OptiStore: An On-Demand Data Processing Middleware for Very Large Scale Interactive Visualization

BY

CHONG ZHANG
B.S., Wuhan University, Wuhan, China, 1997
M.S., Wuhan University, Wuhan, China, 2000

THESIS

Submitted as partial fulfillment of the requirements
of the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2007

Chicago, Illinois

ACKNOWLEDGEMENTS

I would like to acknowledge many people for helping me during my doctoral work. Through the last seven years, I've grown so much personally and in my research. A lot of it was the direct result of the great people I was fortunate enough to learn from and work with.

First of all, I have to thank my advisor, Dr. Jason Leigh, for giving me both the freedom to work on my research interest and the enlightening guidance to help me getting more mature in my research. Throughout my doctoral work he encouraged me to develop independent thinking and analytical skills. He taught me how to conduct and evaluate research work systematically, from which I believe I will benefit greatly in my future career.

I am also very grateful for having an exceptional thesis committee and would like to thank Prof. Thomas DeFanti, Prof. Andrew Johnson, Prof. Luc Renambot, and Ms. Pamela Sydelko for their insightful feedback and continuous assistance during my thesis writing.

I owe a special note of gratitude to Ms. Pamela Sydelko, who hired me as an intern at Argonne National Laboratory, supported my research financially and gave me so many insightful ideas to apply visualization techniques to industrial projects. I should also thank Dr. Boaz Super, my former advisor, who recommended my admission to UIC.

ACKNOWLEDGEMENTS (CONTINUED)

I own many thanks to EVL faculty and staff member, Xi Wang, Dana Plepys, Alan Verlo, Lance Long, Laura Wolf, and Patrick Hallihan, who helped me with both my research and my life in the past seven years; I also wish to thank other people with whom I have the opportunity to work including Eric He, Nicholas Schwarz, Jinghua Ge, Venkat Vishwanath, Rajvikram Singh and CAVERN group. It is such an enjoyable and fun experience I will never forget.

Finally, my deepest thanks to my family and close friends, who have helped me, get through some difficulties in the last few years. I thank my best friends, Fang Liu, Zhi Liu, Chuan He, Yi Li and my soccer team. And to my parents, Lihou Zhang and Jufang Hu, my sister, Hui Zhang, and my beloved, Anne Tong. Without their sacrifices, none of this would have been possible.

CZ

TABLE OF CONTENTS

CHARPTER PAGE

CHAPTER 1 INTRODUCTION 1

1. 1 Background 1
    1.1.1 Large Scale Datasets 4
    1.1.2 Distributed Data Repositories 8
1. 2 The Problem 11
    1.2.1 Scalability with Data Size 11
    1.2.2 Interactivity in Visualization 13
    1.2.3 Flexibility to Access Data Repositories 17
1. 3 The Approach 18
1. 4 Summary of Contributions to the Field 31
1. 5 Document Organization 33

CHAPTER 2 RELATED WORKS 35

2. 1 Multi-Resolution Filtering in Scientific Visualization 35
2. 2 View-dependent Data Processing 36
2. 3 Distributed and Parallel Data Filtering 38
2. 4 Related Systems and OptiStore 39
    2.4.1 Previous Systems for Large Scale Visualization 39
    2.4.2 Comparison of OptiStore and Other Systems 43

CHAPTER 3 OPTISTORE FRAMEWORK 47

3. 1 Framework Overview 47
3. 2 OptiStore Components 51
    3.2.1 Metadata Query Sub-System 51
    3.2.2 Data Filtering Sub-System 53
3. 3 Software Library Components 59
3. 4 Summary 60

CHAPTER 4 DATA PARTITION AND ORGANIZATION 62

4. 1 Overview 62
4. 2 Data Partition and Organization 63
    4.2.1 Data Partition Methods 63
    4.2.2 The Granularity of Blocks 66
    4.2.3 Data Organization 69

TABLE OF CONTENTS (CONTINUED)

CHARPTER PAGE

4. 3 Data Distribution for Load Balancing 71
4. 4 Implementation and Results 75
4. 5 Summary 77

CHAPTER 5 MULTI-RESOLUTION FILTERING 78

5. 1 Multi-Resolution Filtering with Wavelet Transform 78
5. 2 Wavelet Multi-Resolution Filtering for Multi-Dimensional Datasets 85
5. 3 Implementation of Multi-Resolution Filtering in OptiStore 86
5. 4 Summary 89

CHAPTER 6 VIEW-DEPENDENT DATA PROCESSING 90

6. 1 View-dependent Visualization and Data Processing 90
6. 2 Visibility Culling 92
6. 3 Implementation of View-dependent Data Processing 94
6. 4 Summary 99

CHAPTER 7 REAL-TIME DATA PROCESSING AND CACHING 100

7. 1 Furthest Object Replacement (FOR) Algorithm 100
7. 2 Data Processing and Caching With Prediction 109
7. 3 Implementation of Parallel Data Processing with Remote Memory Access 120
7. 4 Summary 126

CHAPTER 8 EXPERIMENTAL STUDY 127

8. 1 Hypotheses & Predictive Models 127
8. 2 Experiments 133
    8.2.1 Experiment hardware, software and data 133
    8.2.2 Experiments 134
8. 3 Results and Discussion 134
8. 4 Summary 140

CHAPTER 9 CONCLUSION AND FUTURE WORK ..... ERROR! BOOKMARK NOT DEFINED.

9. 1 Contributions of OptiStore 141
9. 2 Future Work 143

TABLE OF CONTENTS (CONTINUED)

CHARPTER PAGE

CITED LITERATURE 145

VITA 158

LIST OF TABLES

TABLE PAGE

Table 1-1: The approaches in OptiStore address the motivations in very large scale interactive visualization 31

Table 2-1: Comparison of OptiStore and other systems 46

Table 4-1: The overall process utilizations with different data distribution scheme..... 76

Table 6-1: Finding frustum's bounding plane in modeling space 96

Table 7-1: Comparison of time complexity of the operations between FOR and LRU algorithms 108

Table 8-1: Proposed approaches and their effects on the predictive model..... 132

Table 8-2: Average latency versus data size 136

Table 8-3: Number of filtering processors versus data size 137

Table 8-4: Average access time versus data server cache size..... 139

Table 8-5: Average access time versus number of client processors..... 140

LIST OF FIGURES

FIGURE PAGE

Figure 1-1: Comparison of a modern CPU architecture with one possible configuration of the OptIPuter 4

Figure 1-2: Over terabyte USGS imagery texture 7

Figure 1-3: Processes in Scientific Computing Pipeline 16

Figure 1-4: The fundamental concept of multi-resolution. (a) A complex object is simplified, (b) Creating levels of detail or LODs to reduce the rendering cost of small, distant, or unimportant geometry. Courtesy of David Luebke [Luebke 2001] at University of Virginia©..... 20

Figure 1-5: Traditional distributed visualization pipeline. Model (A) is the standard data flow in visualization pipeline; (B) shows the filtering server, residing on data repositories; in model (C), the renderer and filter work together; and in (D), OptiStore has a dedicated parallel filtering server in the distributed system. 28

Figure 1-6: To meet the requirement of large scale visualization and exploit availability of high-speed network, I propose a dedicated parallel filtering server in the middle between data repositories and visualization applications 31

Figure 3-1: The diagram of OptiStore Architecture 49

Figure 3-3: OptiStore metadata subsystem: a multi-layer client-server model 52

Figure 3-4: An example scene graph bound with datasets from metadata query..... 52

Figure 3-5: OptiStore data filtering subsystem: a multi-layer three-tier model 54

Figure 3-6: OptiStore client modules 55

Figure 3-7: OptiStore data filtering server modules 56

Figure 3-8: OptiStore client library and its dependencies 61

Figure 3-9: OptiStore query server library and its dependencies..... 61

LIST OF FIGURES (CONTINUED)

FIGURE PAGE

Figure 3-10: OptiStore data server library and its dependencies..... 61

Figure 4-1: Data partition in slabs..... 65

Figure 4-2: Data partition in cells..... 66

Figure 4-3: Data partition into cells with finer granularity..... 67

Figure 4-4: Data partition and distribution of a multi-resolution dataset 70

Figure 4-5: One-on-one mapping from a dataset block to a page in memory..... 71

Figure 4-6: Two level three-dimensional scan-line curves, not a space-filling curve..... 72

Figure 4-7: Two examples of space-filling curves..... 73

Figure 4-9: Two level three-dimensional Hilbert curves. 74

Figure 5-1: Wavelet multi-resolution analysis on multi-dimensional datasets. (A) A two dimensional dataset filtered into R level resolutions as a quadtree; (B) A three dimensional dataset filtered into R level resolutions as an octree. 79

Figure 5-2: Filter bank implementation of discrete wavelet transform. (A) H and G are the analysis low-pass/high-pass pair. c[n] and d[n] are the scaling and wavelet coefficients, respectively. (B) It is the corresponding inverse wavelet transform. $\hat{H}$ and $\hat{G}$ are the synthesis low-pass/high-pass pair..... 82

Figure 5-3: The lifting scheme. (A) Typical lifting steps: Split, Predict, and Update; c[n] and d[n] are the scaling and wavelet coefficients, respectively. (B) Typical inverse lifting steps: undo Update, undo Predict, and Merge. 82

Figure 5-4: The lifting scheme is chose in OptiStore 85

Figure 5-5: Three-dimensional wavelet analysis 87

Figure 5-6: Implementation of multi-resolution filtering..... 88

LIST OF FIGURES (CONTINUED)

FIGURE PAGE

Figure 6-1: Four types of visibility culling techniques 93

Figure 6-2: View frustum culling at different resolution levels 94

FIGURE PAGE

Figure 6-3: Bounding boxes of a sphere for view frustum culling 98

Figure 6-4: Back face culling on a sphere 99

Figure 7-1: The least recently used (LRU) replacement algorithm 102

Figure 7-2: The weakness of LRU algorithm with spatial data 103

Figure 7-3: Data structure in the furthest object replacement (FOR) algorithm 105

Figure 7-4: Examples of 2D Manhattan distance and Euclidean distance 106

Figure 7-5: Pseudo-code of FOR algorithm 107

Figure 7-7: Dilation operation for prediction with the same resolution level 112

Figure 7-8: Intra-level predictor 113

Figure 7-9: Prediction on different resolution levels 114

Figure 7-10: An example of Markov chain transition matrix 118

Figure 7-11: Markov chain for data access patterns via transition probabilities 118

Figure 7-12: Integration of predictor with data filtering and caching 119

Figure 7-13: Multi-dimensional data sub-blocks mapped to one-dimensional array in memory on the target node 122

Figure 7-14: Remote memory access on OptiStore data filtering server 123

Figure 8-1: Abstract data access model of OptiStore 128

LIST OF FIGURES (CONTINUED)

FIGURE PAGE

Figure 8-2: Access latency versus data size 136

Figure 8-3: Number of filtering processors versus data size 137

Figure 8-4: Average access latency versus cache size 138

LIST OF ABBREVIATIONS

2D	Two Dimension(al)
3D	Three Dimension(al)
AABB	Axis Aligned Bounding Boxes
API	Application Program Interface
CERN	European Institute for Nuclear Research
DCT	Discreet Cosine Transform
DWT	Discrete Wavelet Transform
EOSDIS	Earth Observing System Data and Information System
FOR	Furthest Object Replacement
GPU	Graphics Processing Unit
GPGPU	General-Purpose computing on GPU
LAN	Local Area Network
LOD	Level-Of-Details
LRU	Least Recently Used
MIMD	Multiple Instruction stream, Multiple Data stream
MPI	Message Passing Interface
NASA	National Aeronautics and Space Administration
NCAR	National Center for Atmospheric Research
OBB	Oriented Bounding Boxes
OO	Object-Oriented
PB	PetaByte(s)
RAM	Random Access Memory
RMA	Remote Memory Access
RTT	Round-Trip Time
SPMD	Single Process Multiple Data stream

LIST OF ABBREVIATIONS (CONTINUED)

TB	TeraByte(s)
USGS	United States Geological Survey
UUID	Universally Unique Identifier
WAN	Wide Area Network

SUMMARY

OptiStore is an on-demand data processing middleware for extremely large scale interactive visualization applications. It aims to develop a data processing service system that bridges the gap between the size of the very large datasets and the performance of interactive high-speed parallel visualization applications in the context of OptIPuter. Compared with the predominant strategy by preprocessing data on data repository before visualization, OptiStore processes the data on-demand and interactively so as to minimize the need to manage extraneous pre-processed copies of the data that will become a major problem as scientists continue to amass vast amounts of data.

