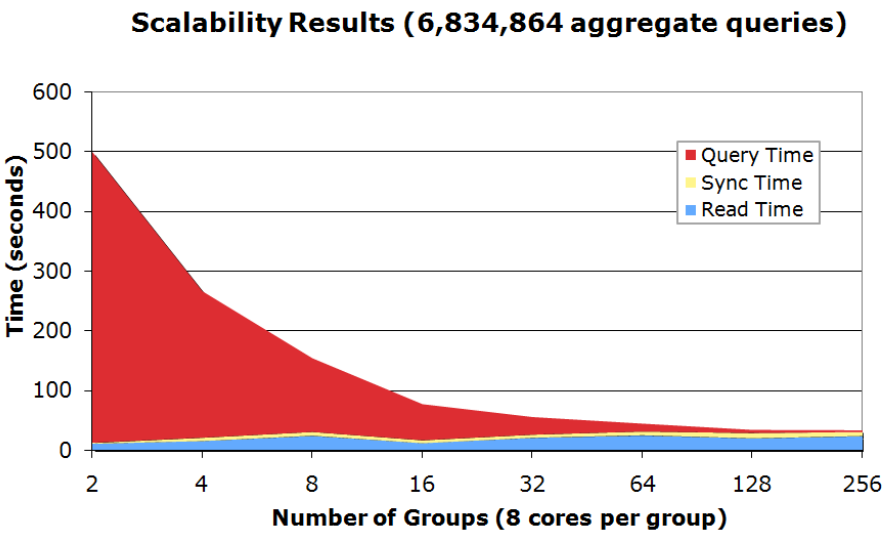


Figure 4.9: Scalability Results for the IPCC_A2 data set for a varying number of data server groups. Each run consists of about 6.8 million queries.

cessing of the data set before broadcasting copies to the other groups. The reading time actually shows a slight increase because of the time it takes to broadcast the data to the other groups. The synchronization time shows the amount of time that is spent in synchronizing the communication between the querier and the servers when querying for data. Although this time is negligible in this example, it becomes more prevalent as the aggregate number of queries increases.

The importance of scalability is paramount, especially when the problem size grows enough to generate billions of queries. Table 4.1 shows some of the timing results at different resolutions of the MODIS data set with different amounts of groups. In the experiments, the continental United States of the MODIS data set was queried with 7 years of 46 time steps per year. This totals to a maximum of 322 queries per point per lambda. Because of the exceptionally high resolution of the data set, the problem becomes unfeasible to solve on a small scale system. In our experiment, we found it necessary to use 16,384 cores to handle the roughly 5.5 billion queries generated by the 900 x 1800 resolution MODIS data set. The synchronization times also become more noticeable as the number of aggregate queries increases. This is primarily because each query incurs an implicit synchronization between the servers and the querier, which turns out to be costly as the aggregate number of queries grows to the billions. Table 4.1 also shows the average queries per second that are executed per group. This time decreases as the resolution increases because of the larger amount of data that is being queried, but even at 900 x 1800 resolution we still achieve roughly 1,800 queries per second.

Table 4.1: Timing results for MODIS data queries with different resolutions for the continental U.S. The final run utilized 16,384 processors in 1,024 groups and contained more than 5.5 billion queries. It lasted almost one hour.

#Procs Per Group	Groups	Resolution	Read Time	Query Time	Sync Time	Total	#Queries Issued	Queries Per Second Per Group
8	512	225 x 450	38.56	142.04	43.86	224.46	563,599,360	7750
8	256	450 x 900	24.75	2966.33	188.74	3179.82	2,122,741,760	2795
16	1024	900 x 1800	67.69	2913.37	342.816	3323.88	5,506,457,600	1846

Chapter 5

Conclusions

In this dissertation, we have addressed the problems as laid out in chapter 1. We will now summarize our findings and results as well as discuss opportunities for further research in this field.

In chapter 4, we have demonstrated the capability to analyze parameter sensitivity in event-based query-driven visualization. We have utilized this capability to enable climate scientists to establish trends in events to shift through parameter space over time, and we have statistically analyzed and visualized these trends in various large-scale climate modeling data sets.

In order to allow for the enormous amount of discrete data queries necessary to support event sensitivity driven visualization, we have enhanced our data server framework from chapter 2 and successfully demonstrated its capabilities by applying it to terascale data on a massively parallel infrastructure.