In the architecture of OptIPuter, the distributed components, such as rendering clusters, data storage clusters and computation clusters are inter-connected by wide area optical networks. Hence the data that the visualization cluster demands at one site may be stored at other sites on different remote data storage clusters. The goals of OptiStore are to help the visualization users to access large amount of data (from terabytes to petabytes) on remote locations, query them on the distributed servers, transfer them among OptIPuter components, and filter and transform them from one data model to another in near real-time. Furthermore, OptiStore is an extensible middleware framework, into which more new data structures and filters can be integrated.

SUMMARY (CONTINUED)

In order to address the issues of scalability of data size, interactivity in data exploration and flexibility of data filter deployment, I proposed the following techniques in this dissertation: load-balancing data partition and organization, multi-resolution analysis, view-dependent data selection, runtime data preprocessing and dedicated parallel data filtering.

To achieve high overall utilization and reduce latency cost, we developed a load-balancing data partition and organization mechanism. To ensure the scalability with the size of the datasets, the multi-resolution analysis and view-dependent culling were applied for processing the necessary data in the view of the visualization application. To take advantage of the increasing network bandwidth, we decoupled the data filter from visualization applications and data repository servers by transferring the bulk of data through the high-speed network infrastructure. By separating the data filter services from other processes in the distributed visualization pipeline, the data providers can maintain and share the data repository with less effort, and users can explore more large datasets available on the LambdaGrid and deploy their own filters flexibly. We developed a novel caching algorithm and a prediction model for prefetching and preprocessing to minimize the data access latency to meet the requirement of interactive visualization.

CHAPTER 1 INTRODUCTION

1. 1 Background

Defined in the milestone paper [McCormick 1987], *visualization is a tool both for interpreting image data fed into a computer, and for generating images from complex multi-dimensional datasets*. Nowadays, the data size on both ends of visualization tools is increasing dramatically fast: the resolution of image data is increasing higher and higher due to the emergence of the new technologies of very high resolution displays and cluster-driven tiled displays [Ni 2006], and it is easy for the multi-dimensional datasets to reach the size of terabytes and even petabytes because scientists gain access to ever increasing computational resources [Hey 2003]. It is always an extreme challenge to interactively explore and visualize very large multi-dimensional datasets on very high resolution display devices. Furthermore, as the emergence of a global LambdaGrid for e-Science and other mature high-speed networking technology, the collaboration between scientists is more practical and popular [Freitag 2001; Leigh 2003; McCormick 2005; Saltz 2003]. Thus it is becoming urgent and significant for scientists to have some visualization applications that can help them to analyze and explore those extremely large datasets on remote data repository sites.

In 2002, DeFanti and Leigh proposed an experimental OptIPuter architecture to cope with this challenge [DeFanti 2002; Leigh 2003]. The OptIPuter [Smarr 2003] is a National Science Foundation funded project between the University of Illinois at Chicago, and the University of California, San Diego, to interconnect distributed storage, computing and visualization resources using a backplane constructed from a grid of deterministic high speed networks. In partnership with the Scripps Institute, US Geological Survey's EROS (Earth Resources Observation System) Data Center, and the Biomedical Informatics Research Network, the specific application goal of the project is to develop advanced computing systems to support collaborative data exploration in the Geosciences and the Neurosciences.

In this architecture, the network becomes the backplane and the clusters of computing systems become the computer peripherals. For example, a cluster of computers with high end graphics cards is considered as a single giant graphics card; and a cluster with terabytes of parallel file storage is considered as a single giant disk drive. Driven by the parallel computer clusters, a large dataset can be streamed from the distributed parallel I/O clusters to the high end visualization clusters.

Figure 1-1 (A) depicts a typical architecture for a modern day PC. Highlighted in yellow are the caches that are a routine part of the components of the architecture. For example, the graphics card has onboard fast graphics RAM, the CPU has L1/L2/L3 caches, and so on. Data from the disk are transferred to the CPU via the PCI bus, whereas data from

the CPU is transferred to the graphics card via PCI-Express bus. Figure 1-1 (B) shows one possible configuration of the OptIPuter mimicking the standard PC architecture except using clusters of computers, optical switches, and multi-gigabit network connections. A similar configuration to this was used for iGrid [Zhang 2003], although this particular layout is our present configuration. A LambdaNode is a high-performance cluster connected with another one using gigabit or 10 gigabit network interface adapters. Illustrated are three classes of LambdaNodes. The LambdaDataNode is primarily a cluster with large RAID-ed disks. The LambdaComputeNode is a cluster with large amounts of physical memory and multiple CPUs. The LambdaVisualizationNode is a cluster with high-end commodity graphics cards (such as the nVidia Geforce 4 Ti). All network links are linked with TransLight [DeFanti 2003].

OptIPuter architecture has already included some novel applications and middleware such as the high-speed network protocol – LambdaStream [Xiong 2005], the scalable graphics middleware – SAGE [Renambot 2004], the scalable visualization applications – JuxtaView [Krishnaprasad 2004] and Vol-a-Tile [Schwarz 2004b]. These applications not only make the distributed high-resolution rendering and display possible and feasible but also require other middleware to aid the end users to visualize and explore very large datasets on remote sites. When to implement the visualization applications based on the model of OptIPuter, the developers or users may encounter some difficulties: (1) how to access and transform extremely large datasets in remote repositories; (2) how to

reduce long latency in interaction when visualizing the large dataset; and (3) how to offload the overhead work for the distributed dataset management.

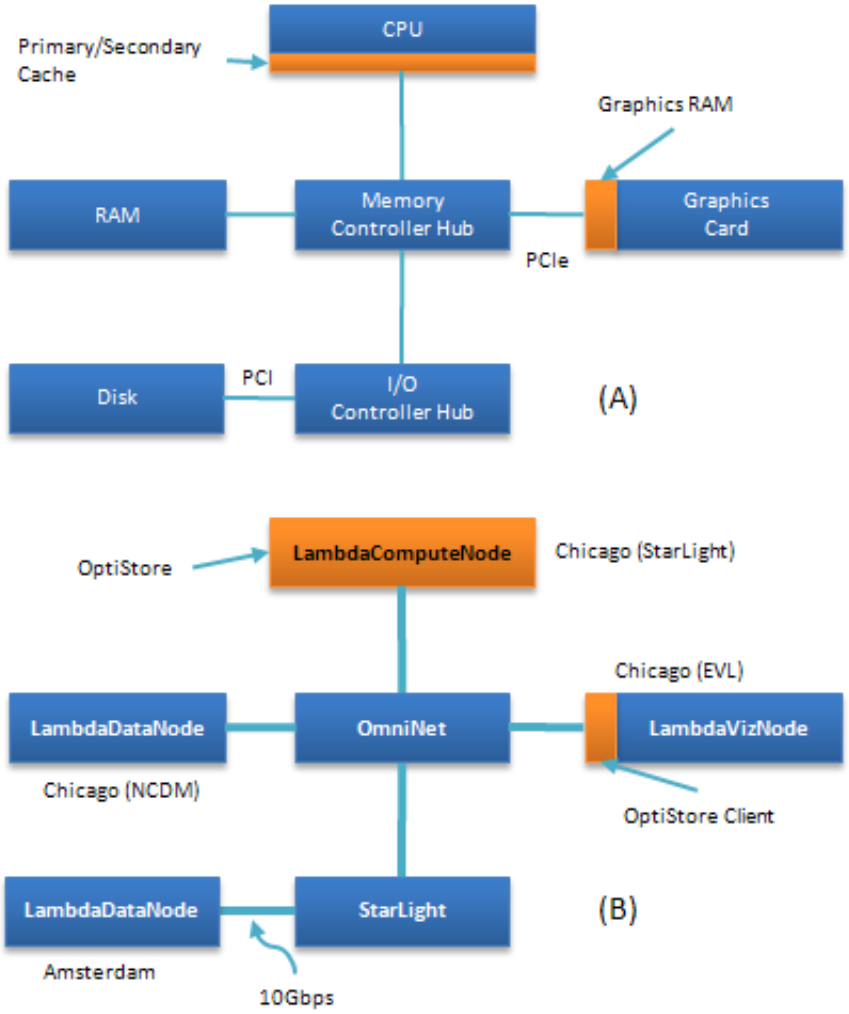


Figure 1-1: Comparison of a modern CPU architecture with one possible configuration of the OptIPuter

1.1.1 Large Scale Datasets

This is the age of information explosion with the exponent increase of data size and data complexity. In the field of science, the state-of-the-art simulations of physical systems

can generate terabytes to petabytes of time-varying data where a single time step can contain more than a gigabyte of data per variable [Kniss 2001]. High-accurate sensors (i.e. National Aeronautics and Space Administration's (NASA) shuttle radars or Computed tomography scanners and Magnetic Resonance Imaging scanners in biomedicine) can acquire very high-resolution images.

There are some concrete examples that demonstrate the spectacular growth of scientific data generation:

- For instance, in geosciences, the NASA's Blue Marble [Stockli 2005] dataset has eight files for each month out of 12 months. Each file is a block of the whole earth image and has the size of 21600 x 21600 x 3 (width x height x RGB) in the full resolution. The total dataset size is around 125GB. United States Geographical Survey's (USGS) aerial images are even larger. For example, the 0.3 meter-resolution aerial imagery dataset of Chicago metropolis-area, consists of 67 x 60 = 5092 images, where each image is of 5000 by 5000 pixels and 3 bytes apiece. The total size of the dataset is about 356GB [Barclay 1998; Krishnaprasad 2004]. Similarly, the aerial imagery dataset of San Diego is about 240GB. If a visualization application attempts to render the Blue Marble image with the USGS aerial images of some big cities (shown as Figure 1-2), the size of the data to access and process will exceed several terabytes eventually.

- Another example would involve the visual human project datasets. "*The Visible Human Project datasets are designed to serve as a common reference point for the study of human anatomy, as a set of common public-domain data for testing medical imaging algorithms, and as a test-bed and model for the construction of image libraries that can be accessed through networks*", Ackerman declares [Ackerman 1998]. The original visible woman dataset, just one of the Visible Human Project datasets, consists of 5189 axial slices, each with 2048x1216 pixels, 3 bytes per pixel. The total size for the dataset is roughly 38.8GB, which presents difficulties in visualizing the entire or an arbitrary portion of the dataset even at this coarse resolution. Since the datasets are three dimensional, with the increasing acquisition resolution, the size of the datasets will scale cubically.

- Similarly, the simulation in seismology usually generates very large scale data. In Schwarz's paper [Schwarz 2004a], the researchers gather and interpolate Bolivian earthquake simulation onto a 512^3 regular grid at 16-bit resolution takes about 12 minutes per frame of 33.8 hours. The total size of just one simulation data approximates 43GB. If the researchers double the size of the grid and frequency of the sampling time, then they can gather the size of the dataset will increase to 678GB according to the order of two.

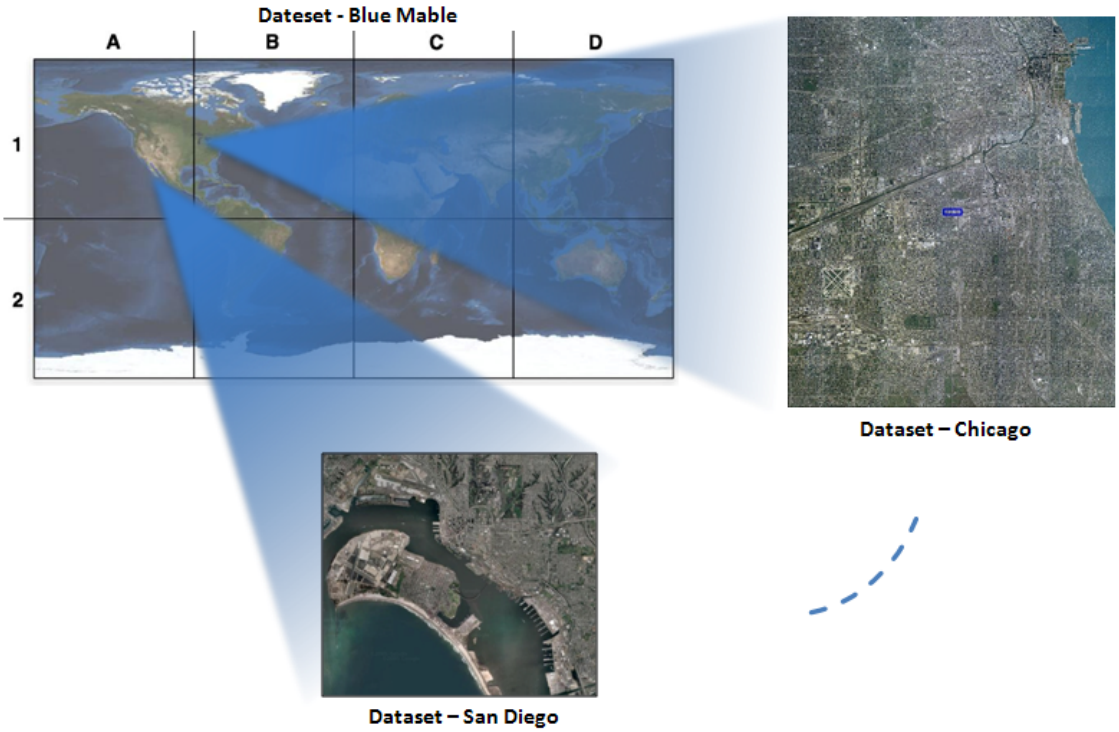


Figure 1-2: Over terabyte USGS imagery texture

Furthermore, in various scientific fields, including astronomy, bioinformatics, environment science, particle physics, medicine and health, social science etc. [Hey 2003], the era of petabyte scale data has already arrived:

- The National Center for Atmospheric Research's (NCAR) Mass Storage System exceeded one petabyte of climate data in 2002 [Lester 2003], and had added a second petabytes within just two years [Bevirt 2004].

- In 2005, NASA's Earth Observing System Data and Information System (EOSDIS) had already held over three petabytes of data in earth science data in a geographically distributed mass storage system [Behnke 2005].

- To be launched in May 2008, Large Hadron Collider (LHC), an accelerator currently under construction at the European Organization for Nuclear Research (CERN) will generate around 15 petabytes of useful data per year; these data that will be analyzed by a worldwide scientific collaboration of physicists [Jones 2006].

The concrete examples and tendency in science demonstrate that the scientific datasets are currently growing extremely large in many research fields. While the size of various datasets is still increasing, how to access and filter the over-terabyte and even petabyte datasets while visualizing them is an urgent research problem.

1.1.2 Distributed Data Repositories

LambdaGrids are large and diverse. Normally, it is necessary and unavoidable to have the data repositories distributed over the wide-area-network. The reason lies under the following five facts:

1. In the OptIPuter model, the data repositories are distributed. From the description in previous section, OptIPuter architecture treats the distributed clusters as basic functional components in computer architectures. And all of the components are inter-connected with each other by optical network. The dedicated optical network makes the high-speed data movement possible.

2. The datasets may be studied in different knowledge domains. Different scientists in different fields may have different interpretation of the same dataset. For example, as to the height field datasets from USGS, the geologist may be interested in visualizing the physiognomy by generating the terrain, while the seismologist just need to simple geometry of the terrain to demonstrate the locations of earthquakes. It is waste of resource to keep one copy locally of the data for different users.

3. The datasets may obtained by different data acquisition methods. Obviously, datasets are acquired by different means and tools, even in the same research field, i.e. CT, MRI and Ultrasound scanners. The dataset acquired by one method is sometimes formatted in one specific format and stored into a dedicated data repository associated with this scanning device.

4. The datasets may belong to different research institutes. Even though the datasets are within the same domain and acquired by the same means, sometimes they are copyrighted by different research institutes or companies. To keep the data repositories intact is sometimes necessary under such circumstances.

5. Other factors also determine the physical location of data repositories. There are a lot of factors that influent the companies or institutes' decision to choose the data repository locations, such legacy, cost and etc. For example [Kirkpatrick

> 2006], Microsoft® locates its next data center not far from the Grand Coulee Dam, the third largest hydroelectric dam in the world, because the facilities of the data center will be so energy-intensive that the primary cost of operating them will probably be electricity.

However, interactive visualization applications and data intensive applications have a wide variety of needs and data-access patterns. The distributed data storage systems vary from one to another as well. It is, therefore, critical to allow applications the flexibility to construct application-specific interfaces, data access patterns, data distribution layouts, and so forth. Studies of data-intensive scientific applications demonstrate the performance benefits of using application-level interfaces that enable advanced parallel-I/O techniques like collective I/O, pre-fetching, and data sieving. A system based on a flexible and powerful low-level interface encourages the development of application-specific libraries that provide the interface and features that benefit applications. For example, tailoring the pre-fetching and caching policies to match the application's access patterns can reduce latency and avoid unnecessary data requests, and matching data-distribution policies to the application's access patterns can optimize parallel access to distributed disks.

The importance of providing and managing distributed access to storage is that laboratory instrumentation environments, hospitals, etc., are frequently not the best place to maintain a large-scale digital storage system. Such systems can have considerable

economy of scale in operational aspects, and an affordable, easily accessible; high-bandwidth network can provide location independence for such systems.

1. 2 The Problem

From the introduction in the previous section, it can be noticed that the scalability of the system with the dataset size, the interactivity in visual data exploration and flexibility to access the distributed data storage become three of the main focuses and challenges in this architecture of OptIPuter as well as in current scientific and information visualization research. The goal of OptiStore is to address these concerns.

1.2.1 Scalability with Data Size

In the field of visualization research, the scalability is one of most highly significant issues. In Christ Johnson's survey about top scientific visualization research problems [Johnson 2004], thanks to the grid technology and distributed computing, it has become a frontier research topic to develop scalable algorithms that can take the advantage of distributed resource; in Chaomei Chen's paper about ten top information visualization research problems [Chen 2005], to explore the scalability in this area is much more immature and desirable compared to the scientific visualization.

The main reason to develop scalable systems in visualization research is that the data size is growing too fast. Since the size of the datasets has passed over terabyte level and is reaching petabyte level, only scalable systems can catch up the pace of the data growth. Scalability is an important concept in parallelism research. It indicates that the

system can increase the throughput under an increase of the load when resources are added [Gray 2003]. In the context of OptIPuter, the scalability of OptiStore is manly about the capability to process and serve increasingly large data load and request by adding new computation nodes and network bandwidth. It is in a scale-out perspective [Agerwala 2006].

Due to maturity of the high-resolution and scalable display system technology [Leigh 2007; Ni 2006], the data management system should take care of not only the scalable size of input raw data but also the scalable size of the output data (for example, the resolution of rendered images). Traditionally, if the display resolution is low, many parallel sort-last visualization algorithms can solve the scalability in large data processing – partitioning the data in small chunks and then composing the final output rendered images [Moreland 2001]; on the contrary, if the data size is limited, many parallel sort-first visualization algorithms can render the data on high-definition or tiled display devices – every computer node behind each display loading the replicated whole dataset and rendering visible geometries in the corresponding area [Bethel 2003; Corrêa 2002].

Li presents an system RIVA that can fulfill the scalability on both ends [Li 2002a]. But their datasets have to be processed and compressed in advance to achieve the data scalability. Furthermore, the dataset is two-dimensional data, to render which is much less complicated than to render higher dimensional data.

Other researchers have exploited view-dependent continuous Level-Of-Details (LOD) algorithms in geometry simplification for a while [Levenberg 2002; Luebke 2002]. Guthe et al. presents an algorithm of wavelet multi-resolution analysis of volume data with hierarchical view-dependent cell selection for volume rendering [Guthe 2002]. Even though, most of their algorithms work under the low-bandwidth network environment and the data need to be processed and compressed in advanced, the extension of their algorithms into the parallel and distributed computation environment can benefit the data size scalability in OptiStore. However, the main obstacle to exploit this strategy is that in these algorisms, the datasets have to be processed or compress in advanced so that this strategy can result in a problem of the following issues: interactivity in visualization and flexibility to access distributed data repositories.

1.2.2 Interactivity in Visualization

Interaction is often referred to human-computer interaction which is one of the most basic features of information visualization and scientific visualization. Interactivity is a requirement that facilitates the interrogation of the data or the observed objects. Munzner [Munzner 2000] summed up three aspects of the interaction:

Navigation: interactive navigation consists of changing either the viewpoint or the position of an object in a scene.

Making choices: interactivity is also common in non-navigational settings, for example through radio buttons on a control panel or menu choices that affect the display.

Animated transitions: viewers have a much easier time retaining their mental model of an object if changes to its structure or its position are shown as smooth transitions instead of discrete jumps.

Interactivity is the great challenge and opportunity of computer-based visualization. Straightforward navigation and manipulation of the data which is represented as graphical objects on the screen can help the scientists to detect the intrinsic pattern of the data and understand the context meaning of the whole datasets.

However, scientists are faced with a problem that as their simulations grow to sizes where interesting features and structures can be resolved, the features themselves become too small (relative to the size of the data) to find, and the structures too large to visualize. Interactive navigation and exploration of these datasets is essential, and small features can only be properly understood is shown in the context of larger structures, showing both large scale structures and small scale features in the same visualization is essential. Nevertheless, the data size prevents efficient rendering of even single frames, let alone multiple frames per a second that is required for interactive exploration [LaMar 2003]. Furthermore, due to the long distance remote data repositories over wide-area LambdaGrid, the high latency of

the interaction between data query and data visualization is becoming obvious; the large delay of the response is sometimes unbearable to the users.

More than being referred to the one in human-computer interface when exploring the data, interactivity can also be considered as interaction with other processes in scientific computing pipeline. Conventionally, visualization is most often seen as a post processing step in the scientific computing pipeline (geometric modeling -> simulation -> visualization), illustrated as Figure 1-3 [Johnson 2004]. However, scientists now require more from visualization than a set of results and a tidy showcase in which to display them. The 1987 National Science Foundation Visualization in Scientific Computing workshop report poses the problem in these terms [McCormick 1987]:

> *Scientists not only want to analyze data that results from super-computations; they also want to interpret what is happening to the data during super-computations. Researchers want to steer calculations in close-to-real-time; they want to be able to change parameters, resolution or representation, and see the effects. They want to drive the scientific discovery process; they want to interact with their data.*
>
> *The most common mode of visualization today at national supercomputer centers is batch. Batch processing defines a sequential process: compute, generate images and plots, and then record on paper, videotape or film.*

Interactive visual computing is a process whereby scientists communicating with data by manipulating its visual representation during processing. The more sophisticated process of navigation allows scientists to steer, or dynamically modify computations while they are occurring. These processes are invaluable tools for scientific discovery.