In chapter 3, we have presented and demonstrated a system that increases the utility of range query systems. By providing a high-level language for temporal data analysis, we allow scientists to consider their data at a higher level, describing multivariate features of interest in a language that allows the specification of uncertainty. The results show that significant scientific understanding can be achieved across application domains when the richness of data description is increased.

The approach has the potential to solve many large-scale data understanding problems in scientific simulations. In applications where scientists can only describe results in a "fuzzy" manner, systems that directly measure and express uncertainty have great promise to provide heretofore undiscovered scientific insight.

The pattern matching system decomposes general queries into a series of specific queries that are executed by the scalable data server framework presented in chapter 2. However, the two systems are not directly coupled. Thus, it would be possible to integrate other indexing systems that use compound range queries such as the FastBit bitmap indexing system [Stockinger et al., 2005] into our system to explore the performance characteristics of such a combination.

In chapter 2, we have presented an efficient and scalable data server framework that maintains a nearly optimal load-balance for almost all types of compound ranges queries. It relies on external sorting according to a high-dimensional space filling curve order in attribute space, and an efficient M-ary search tree to efficiently skip irrelevant voxels in the dataset. These enabling techniques make it feasible to use more attributes than before by simply increasing the number of servers when required. As long as a sufficient number of servers are available, any amount of data can be handled.

For a user of our system, e.g. an application scientist, it is common to interact with the dataset of a simulation to verify correctness or to explore newly formulated hypotheses. To this end, one should be able to navigate not only in time and space, but also in the high-dimensional attribute space. By treating simulated and derived variables, as well as temporal and spatial coordinates, uniformly as attributes, some types of queries that have not been widely used in current visualization production are now convenient operations. Additionally, in a way, every server in our system accepts queries in the same way that web servers function. This allows parallel servers to operate persistently, while the client can dynamically start, stop or simply shutdown.

Our approach does have the following limitations. Due to technical considerations described in section 3.2.1, the data size can increase. In few cases, this increase can be

five fold, but can be mitigated by applying data compression and filtering techniques also described in section 3.2.1. Additionally, the loss of inherent neighborhood information and the sparse data set resulting from a user query limits the use of traditional rendering techniques. As we lack the information to compute gradients in a local neighborhood, shaded visualizations cannot be easily achieved. While this problem has long persisted in sparse volume rendering, our work exposes the need for further research in that area.

Future potential research opportunities includes allowing temporal queries to extend across multiple time steps. Our current system allows the discovery of temporal events which occur between two successive time steps. Additional understanding can be achieved by allowing the description and detection of events that cross many time steps (e.g. “Increase of snowfall of 40% or more over a range of up to 3 months”).

Expressive systems such as ours often spur interactive data exploration. Users frequently issue multiple general queries to explore similar temporal features. We suggest to apply illustrative visualization methods to effectively display the composition of multiple query results into single views. We believe that one feasible way to achieve this goal would be volumetric texture generation and new color design methods.

Bibliography

Bibliography

[Abdulla et al., 2001] Abdulla, G., Baldwin, C., Critchlow, T., Kamimura, R., Lozares, I., Musick, R., Tang, N. A., Lee, B. S., and Snapp, R. (2001). Approximate ad-hoc query engine for simulation data. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 255–256, New York, NY, USA. ACM.

[Akiba et al., 2007] Akiba, H., Ma, K.-L., Chen, J. H., and Hawkes, E. R. (2007). Visualizing multivariate volume data from turbulent combustion simulations. *Computing in Science & Engineering*, 9(2):76–83.

[Ananthakrishnan et al., 2007] Ananthakrishnan, R., Bernholdt, D. E., Bharathi, S., Brown, D., Chen, M., Chervenak, A. L., Cinquini, L., Drach, R., Foster, I. T., Fox, P., Fraser, D., Halliday, K., Hankin, S., Jones, P., Kesselman, C., Middleton, D. E., Schwidder, J., Schweitzer, R., Schuler, R., Shoshani, A., Siebenlist, F., Sim, A., Strand, W. G., Wilhelmi, N., Su, M., and Williams, D. N. (2007). Building a global federation system for climate change research: The earth system grid center for enabling technologies (ESG-CET). *Journal of Physics: Conference Series*, 78:012050 (7pp).

[Bajaj et al., 1996] Bajaj, C., Pascucci, V., and Schikore, D. (1996). Fast isocontouring for improved interactivity. In *Symposium on Volume Visualization 1996*, pages 39–46.

[Bethel, 2000] Bethel, W. (2000). Visualization dot com. *IEEE Computer Graphics and Applications*, 20(3):17–20.