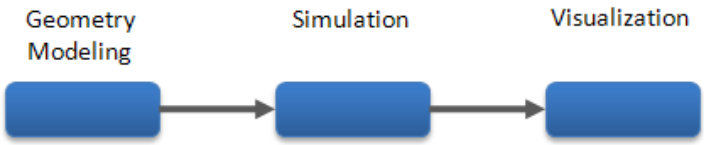


Figure 1-3: Processes in Scientific Computing Pipeline

Thus, from the latter interpretation of the interactivity, timeliness turns out to be a very important issue for visualization applications when the applications try to access and visualize very large datasets. The timeliness in visualization application is the requirement that the users can visualize the datasets at the right time. In practical applications [Bryson 1996; Singh 2006], that means the programs can allow the users to view and explore the latest experiment/simulation results as soon as possible. For examples, the applications should let the users view the latest physical simulation results so as to adjust the experiment models, to see the latest climate forecasting data rather than “aft-casting” data; to detect the arbitrage before the financial exchanges are closed.

However, since the dataset volume is very huge, it is normal to visualize the raw data after batch processing the data. Timeliness becomes a critical issue in this scenario as

the scheduled data processing is not suitable for the interactive visual computing. To reduce the latency in the interaction during visualization as well as in the overall visual computing pipeline, especially in visualizing and processing very large distributed datasets is a critical problem and one of the utmost goals of OptiStore.

1.2.3 Flexibility to Access Data Repositories

In the background introduction, it is explained that the distributed data repositories causes a lot of difficulties in such data intensive applications. As the distributed data repositories become more and more common, two trends should be noticed:

1. The data providers have huge datasets. After data acquisition, the data providers intend to store their data in their favor format. Even though many research institutes plan to unify the all the data formats into one universal data format [Ellis 2004; Folk 1999; Rew 1990], it still is too difficult to unify the formats. Data-providers are not usually willing to incorporate one-off data-filtering needs of their users.

2. Cheap storage and networking is making it possible for the first time to efficiently publish and move vast amounts of raw data. This explosive availability of data is enabling users to come up with imaginative ways to interpret it. Therefore users now have the means to preprocess/filter the data for themselves rather than use the preprocessing capabilities given by the data providers. Also users of the data don't really want to be in the business of

maintaining those large data-sets. They only want to mine it for useful information.

Currently, either the users require the data providers to transfer the datasets on the server side or the user download a small subset of datasets to their local disk. In the former case, the data-provider may not be able to anticipate all the data-processing/filtering needs of all their users. In the latter, evidently, it does not satisfy the need of the large scale visualization applications.

It is an essential issue to provide flexibility to manage the data repositories for the data providers and flexibility to access/process distributed large datasets for the users.

1. 3 The Approach

To address the problems driven by the requirement of scalability, interactivity, and flexibility, I proposed and designed OptiStore - an on-demand distributed data management system of various very large datasets for scientific visualization and information visualization. These large datasets normally includes structured raster image textures, height field or elevation terrain images, graphics models, time-varying extension of these data models and higher dimension regular grid data. Usually, the data is stored on different locations and maybe affiliated with different organizations and the data are organized by the local server using various data management systems, such as relation databases, object-oriented databases, or data repositories as simple as indexed file systems (in some cases,

the raster image dataset and time-varying volume data exist as sets of files). OptiStore should provide an application interface to query the distributed data repositories, access the spatial data, and maintain the systems. The visualization tools don't need to care about where the data is, in what format the data is organized, how to access the data, which part of the data it should crop from data files and etc. They simply request the data from OptiStore by giving geometry coordinates, and then OptiStore should handle the rest of the work and feed back the data. Moreover, OptiStore also should have the capability of data filtering and data processing so that it can provide convenient and fast data access and assist the user to discover inner relation within the original crude data.

The proposed OptiStore middleware system includes the functionalities as follows:

1. *Multi-resolution analysis for multi-dimensional datasets*

In 1996, Ben Shneiderman introduced the visual-information-seeking mantra into the visualization theory: *overview first, zoom and filter, then details-on-demand*. [Shneiderman 1996]. Most of the scientific visualization user interfaces are employed in this way. The overview-zoom-filter-detail is the natural manner for the users to explore the data. Essentially, it is the way to visualize the data at different resolutions.

Furthermore, as the previous chapter explains, the very large datasets are not able to be load into neither the computer's main memory nor the memory on the graphics card at once. It is reasonable for researchers to resort to down-sampling / sub-sampling the dataset. Figure 1-4 shows that under some circumstances, the full resolution of the dataset

is not always necessary. In this example, when the data is far from the viewpoint of the user, higher resolution of the geometry data does not contribute more that the coarser dataset onto the display.

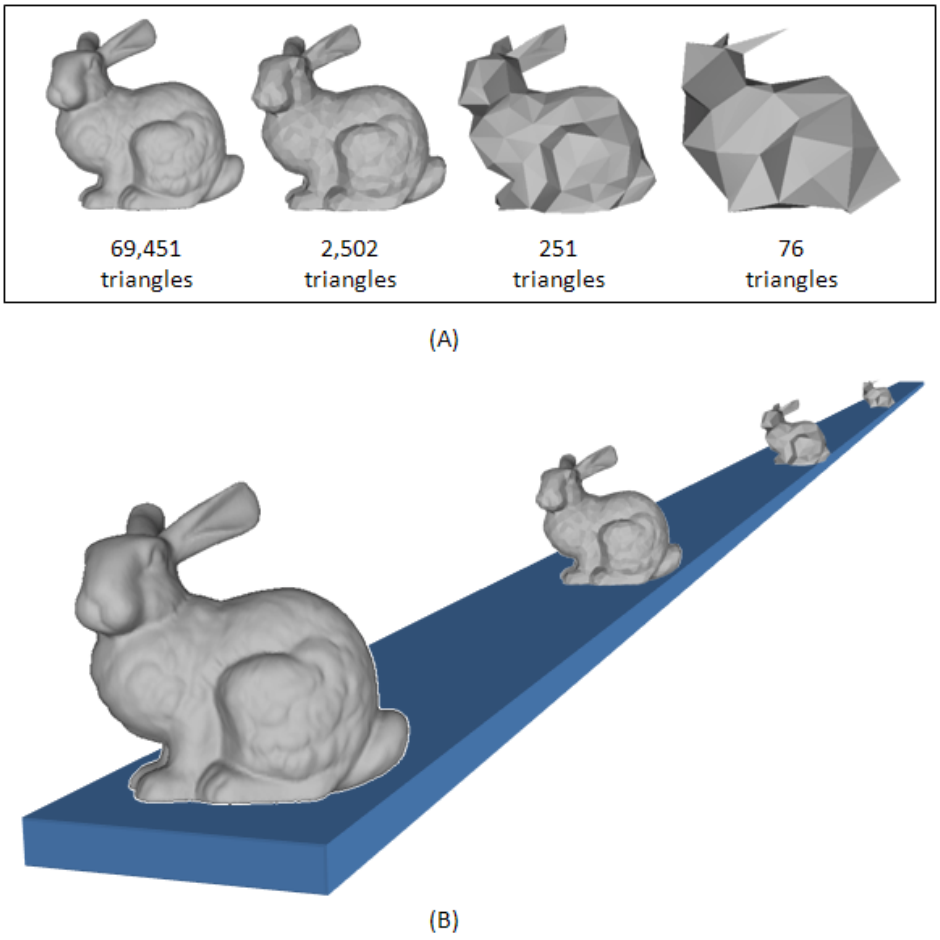


Figure 1-4: The fundamental concept of multi-resolution. (a) A complex object is simplified, (b) Creating levels of detail or LODs to reduce the rendering cost of small, distant, or unimportant geometry. Courtesy of David Luebke [Luebke 2001] at University of Virginia©.

Last but not the least, in many large scale or distributed visualization applications, latency hiding techniques have already become a major approach to reduce user perceived latency. In a nutshell, these techniques aim to trade data fidelity for prompt response time, for instance, by intelligently rendering lower-resolution of the requested data while the data is still being transmitted if the user changes the view-point rapidly. In this case, user-perceived latency can be greatly reduced with the same actual latency while the loss of data fidelity is negligible due to high-frequency user-interaction. In order to fulfill these techniques, multi-resolution filtering or level-of-detail processing on the datasets is unavoidable.

In this research, OptiStore mainly utilized multi-resolution model for regular grid data. OptiStore has two methods: 1) if the datasets are too large to load, it will sub-sample the data while loading the data; 2) to keep as much fidelity of the data as possible, it will utilize wavelet analysis to generate coarser level of resolutions.

2. View-dependent data processing to accelerating data access

The multi-resolution methods are useful for fast navigation through the dataset, but maximum resolution is limited by the memory size and bandwidth of the network. It is not scalable to render the highest resolution of the dataset at once. In contrast, visualization applications often work with full-resolution datasets either as the computation proceeds or as a post-processing step [Haimes 1994]. In the graphics cluster, the data is distributed across the processors, and derived visualization entities such as pixels, voxels, isosurfaces or

streamlines are computed in parallel and communicated to the graphics cluster for display. The advantage of this technique is that no information is lost in a data reduction process; however, for scientists who run their parallel visualization applications at remote sites, there are two potential drawbacks to using this method for interactive visualization in the OptIPuter model. First, the remote disks are accessed through network and then disk I/O, which makes it difficult to predict when the "interactive" job will run. Second, these techniques are useful for interactive exploration of the dataset only if there is sufficient network bandwidth between the graphics cluster and the disk cluster. Both of them limit the scalability in data access and processing, and lead to some high latency during the interactive exploration.

This difficulty in higher resolution data access can be solved by the view-dependent data selection. Researchers [Cohen-Or 2003; De Floriani 2000; El-Sana 1999; Hoppe 1997; Levenberg 2002] exploited this fields for geometry simplification, walkthrough in very large scenes, visualization of large scale objects and etc. Durand [Durand 1999] gave a comprehensive survey in the visibility study. And Hoppe [Hoppe 1997] summarized the reasons why the view-independent rendering increases the throughput of data access. Similarly, the cluster level visualization application will also have three difficulties if the data access is view-independent:

1. Many geometry primitives of the corresponding represented dataset may lie outside the view frustum, and thus do not contribute to the rendered image.

2. Similarly, it is often unnecessary to render those primitives oriented away from the viewer, and such primitives are usually culled using a “back-facing” test, but again at a cost.

3. Within the view frustum, some regions of the model may lie much closer to the viewer than others. View-independent multi-resolution data processing fails to provide the appropriate level of detail over the entire dataset.

In OptiStore, the data space is partitioned into quad-tree or octree and their extension in higher dimensional spaces. The view matrix is passed from the visualization application to OptiStore client. The view-culling is applied to the bounding box of each node in the tree.

3. Runtime data processing

If the data filter is designed to support the flexibility of data management without preprocessing the datasets, the data filtering has to run on the fly during visualization. However, if the datasets are too large for the computation cluster, it will be extremely difficult to catch up the interactive frame-rate on the visualization side. Even though the techniques of level-of-details and view culling help to accelerate data access and filtering, when the computation of data filtering is too complicated, the response time from the user

request to receipt of filtered data will still be high. Thus before the arrival of the next request, the data processing and filtering in advance may reduce the response time, like the mechanism of cache prefetching.

In cache prefetching algorithms, the well-designed predictor can improve the hit ratio much higher. Like these algorithms [Doshi 2003; Johnston 1997; Rhodes 2005], the data filter of OptiStore attempts to predict the next user's operation based on the prior history and start a new filtering process before the next request arrives.

Unlike the cache prefetching, the preprocessing predictor considers not only the data of the next request but also the operation of the next request. For example, a two-dimensional caching algorithm can fetch larger region of data than the one in the current, just like normal caching; but since it has no knowledge about the next operation on this data, the original data will not be processed until the request comes so that the delay of the response is still high.

In OptiStore, a simple and effective page replacement algorithm was proposed. And together with the level of resolution, the view information and the movement, the filter operation constitutes a state vector. A Markov chain model [Doshi 2003] has been applied to predict new operation and dataset.

4. Distributed high-performance multi-dimensional data filtering

McCormick and Ahrens [McCormick 2005] conclude that there are four fundamental techniques that can be used to solve the large-data visualization problem. These techniques can also be applied to very large scale distributed visualization applications:

1. Data streaming (data partition along time). It is one of most common solutions when the incoming data is possible to partition and the output result can be combined later with intermediate temporary results. It is feasible to partition the data into small parts, process and render those parts one by one, store the temporary result of each part, and combine all the temporary results into the final result. Obviously, it scarifies execution time for data size that exceeds computation resources (hardware and/or software). It should be employed as a last resort for interactive visualization applications.

2. Data parallelism, data partition based on space. Data parallelism is like Single Instruction Multiple Data (SIMD) model. With data parallelism, the data is partitioned into small parts and the same procedure of an application executes in parallel with those parts of data simultaneously. It is like data stream technique to partition data into small parts. But unlike data stream technique, it assigned the subsets of data to different processors or systems, instead to different execution time slots. Data parallelism can be implemented as an extension of the data-decomposition technique.

3. Task parallelism, procedure partition based on processors. With task parallelism, the incoming data can be same or different but the task processes are different and run with the incoming data on a number of processors simultaneously. The key advantage of this technique is that it enables multiple portions of a visualization task to be executed in parallel. The main disadvantage of this technique is that the number of independent tasks that can be identified, as well as the number of CPUs available, limits the maximum amount of parallelism. In parallel computing, one common application of this technique is that the master processor and the slave processors are assigned different tasks.

4. Pipeline parallelism, procedure partition based on time. Since the output of a procedure in a pipeline is the incoming data of the next procedure. Thus pipeline parallelism occurs when a number of modules in an application execute in parallel but on different data (thus distinguishing this process from task parallelism). In distributed visualization environment, as the procedures of the pipeline runs on separate systems, this technique can be applied to achieve high performance and overcome the limitation of data size.

Because the datasets are huge and distributed on remote sites with different procedures in visualization pipeline, the latency becomes crucial for the interactive visualization tools when they access the datasets. To overcome the high latency, we tried to combine the all of the last three techniques to speed up the data processing for distributed

visualization. Data parallelism and task parallelism are employed to OptiStore as many other parallel visualization application. But we applied a much more parallel pipelined paradigm than other applications.

Usually, the data flow in visualization pipeline is illustrated as Figure 1-5 (A). The visualization pipeline consists of data source, data filtering, rendering and display. In most cases, it is impossible to visualize very large datasets interactively without filtering the data before rendering [Pavlakos 2005]. There exist several reasons, listed below:

1. Conversion of the original data into a multi-resolution or hierarchical form can allow interactive data exploration at different level of details efficiently.

2. Reorganization of the original data forms on data repositories into those suitable for computation hardware and graphics hardware. For example, partition the original large data files into power-of-two blocks as image texture for graphics card.

3. Re-sampling the original data onto a different type of grid. For instance, it is useful to sample unstructured or irregular data into structure regular grids.

4. Many other operations are also important, such sub-setting, down-sampling, extraction, aggregation, and so on.

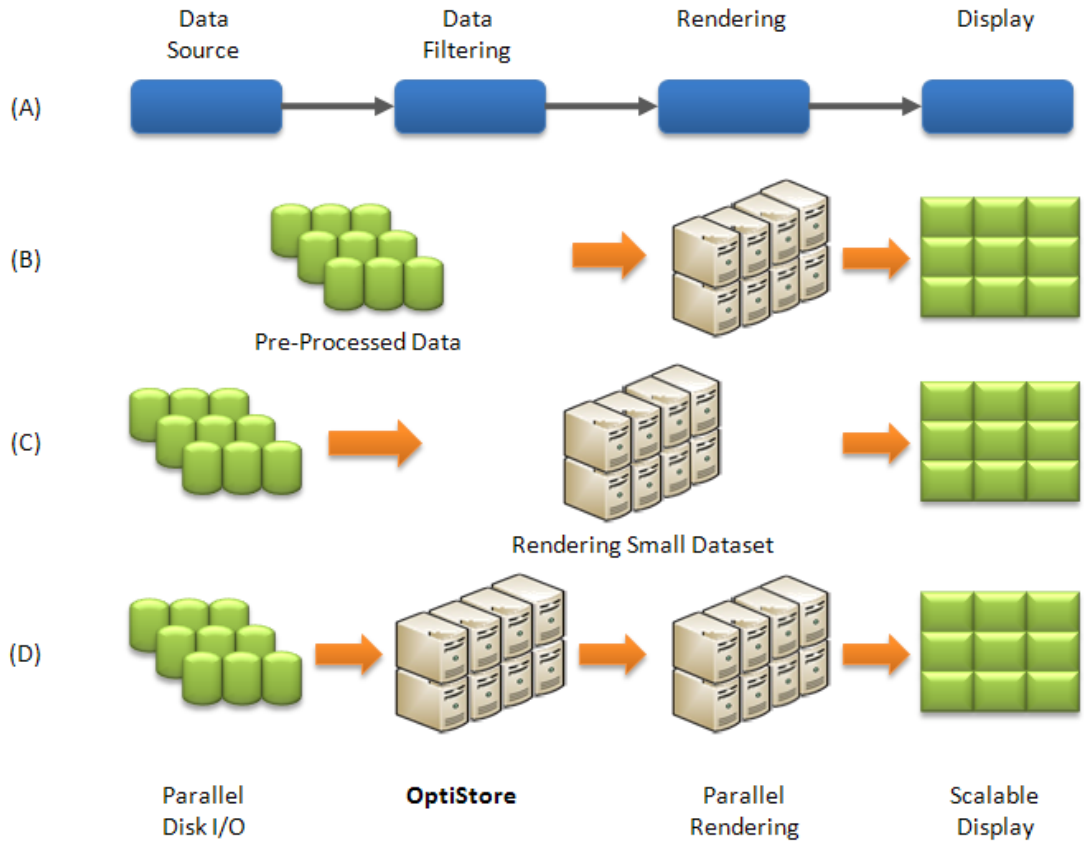


Figure 1-5: Traditional distributed visualization pipeline. Model (A) is the standard data flow in visualization pipeline; (B) shows the filtering server, residing on data repositories; in model (C), the renderer and filter work together; and in (D), OptiStore has a dedicated parallel filtering server in the distributed system.

Current predominant parallel pipelined strategies for large scale data visualization belong to either of the following:

1. Process/compress the data at the source in advance and send the preprocessed/compressed data to the visualization clients, shown as Figure 1-5 (B). If the bandwidth is sufficient, larger portion of preprocessed dataset (up to

the whole dataset) can be delivered over the network. This is the most common solution applied to remote interactive visualization [Callahan 2005; Guthe 2002; Liang 2005; Prohaska 2004].

2. Provide only small pieces of data at the time for visualization, shown as Figure 1-5 (C). If the data to visualize is small, it is possible to run rendering programs as well as filtering procedures on the same system.

It can be summarized that the decision to use these strategies depends the size of data and available bandwidth. We can partition the data-size-and-bandwidth space into three categories shown in Figure 1-6. Due to the large size of the datasets, the server-side filters mostly have to preprocess data into specific file formats or databases. The client-side filters are the traditional filters that just process small datasets because of the limited network bandwidth. Since the bandwidth of wide-area-network is increasing dramatically [DeFanti 2002], OptIPuter becomes a mature model to exploit much more efficient pipeline parallelism. In this dissertation, I'll present a new prototype of distributed data filter for interactive visualization, which is decoupled from data repository servers and visualization application clusters, called filter in-the-middle, shown as Figure 1-5 (D).

With this model, we do not have to preprocess the data on data repository if we can employ the high pipeline parallelism paradigm in distributed visualization applications. Even though the available network bandwidth increases dramatically, many distributed

visualization systems still use preprocessing model in order to reduce latency. Compared to preprocessing paradigm, this on-demand data processing has the following advantages:

1. Timeliness. The users can visualize the latest data as soon as possible, i.e. to view the latest physical simulation results so as to adjust experiment; to see the latest climate forecasting data rather than “aft-casting” data; to detect the arbitrage before the financial exchanges are closed;

2. Efficiency. The users can view the data before deciding to process the whole dataset and store additional data on the disk;

3. Interactivity. To process the data based on interactive visualization and demand rather than texts and numbers;

4. Scalability. The users can view much more distributed datasets than they can store on their specific sites with preprocessed dataset;

5. Flexibility. On the one hand, data-providers are not usually willing to incorporate one-off data-filtering needs of their users. On the other hand, the users of the data don't really want to be in the business of maintaining those large data-sets. Thus, the real-time data filtering can satisfy the need of the users on both ends.

The only possible disadvantage is that this filter-in-the-middle paradigm may have higher latency. Hence, it is the main challenge for OptiStore to minimize overall latency.

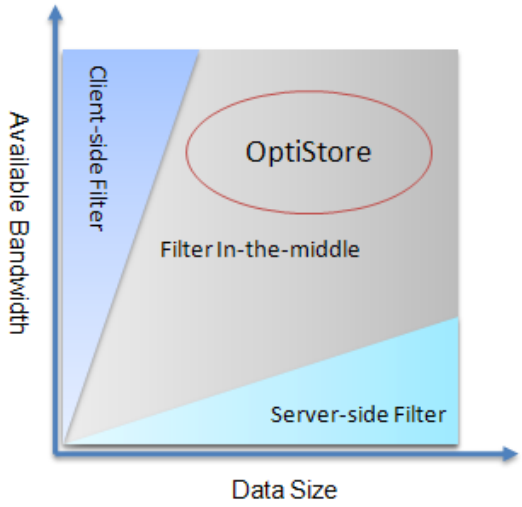


Figure 1-6: To meet the requirement of large scale visualization and exploit availability of high-speed network, I propose a dedicated parallel filtering server in the middle between data repositories and visualization applications

In summary, the proposed approaches address the motivations in very large scale interactive visualization, as listed in Table 1-1.

Motivation	**Approaches**			
	Multi-Resolution Analysis	View-Dependence	Run-time Preprocessing	Distributed & Parallel Computing
Scalability	Yes	Yes	Yes	Yes
Interactivity	Yes	Yes	Yes	Yes
Flexibility			Yes	Yes

Table 1-1: The approaches in OptiStore address the motivations in very large scale interactive visualization

1. 4 Summary of Contributions to the Field

OptiStore is designed and implemented as an on-demanded data management system for very large scale interactive visualization. It addresses several problems of data service for visualization application in OptIPuter architecture: scalability, interactivity, timeliness and flexibility. Compared to other data management systems, this dissertation features the following contributions:

- Design a flexible data filtering middleware model for visualization applications. This dissertation presents a distributed high-performance data filtering model. In this distributed computing model, high-performance data processing system may be separate from both of data repository systems and visualization systems so that it affords more flexibility and power to visualize very large datasets. Especially, OptiStore is such a system that also supports interactive scientific visualization applications.

- Develop OptiStore as an on-demand data processing system. OptiStore aims to provide data querying, filtering and processing services on the fly so that various interactive visualization applications can benefit from this near real-time system. In this dissertation, we implemented several techniques to minimize the latency: load-balancing data partition and organization, multi-resolution filtering, view-dependent data processing, fast remote memory access, Markov chain predictor and etc.

- Realize OptiStore as a scalable data filtering. The techniques, such as load-balancing data partition, multi-resolution filtering, multi-dimensional caching and etc., are designed to load and process very large high-dimensional datasets.

- Introduce a novel cache replacement algorithm for multi-dimensional datasets. The new algorithm was designed for multi-dimensional data with the spatial indexing information in the whole system. It shows superiority over the popular Least Recently Used (LRU) algorithm.

- Implement OptiStore as a flexible extensible framework. The middleware system was designed with design patterns and developed with object-oriented programming language. Various data formats, data filters, caching algorithms can fit into the framework. New data models, data filters, caching mechanisms and networking transfer protocol can be easily added.

1. 5 Document Organization

The remainder of the dissertation is organized as follows.