[Biddiscombe et al., 2007] Biddiscombe, J., Geveci, B., Martin, K., Moreland, K., and Thompson, D. (2007). Time dependent processing in a parallel pipeline architecture. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1376–1383.

[Blelloch et al., 1998] Blelloch, G. E., Plaxton, C. G., Leiserson, C. E., Smith, S. J., Maggs, B. M., and Zagha, M. (1998). An experimental analysis of parallel sorting algorithms. *Theory of Computing Systems*, 31(2):135–167.

[Blondin et al., 2003] Blondin, J. M., Mezzacappa, A., and DeMarino, C. (2003). Stability of standing accretion shocks, with an eye toward core collapse supernovae. *The Astrophysical Journal*, 584:971.

[Brodlie et al., 2004] Brodlie, K., Duce, D., Gallop, J., Sagar, M., Walton, J., and Wood, J. (2004). Visualization in grid computing environments. In *VIS 2004: Proceedings of the conference on Visualization 2004*, pages 155–162, Washington, DC, USA. IEEE Computer Society.

[Cabral et al., 1994] Cabral, B., Cam, N., and Foran, J. (1994). Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *IEEE/ACM Symposium on Volume Visualization 1994*, pages 91–98.

[Card et al., 1999] Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Cignoni et al., 1997] Cignoni, P., Marino, P., Montani, C., Puppo, E., and Scopigno, R. (1997). Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170.

[Critchlow et al., 2000] Critchlow, T., Fidelis, K., Ganesh, M., Musick, R., and Slezak, T. (2000). Datafoundry: Information management for scientific data. *IEEE Transactions on Information Technology in Biomedicine*, 4:52–57.

[Friendly and Denis, 2001] Friendly, M. and Denis, D. (2001). Milestones in the history of thematic cartography, statistical graphics, and data visualization. http://www.math.yorku.ca/SCS/Gallery/milestone/.

[Gao et al., 2005] Gao, J., Huang, J., Johnson, C. R., Atchley, S., and Kohl, J. A. (2005). Distributed Data Management for Large Volume Visualization. In *Proceedings of IEEE Visualization 2005*, pages 183–189.

[Glatter et al., 2008] Glatter, M., Huang, J., Ahern, S., Daniel, J., and Lu, A. (2008). Visualizing temporal patterns in large multivariate data using modified globbing. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1467–1474.

[Glatter et al., 2006] Glatter, M., Mollenhour, C., Huang, J., and Gao, J. (Sept.-Oct. 2006). Scalable data servers for large multivariate volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1291–1298.

[Gosink et al., 2007] Gosink, L., Anderson, J., Bethel, W., and Joy, K. (Nov.-Dec. 2007). Variable interactions in query-driven visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1400–1407.

[Gotsman and Lindenbaum, 1996] Gotsman, C. and Lindenbaum, M. (1996). The metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing*, 5(5):794–797.

[Healey and Enns, 1999] Healey, C. G. and Enns, J. T. (1999). Large datasets at a glance: Combining textures and colors in scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):145–167.

[Heer et al., 2005] Heer, J., Card, S., and Landay, J. (2005). prefuse: A toolkit for interactive information visualization. In *Proceedings of CHI 2005, Portland, OR*, pages 421–430.

[Hoffman et al., 2007] Hoffman, F. M., Covey, C. C., Fung, I. Y., Randerson, J. T., Thornton, P. E., Lee, Y.-H., Rosenbloom, N. A., Stöckli, R. C., Running, S. W., Bernholdt,

D. E., and Williams, D. N. (2007). Results from the carbon-land model intercomparison project (C-LAMP) and availability of the data on the earth system grid (ESG). *Journal of Physics: Conference Series*, 78:012026 (8pp).

[Huang et al., 2007] Huang, J., Liu, H., Beck, M., Gaston, A., Gao, J., and Moore, T. (2007). Visualization viewpoints: Dynamic sharing of large-scale visualization. *IEEE Computer Graphics and Applications*, 27(1):20–25.

[Huang et al., 2000] Huang, J., Mueller, K., Shareef, N., and Crawfis, R. (2000). Fastsplats: Optimized splatting on rectilinear grids. In *IEEE Visualization 2000*, pages 219–227.

[Itoh and Koyamada, 1995] Itoh, T. and Koyamada, K. (1995). Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1:319–327.

[Jänicke et al., 2008] Jänicke, H., Bottinger, M., and Scheuermann, G. (2008). Brushing of attribute clouds for the visualization of multivariate data. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1459–1466.