Chapter 2 reviews and discusses previous work on multi-resolution analysis, visibility culling and distributed and parallel data processing. A comparison of OptiStore and other related system for very large scale visualization is drawn.

Chapter 3 explains the conceptual framework of the proposed system as a scalable parallel data filtering middleware, followed by a brief summarization of the functionalities of each component subsystem.

Chapter 4 presents data management and distribution techniques in parallel data filtering systems. It thoroughly discusses the motivation and the benefits of predefined data distribution along a space-filling curve.

Chapter 5 describes a multi-solution analysis filter for creating a resolution pyramid from original datasets. The basic idea and design details are described.

Chapter 6 elaborates on view-dependent techniques. A visibility culling method for high-dimensional dataset is studied.

Chapter 7 mainly discusses the research work in real-time data processing. A new page replacement algorithm – Furthest Object Replacement (FOR) is presented; remote memory access methods are discussed; and the design of a prediction model is described.

In Chapter 8, experimental models are presented and discussed; experimental results are reported.

Then the dissertation is concluded by 0.

CHAPTER 2 RELATED WORKS

Previously, researchers have proposed various techniques in the areas of multi-resolution filtering of graphical and scientific data, visibility culling and parallel data filtering. In this chapter, we will review and discuss those previous work and literature closely related to the research in this dissertation.

2. 1 Multi-Resolution Filtering in Scientific Visualization

Many scientists have done research work on the multi-resolution analysis (as known as level-of-detail in computer graphic modeling, LOD) for different types of datasets: structured grid [Franke 1999; Guthe 2002; Li 2002b], vector fields [Hua 2003; Jobard 2001], unstructured grid [Abgrall 1998; Rhodes 2002, 2003], graphical primitives [Garland 1999; Guthe 2002; Heckbert 1994; Luebke 2002] and time-varying data [Gao 2004b; Ma 1998; Ma 2003]. To create a reduced dataset, researchers commonly build a hierarchical, multi-resolution representation of the data or geometric model to be visualized (see [Garland 1999; Luebke 2002] for an overview of these methods). In these techniques, a series of coarse representations of the data is constructed using, for example, Quad-trees or Octrees [Herzen 1987; Lindstrom 1996], progressive meshes [Hoppe 1996], or wavelets [Wang 2005]. The level of detail in each region is controlled through a variety of mechanisms, such

as error tolerance bounds that control fidelity to the original model, or user input, such as field of view.

2. 2 View-dependent Data Processing

The view-dependent data processing means that when to process the data, what data to process and/or how to process the data depends on viewing direction, distance, lighting, visibility and etc. Among these factors, visibility is usually the most crucial one because processing the invisible data will result in a lot of duplication and unnecessary work being done.

In order to avoid the unnecessary computation, many visibility culling methods have been proposed for large scale polygon and volume rendering applications [Bittner 1998; Brière 1996; Gao 2004a; Hoppe 1997; Hudson 1997; Klosowski 2000; Zhang 1997]. A survey about visibility for walkthrough application was given in [Cohen-Or 2003] and a comprehensive survey about three-dimensional visibility can be found in [Durand 1999]. Visibility computation algorithms are primarily classified into four categories: image space, object space, viewpoint space and line space [Durand 1999].

Image space methods perform their operations in their 2D projection planes (or other manifolds) and even some intermediate planes. They often deal with a discrete or rasterized version of this plane, sometimes with depth information for each point. Hardware based z-buffer [Fournier 1988] and extensions of this technique, like hierarchical

z-buffer [Greene 1993; Zhang 1997], are frequently utilized in image space visibility computation for occlusion culling. Normally, the visibility computation in this scheme is deeply intervened in the rendering pipelines.

In contrast, object space is the three or higher dimensional space in which the scene is defined. It exhibits the widest range of approaches. Among them, Binary Space Partitioning (BSP) tree is one of most employed methods in this category. It first appears in [Fuchs 1980] for occlusion culling. And other improved versions of BSP trees are also proposed later on [Agarwal 1997; Paterson 1990]. Shen et al. present a Time-Space Partitioning (TSP) tree algorithm for time-varying datasets [Shen 1999].

Viewpoint space is usually equivalent to object space since perspective projection is used in most visualization and graphics applications.

Line space methods characterize visibility with respect to line-object intersections. In this category, multidimensional image-based approaches can be applied for scientific visualization. In multidimensional image-based approaches, a five-dimensional plenoptic function is defined. The plenoptic function describes light transport in a scene, similar data-structures have thus been applied for global illumination simulation [Adelson 1991; Chang 2002; Gao 2004a; Lalonde 1999; McMillan 1995]. Like image space scheme, these approaches are usually highly coupled with rendering pipeline.

2. 3 Distributed and Parallel Data Filtering

Parallel computing power can be utilized to accelerate the data processing efficiently and distributed computing model can be deployed to exploit more data processing resources.

A number of parallel data filtering techniques have proposed for parallel data filtering and image processing. Different parallel image processing environment (PIPE) systems are designed in a single process multiple data (SPMD) architecture for 2D image processing [Hamdi 1997; Lee 1995; Nicolescu 2002; Seinstra 2002; Serot 2002]. Meerwald *et al.* used multi-threads and shared-memory system to do wavelet transforms and image compression for JPEG2000 [Meerwald 2002], even though it required shared memory among the processors. DataCutter is another SPMD PIPE system, which was extended to higher dimension data [Beynon 2000]. But it lacks a lot of complicated spatial operations, like convolutions and etc.

As the fast growth of networking technology, distributed computing becomes more and more popular. Grid computing is a distributed computing technology over the wide-area-network [Foster 1998]. Larry Smarr *et al.* proposed another grid computing architecture - OptIPuter LambdaGrid project, an optical backplane for planetary scale distributed computing [Smarr 2003]. Distributed real-time processing in OptIPuter architecture was also presented [Kim 2004]. And DataCutter system was also implemented in a grid environment [Beynon 2002].

2. 4 Related Systems and OptiStore

2.4.1 Previous Systems for Large Scale Visualization

Since the last decade, many researchers have designed various model and found various solutions to explore very large datasets visually and interactively in the distributed environment. The following systems are those that have a few similar goals and features as OptiStore. In the rest of this sub-section, those systems are introduced briefly one by one, including pros and cons:

1. *Globus GridFTP[Allcock 2003] with HDF5[Yang 2005]*

Globus GridFTP is a similar project to OptiStore. It's a middle-ware tool kits for data transfer on Grid infrastructure. The latest version GridFTP provides functionalities like partial file transfer, stripped data transfer and etc. HDF5 can store two primary objects: datasets and groups. A dataset is essentially a multidimensional array of data elements, and a group is a structure for organizing objects in an HDF5 file. With the support of HDF5, the visualization applications can access arbitrary part of the spatial data from remote site. Some visualization applications [Prohaska 2004] use this solution.

Several disadvantages exist: 1) GridFTP works as file transfer tools. It doesn't have the spatial query mechanism. All the data are viewed as 1D file. Even with the help of HDF5, it's still not convention for flexible spatial queries or geometry operations. 2) With HDF5 and GridFTP API, the application can access partial spatial data. Even though HDF5 is a widely used file format in scientist community, it's not standard format for most images and

volume data. The users have to preprocess the raster images, which leads to additional space and incompatibility between different applications.

Other solutions, like [Allcock 2001; Baer 2004], take NetCDF as the file format in their data repositories. They are still categorized as GridFTP with HDF5 because the difference of data formats does not affect the three aspects of the system: scalability, interactivity and flexibility. In rest of the proposal, when the Globus project is mentioned, it will be referred to Globus GridFTP with special data format such as NetCDF and HDF5.

2. Paradise [DeWitt 1994]

The objective of the Paradise project was to design, implement, and evaluate a scalable, parallel geographic information system that is capable of storing and manipulating massive datasets. By applying object-oriented and parallel database technologies to the problem of storing and manipulating geographic information, the users hope to significantly advance the size and complexity of GIS datasets that can be successfully stored, browsed, and queried.

Paradise has the similar disadvantage as the previous solution: data preprocessing. It has to partition the data into its own format and store with their own storage system: SHORE [Yu 1997]. It's not compatible to other systems.

3. Amanda[Oldfield 2002]

Amanda is a framework that schedules the data processing and distributed disk I/O. Amanda provides a set of toolkit to filter data and transfer data in a grid. Its *SHIP* objects are the agents that gather and transform the raw data for the clients. It is file-system independent and scalable with the data size.

But Amanda has to fetch the full resolution dataset when transforming the data. It does not have any mechanism to generate coarser resolutions of the data and cull the necessary data.

4. DSTP [Bailey 2000] with PDS

The Data Space Transfer Protocol (DSTP) is a protocol to retrieve distributed database [Bailey 2000; Grossman 2003] from Laboratory of Advanced Computing at University of Illinois at Chicago. Each database must have a Universal Correlation Key (UCK), which represent a field in the database. DSTP installs an adapter on each database, which can convert database tables into raw tabular datasets. DSTP includes commands for retrieving meta-data, retrieving UCKs, retrieving data and subsets of data, and mechanisms for sampling, working with missing data, and merging by UCKs. DSTP protocol itself cannot handle very large dataset because it is a TCP-based command message protocol. Therefore, another protocol - Photonic Data Services (PDS) was devised, which aimed to provide bulk data transfer service over LambdaGrid.

DSTP protocol has an obvious limitation that all the data has to be tabular. It cannot handle any data that has higher dimensions more than three. And in the two-dimensional case, because it uses UCK to identify fields/columns, it is hardly scalable in the second dimension.

5. Granite Scientific Database System

Granite Scientific Database System (Granite SDB) was proposed by Rhodes [Rhodes 2002, 2003]. It is a java-based database for general spatial scientific data. Each dataset is loaded into a hyper-volume, called *lattice*. Based on this regular structured grid, a multi-resolution hierarchical model is generated for that dataset.

The Granite SDB can well support multi-resolution data model and different data sources. But the data has to be converted into the database in advance.

6. DataCutter with SRB

DataCutter is another middleware that provides data filtering services [Beynon 2000; Kurc 2000]. San Diego Supercomputing Center's Storage Resource Broker (SDSC's SRB) [Baru 1998] can support distributed across multiple organizations and heterogeneous storage systems. Even though it was claimed that DataCutter *filters can execute anywhere*, essentially they were still implanted on each storage system on the server side because *they are intended to run on a machine close (in terms of network connectivity) to the archival storage server or within a proxy* [Beynon 2000]. Both Allock et al [Allcock 2002] and Oldfield

et al [Oldfield 2002] categorized it as server-side filtering. When the data service client requests the data, the filter sub-sample or subset the data and applied specific filters on the subset data.

DataCutter implemented with SRB can provide flexible distributed data access interface. But as the data filters are essentially located on the storage repository servers, it is still inflexible to deploy filtering system. Additionally, the filters are limited on spatial operations, so that without the aids of advanced techniques like multi-resolution analysis, visibility culling and predicted caching, it is difficult to provide data filtering service the interactive visualization applications.

7. Gao's Approach

Gao designed a distributed data management system for large scale volume rendering system with her own visibility measurement [Gao 2004a; Gao 2005] in a multi-resolution fashion.

However, in her system, the datasets have to be preprocessed with wavelet filters and visibility culling functionality is restricted within direct volume rendering.

2.4.2 Comparison of OptiStore and Other Systems

All of these previous systems can be categorized into three strategies: renderer-side data filtering, storage-side data filtering and an alternative one - render-side data filtering

with preprocessed data on storage side, discussed in previous chapter and shown as Figure 1-5.

Traditional visualization applications can be easily extended into a distributed model by adopting the strategy of renderer-side data filtering. If each node of the visualization cluster fetches data files from remote sites, it can work as the traditional workstation. But as limited bandwidth of long distant network constrains the data size and increases high latency. On the other hand, the limited computation capability to process and access very large data on visualization client leads to that the system cannot be scalable with the data size. DSTP uses this strategy to visualize data.

To overcome the size limitation, some researchers, therefore, preprocess the datasets as specific data file formats on the server side so the visualization application can access the preprocessed data. With special file format, the visualization application can access different level-of-detail of the dataset and any portion of the multi-dimensional data directly. In this way, this alternative strategy can solve the problem caused by limitation of network bandwidth and memory. But the data has to be converted in advance. If the users need other filtering functionalities, they still have to filter the data on the renderer side. And the data provider has to convert their data into those special data formats. This strategy is one of the most popular solutions for many large scale visualization systems. Most of the systems aforementioned fall into this category: Globus, Paradise, Granite, Amanda and Gao.

Then some researchers proposed another strategy. Since the bottleneck occurs on the network connection from storage to renderer. They put the data filter on the storage server side. The data is subset, aggregated and filtered on the storage side before it is transferred to the visualization cluster. The representative system of this strategy is DataCutter. They put the data filter on each storage server, and the storage resource broker system maintenances the metadata information. The storage-side filtering solution doesn't need data preprocessing so that the data providers care neither the maintenance of the data nor the conversion of the data file format. But as the storage servers normally have a lot of high-speed I/O operations and network throughput, the filtering process will increase the additional computation burden on storage server. Furthermore, on one hand, the data users may need to additional filtering functionalities; on the other hand, the data providers do not want or be able to provide every customizable way to filter the data. That will decrease the flexibility of the system. The representative system is DataCutter.

As discussed in the first chapter, currently in the area of data management for very large scale visualization, the main challenge is to satisfy the requirement of scalability, interactivity, timeliness and flexibility. Thus, we compared the systems analyzed in previous section with OptiStore on these four goals.

In summarization, in comparison with other related research projects, OptiStore is the only distributed parallel data filtering system that provides distributed data services near real-time for very large scale interactive visualization.

	Scalability	Interactivity	Timeliness	Flexibility
OptiStore	Yes	Yes	Yes	Yes
Globus	Yes	Yes		
DSTP			Yes	Yes
Paradise				
Granite	Yes	Yes		
Amanda	Yes			
DataCutter			Yes	Yes
Gao	Yes	Yes		

Table 2-1: Comparison of OptiStore and other systems

CHAPTER 3OPTISTORE FRAMEWORK

In previous chapters, the reason why filter-in-the-middle model is adopted has already been discussed and analyzed. In this chapter, we will present how to organize computing power, data repositories, software and etc. for this model.

3. 1 Framework Overview

In order to accomplish the goals of OptiStore system for distributed very large scale visualization: scalability, interactivity and flexibility, the filter-in-the-middle architecture is proposed. This architecture has several features: at the intra-LambdaNode level, the system is designed as a multi-tier distributed system for flexibility; each LambdaNode works in a parallel processing fashion for data scalability and processing speed.

In an overview, OptiStore is a multi-tier distributed system. The reasons to choose this distributed computing architecture are listed as follows:

1. It is natural to adapt multi-tier architecture due to the concept of visualization pipeline. Each pair of adjacent elements in the pipeline work as a pair of client/server system. The output of each component is always the input of the next. The data source -> data filter -> rendering pipeline can be compared to the data-application-presentation architecture.

2. Compared with peer-to-peer models, in the distributed visualization systems, different nodes have different functionalities and different resources. For instance, the data nodes may lack of powerful graphics hardware, while the graphics nodes do not have enough computing power. This cannot meet the requirement of the popular peer-to-peer models because all of the peers should serve both as servers and clients.

3. This is a loose-coupling model. Most of the data transferring and message passing among the components in distributed visualization systems are asynchronous because of the pipeline architecture. Especially the user interface is event-driven. This model can exploit the concurrent utilization of the resources, by minimize task synchronization.

4. It is flexible to develop, implement, extend, and maintain the software on different components in the system. For example, portability is priority-ranked higher in the development of the software on the client while high performance is on the server. So this preference can make the software development and implementation easy and flexible when avoiding designing complicated software to satisfy different requirement in one system.

Figure 3-1 illustrates the diagram of this distributed architecture and Figure 3-2 presents the concept of multi-tier model.

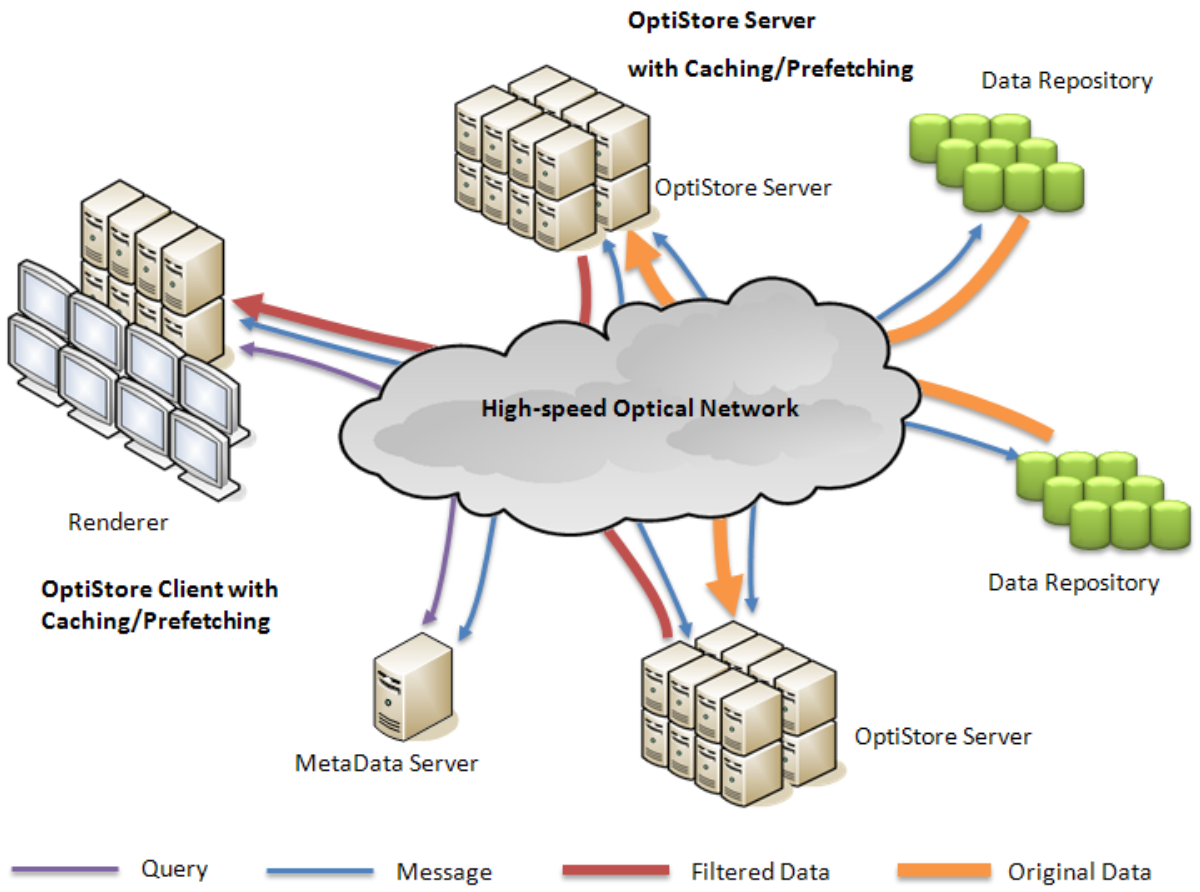


Figure 3-1: The diagram of OptiStore Architecture

As illustrated in Figure 3-1, OptiStore clients are installed on the rendering nodes and provide filtered data for visualization applications (e.g. JuxtaView [Krishnaprasad 2004], Vol-a-Tile [Schwarz 2004b], offline rendering engine and etc.). OptiStore adaptors run on data repositories; they extract meta-data information and original data from fast disk I/O and serve the data for OptiStore servers. Multiple OptiStore servers, typically powerful computing clusters, are placed between the data repository nodes and rendering nodes, which are interconnected via high-speed optical network. OptiStore servers fetch original

meta-data and multi-dimensional datasets, organization the data, process the data and feed query service or filtering service for OptiStore client. Prefetching and preprocessing modules are implanted both in OptiStore clients and OptiStore filtering servers to accelerate data access. The concept of this three-tier model (excluding visualization application on the top and data repositories on the bottom) is demonstrated in Figure 3-2. Each tier in the middle can make the tiers look transparent to the tier above. Consequently, it ensures the heterogeneity and flexibility of the middleware system.

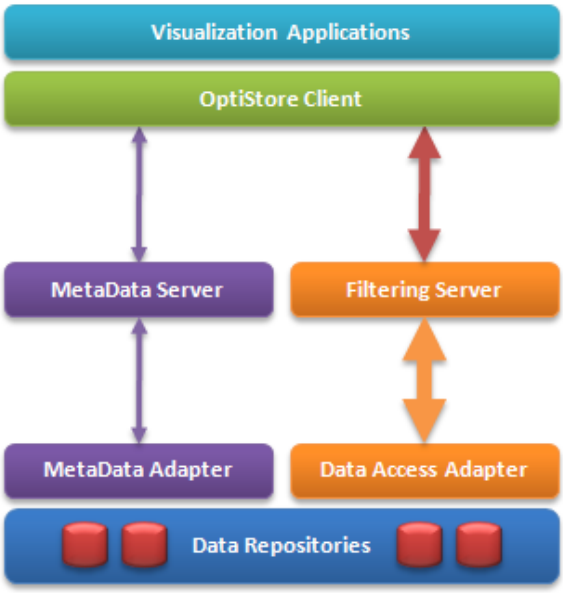


Figure 3-2: The sub-systems in OptiStore: three-tier models

Shown in the diagram of OptiStore architecture (see Figure 3-1) and its conceptual model (Figure 3-2), OptiStore system primarily provides two functionalities: metadata query service and data filtering service. Visualization applications can query the metadata and request specific data filtering via OptiStore client.

3. 2 OptiStore Components

OptiStore comprises two major functional sub-systems: one is metadata service sub-system and the other is data filtering service sub-system. The metadata service sub-system is mainly for dataset publishing, indexing and querying. Whereas the data filtering sub-system is in charge of handling the on-demand data filtering requests from the visualization users.

3.2.1 Metadata Query Sub-System

Metadata query subsystem runs as a basic client-server model. Metadata adapter can extract the metadata information from data repositories - either object databases or indexed file systems; and register the information on OptiStore metadata database. Figure 3-3 depicts the model of metadata query server and adapter, where the dash lines indicate higher level communication over the lower layers, like network, etc. The metadata adapter that runs on the data repository server is a script program that specifies the basic information of spatial data, like dimension, dataset name, location, format, size, and even geo-information. When the dataset is published (or allowed to share), the user just run metadata adapter program to updates the information on the OptiStore metadata server side through socket connections based on QUANTA [He 2003]. The adapter on server side then parses the metadata information and creates a table in the spatial database.

When visualization application users plan to run visualization applications, they just run the query through OptiStore client interface, find the interesting datasets and then bind

the datasets with specified geometry primitives (e.g. planes, boxes and spheres) in a scene graph of xml format.