[Ji et al., 2003] Ji, G., Shen, H.-W., and Wenger, R. (2003). Volume tracking using higher dimensional isocontouring. In *IEEE Visualization 2003*.

[Jolliffe, 2002] Jolliffe, I. T. (2002). *Principal Component Analysis, 2nd edition, ISBN 0-387-95442-2*. Springer.

[Kendall et al., 2008] Kendall, W., Glatter, M., Huang, J., Hoffman, F., and Bernholdt, D. E. (Irvine, CA, May, 2008). Web enabled collaborative climate visualization in the earth system grid. In *Proceedings of International Symposium on Collaborative Technologies and Systems*.

[Kniss et al., 2001a] Kniss, J., Kindlmann, G., and Hansen, C. (2001a). Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *IEEE Visualization 2001*, pages 255–262.

[Kniss et al., 2001b] Kniss, J., McCormick, P., McPherson, A., Ahrens, J., Painter, J., Keahey, A., and Hansen, C. (2001b). Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications*, 21(4):52–61.

[Kniss et al., 2003] Kniss, J., Premoze, S., Ikits, M., Lefohn, A., Hansen, C., and Praun, E. (2003). Gaussian transfer functions for multi-field volume visualization. In *IEEE Visualization 2003*, pages 65–73.

[Kniss et al., 2005] Kniss, J., Van Uitert, R., Stephens, A., Li, G.-S., Tasdizen, T., and Hansen, C. (2005). Statistically quantitative volume visualization. *IEEE Visualization 2005*, pages 287–294.

[Kochevar, 1993] Kochevar, P. (1993). Database management for data visualization. In *Proceedings of the IEEE Visualization '93 Workshop on Database Issues for Data Visualization*, pages 109–117, London, UK. Springer-Verlag.

[Lacroute and Levoy, 1994] Lacroute, P. and Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH 1994*, pages 451–458.

[Leven et al., 2002] Leven, J., Corso, J., Cohen, J., and Kumar, S. (2002). Interactive visualization of unstructured grids using hierarchical 3d textures. In *VVS 2002: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 37–44, Piscataway, NJ, USA. IEEE Press.

[Levoy, 1988] Levoy, M. (1988). Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(5):29–37.

[Li et al., 2003] Li, J., Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. (2003). Parallel netcdf: A high-performance scientific i/o interface. *Supercomputing, 2003 ACM/IEEE Conference*, pages 39–39.

[Liu et al., 2006] Liu, H., Beck, M., and Huang, J. (2006). Dynamic co-scheduling of distributed computation and replication. In *CCGRID 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 592–600, Washington, DC, USA. IEEE Computer Society.

[Livnat et al., 1996] Livnat, Y., Shen, H.-W., and Johnson, C. (1996). A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84.

[Lu et al., 2002] Lu, A., Morris, C. J., Ebert, D. S., Rheingans, P., and Hansen, C. (2002). Non-photorealistic volume rendering using stippling techniques. In *VIS 2002: Proceedings of the conference on Visualization 2002*, pages 211–218, Washington, DC, USA. IEEE Computer Society.

[Lum et al., 2002] Lum, E., Ma, K.-L., and Clyne, J. (2002). A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):286–301.

[McCormick, 1988] McCormick, B. H. (1988). Visualization in scientific computing. *SIGBIO Newsl.*, 10(1):15–21.

[McCormick et al., 2004] McCormick, P., Inman, J., Ahrens, J., Hansen, C., and Roth, G. (2004). Scout: A hardware-accelerated system for quantitatively driven visualization and analysis. In *Proceedings of IEEE Visualization 2004*, pages 171–178.

[Meissner et al., 2000] Meissner, M., Huang, J., Bartz, D., Mueller, K., and Crawfis, R. (2000). A practical evaluation of the four most popular volume rendering algorithms. In *IEEE/ACM Symposium on Volume Visualization 2000*, pages 81–90.

[MPI Forum, 1997] MPI Forum (1997). MPI-2: Extensions to the Message-Passing Interface interface. `http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html`.

[Mueller et al., 1999] Mueller, K., Shareef, N., Huang, J., and Crawfis, R. (1999). High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):116–135.

[Munzner, 2002] Munzner, T. (2002). Guest editor's introduction - information visualization. *Computer Graphics and Applications, IEEE*, 22(1):20–21.

[Musick and Critchlow, 1999] Musick, R. and Critchlow, T. (1999). Practical lessons in supporting large-scale computational science. *SIGMOD Rec.*, 28(4):49–57.

[Parker et al., 1998] Parker, S., Shirley, P., Livnat, Y., Hansen, C., and Sloan, P.-P. (1998). Interactive ray tracing for isosurface rendering. In *VIS 1998: Proceedings of the conference on Visualization 1998*, pages 233–238, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Pascucci and Frank, 2001] Pascucci, V. and Frank, R. (2001). Global static indexing for real-time exploration of very large regular grids. In *ACM/IEEE Conference on Supercomputing (SC 2001)*, pages 2–2.

[RMI, Lawrence Livermore National Laboratory,] RMI, Lawrence Livermore National Laboratory. The ASCI turbulence project. `http://www.llnl.gov/CASC/asciturb`.

[Roettger,] Roettger, S. The volume library. `http://www9.cs.fau.de/Persons/Roettger/library`.

[Rübel et al., 2008] Rübel, O., Prabhat, Wu, K., Childs, H., Meredith, J., Geddes, C. G. R., Cormier-Michel, E., Ahern, S., Weber, G. H., Messmer, P., Hagen, H., Hamann, B., and Bethel, E. W. (2008). High performance multivariate visual data exploration for extremely large data. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA. IEEE Press.

[Samet, 1990] Samet, H. (1990). *The Design and Analysis of Spatial Data Structures, ISBN 0-201-50255-0*. Addison Wesley, New York.

[Shen et al., 1999] Shen, H.-W., Chiang, L.-J., and Ma, K.-L. (1999). A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *IEEE Visualization 1999*, pages 371–377.

[Shen et al., 1996] Shen, H.-W., Hansen, C., Livnat, Y., and Johnson, C. (1996). Isosurfacing in span space with utmost efficiency (ISSUE). In *IEEE Visualization 1996*, pages 287–294.

[Singhal and Zyda, 1999] Singhal, S. and Zyda, M. (1999). *Networked Virtual Environments: Design and Implementation, ISBN 0-201-32557-8*. ACM Press/Addison-Wesley Publishing Co., New York.

[Sirott et al., 2001] Sirott, J., Callahan, J., and Hankin, S. (2001). Inside the live access server. In *17th Intl Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Albuquerque, NM.

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

[Sutton and Hansen, 2000] Sutton, P. M. and Hansen, C. D. (2000). Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):98–107.

[Tiede et al., 1998] Tiede, U., Schiemann, T., and Hoehne, K. H. (1998). High quality rendering of attributed volume data. In *IEEE Visualization 1998*, pages 255–262.

[Tory et al., 2005] Tory, M. K., Potts, S., and Möller, T. (2005). A parallel coordinates style interface for exploratory volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):71–80.

[Trefftz et al., 2003] Trefftz, H., Marsic, I., and Zyda, M. (2003). Handling heterogeneity in networked virtual environments. *Presence: Teleoper. Virtual Environ.*, 12(1):37–51.

[Viegas et al., 2007] Viegas, F., Wattenberg, M., van Ham, F., Kriss, J., and McKeon, M. (2007). Manyeyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128.

[Weiss, 1998] Weiss, M. (1998). *Data Structures and Algorithm Analysis in C++, 2nd edition, ISBN 0-201-36122-1*. Addison-Wesley, New York.

[Wilhelms and Van Gelder, 1992] Wilhelms, J. and Van Gelder, A. (1992). Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227.

[Williams et al., 2007] Williams, D. N., Bernholdt, D. E., Foster, I. T., and Middleton, D. E. (November 2007). The earth system grid center for enabling technologies: Enabling community access to petascale climate datasets. *CTWatch Quarterly*, 3(4).

[Woodring et al., 2003] Woodring, J., Wang, C., and Shen, H.-W. (2003). High dimensional direct rendering of time-varying volumetric data. In *Proceedings of IEEE Visualization 2003*.

[Yu et al., 2004] Yu, H., Ma, K.-L., and Welling, J. (2004). A parallel visualization pipeline for terascale earthquake simulations. In *SC 2004: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 49, Washington, DC, USA. IEEE Computer Society.

Vita

Markus Glatter was born in Herne, Germany on July 14, 1978, and was raised in Balve, North-Rhine Westphalia, Germany. He completed his undergraduate career at the University of Dortmund with a “Diplominformatiker” degree in Computer Science and a minor in Physics in July 2004. He matriculated at The University of Tennessee at Knoxville in fall 2004 and graduated with the Master of Science degree in Computer Science in the fall of 2006. In May 2009, he graduated from The University of Tennessee at Knoxville with the Doctor of Philosophy degree in Computer Science.