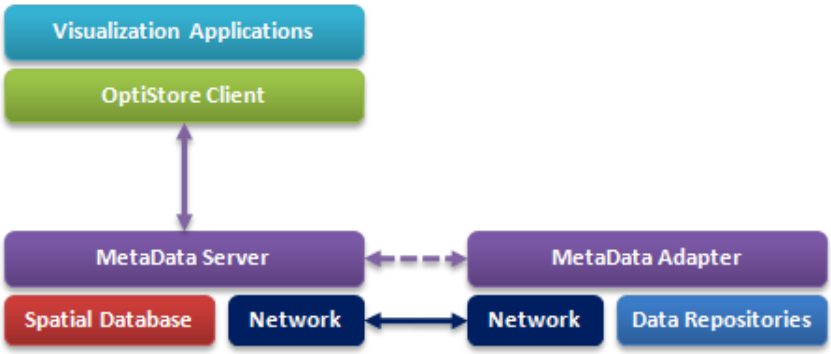


Figure 3-3: OptiStore metadata subsystem: a multi-layer client-server model

The following example (see Figure 3-4) lists an xml file of a scene graph. The scene graph, displaying the earth when being visualized, includes two hemisphere objects with two corresponding datasets as bound textures: the west and east sub-dataset out of the BlueMarble dataset.

```
<?xml version="1.0" encoding="utf-8"?>
<Scene origin="-1.0 -1.0 -1.0" size="1.0 1.0 1.0">
  <Group>
    <Shape type="1">
      <HemiSphere origin="-1.0 -1.0" size="1.0 1.0" radius="1.0"/>
      <Texture dataset="west" server="yorda" gid="201"/>
    </Shape>
    <Shape type="1">
      <HemiSphere origin="0.0 0.0" size="1.0 1.0" radius="1.0"/>
      <Texture dataset="east" server="yorda" gid="202"/>
    </Shape>
  </Group>
</Scene>
```

Figure 3-4: An example scene graph bound with datasets from metadata query

Before requesting the data filtering, the client notifies the master node of the data filtering server to load the datasets by sending the loading request together with the data of the scene graph. Then based on this scene graph, the data filtering servers partition each of the datasets, generate hierarchical data structures for these datasets (such as Quadtree, Octree, etc.), send the data partition information back to the client, and load the data from remote data repositories via data access adaptors.

3.2.2 Data Filtering Sub-System

Data filter subsystem is built using three-tiers: OptiStore data access client, filtering server and data access adapter (see Figure 3-5). And the data transfer mechanism between the tiers is designed as multi-layer model, where the data is transferred at network layer and provided to upper application layer. As the data has been partitioned after the master node of the data filtering servers receives the loading request from the client, when new filtering requests arrive, the nodes on the server will access and process a segment of original data transferred via data access adapter according to the data partition and distribution scheme. Then those server nodes send the filtered data back to OptiStore clients. Based previous request, the filtering servers predict the next request and operation, fetches more data and processes it in advanced before the next data request arrives.

On rendering nodes, OptiStore client serves as data access interface between visualization applications and data filtering server. During runtime, visualization applications call OptiStore API functions to fetched filtered data from remote data filtering server.

OptiStore client API has several modules, illustrated in Figure 3-6, including main controller, data cache, data fetcher, culling module, predictor, and total view module (if necessary).

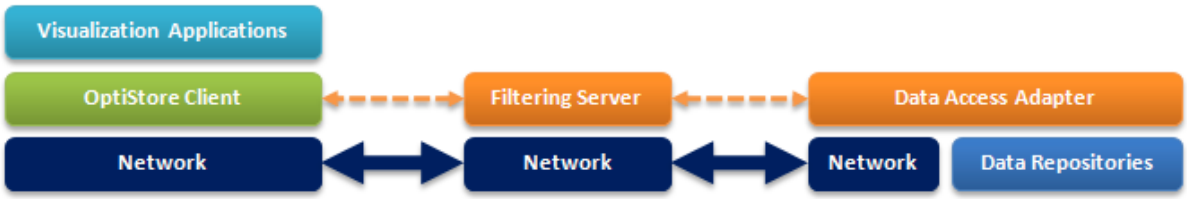


Figure 3-5: OptiStore data filtering subsystem: a multi-layer three-tier model

The duty of OptiStore client can be categorized into two aspects:

1. Update view information (only require for multi-renderer system). If the visualization application is a multi-processor or multi-thread program, a total view module should be placed on the master node/thread to update the whole view information for the data filtering server and synchronize it with all of the rendering nodes.

2. Data caching with prefetching. The main controller receives the view information and data requests from the visualization application. And then it checks whether the data requested exists in cache. If it exists, the cache returns the data to visualization applications; otherwise, it passes view information to the culling module. The culling module culls the unnecessary data out and generates a request queue for the main controller. It also passes the view information to the predictor. Based on some prediction algorithms, the predictor creates another

data request queue and pushes it to the main controller. Once data request queue arrives, the controller notifies remote data filter server and the data fetcher module of data request information. The data fetcher waits for the filtered data from remote data server and transfers the new incoming data to the cache. At last, the cache module loads the newly filtered data and sends them to visualization applications. It should be noted that the requests from culling module has higher priority over those from predictor because those from culling module are the newest requests. So if new request comes, the controller will interrupt fetching data in prediction request queue and update the queue after the latest data fetching finishes.

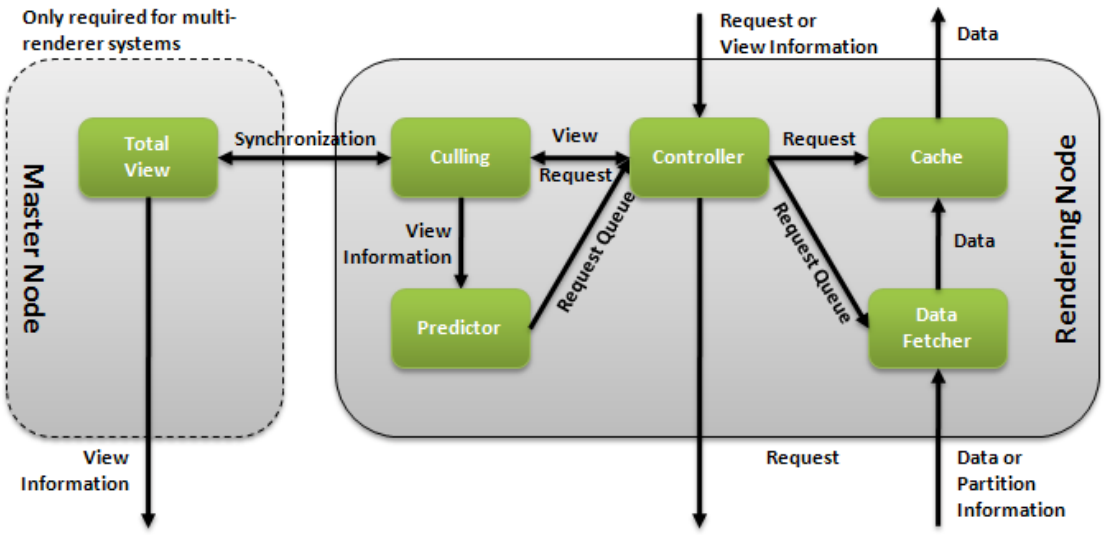


Figure 3-6: OptiStore client modules

The modules on data filtering server is depicted in Figure 3-7. The data filtering server consists of two types of modules: one on the master node and the other on the slave

nodes. The master node includes modules of task dispatcher, data organizer, message sender, and preprocessing and prefetching predictor while the slave node includes modules of main controller, data fetcher, filter, and cache.

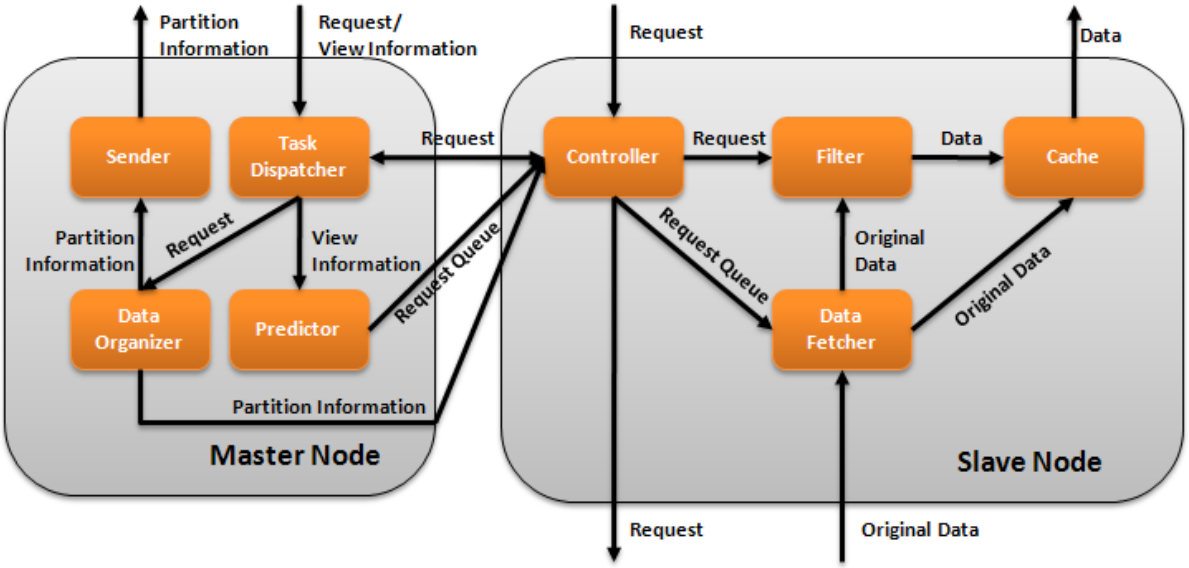


Figure 3-7: OptiStore data filtering server modules

The master node has the global information of the dataset, including view information of the visualization application and data partition and distribution information. It has three main functionalities:

1. Data partition and distribution. When data loading request is initialized by the visualization application via OptiStore client, the request message together with the scene graph is sent to the master node. The task dispatcher passes the request information to the data organizer, which partitions the requested datasets and generates hierarchical structures for them. And then the partition

information is broadcasted to all of the slave nodes and forwarded to the master message sender as well. Finally, the sender sends back the partition information to OptiStore client.

2. Task management. After data loading, OptiStore client has received the data partition and distribution information so that it knows which slave node of the data filtering server to connect when remote data access occurs. On the slave node, once the controller receives the request but the data does not exist on the slave, it will send the request to the task dispatcher on the master node. Then the task dispatcher creates a filtering task and broadcasts the task request to all of the rest slaves.

3. Prediction. Once the visualization application changes its viewpoint or other related parameters, it will update the view information on the master node via OptiStore client. The master keeps this view information as a history track, and predictor module decides what parts of the datasets to be prefetched and processed in advance. This prefetching and preprocessing mechanism will reduce the latency cost when next data filtering and access request arrives.

The modules on the slave node are the core working modules in the whole system. They are integrated on the slave to fulfill two jobs - data filtering and data caching:

1. Data filtering. When data request message comes from OptiStore client to one of the slave nodes, the main controller on that node analyze the request. If the data has already been filtered and loaded in the cache, just send the data back directly. Otherwise, it passes the request to the master node and the task dispatcher on the master node dispatches filtering task to all of the slave nodes. Once the controller receives this task request, it delivers the request to data filter and data fetcher. If requested data is just the original data but not required to process, the data fetcher only fetches data from remote data repositories via data adapters and then pushes the original data into the cache. Otherwise, it fetches the original data and pushes it into the filter. The filter processes the incoming data and transfers the output to the cache on corresponding slave nodes. In current version OptiStore, the filter module includes functionalities such as crop, down-sample, multi-resolution filtering, color transform and isosurfaces extraction.

2. Data caching. The cache on each slave node maintains a buffer to hold a portion of the datasets assigned by data organizer on the master node. It updates the cache according to our novel replacement algorithm if it is over the size limit. Besides this buffer, it also keeps another shared buffer cluster wide. When new filtered data is being produced on other slave nodes, this buffer allows them to write remotely. Since the data partition is designed to avoid overlapping and multiple-reader sharing does not stand, the buffer is not locked up for

simultaneous writing. After the remote writing operation finishes, the buffer is inserted into the cache.

3. 3 Software Library Components

When developing OptiStore middleware, the following factors have been considered:

1. Modularity. As presented in previous section, all of the components in OptiStore system are highly modular. The interfaces between the modules are simple and standard.

2. Extensibility. The whole system is developed in an Object-Oriented (OO) fashion with C++; and various design patterns are applied. For example, the factory pattern is used for the filter module so that more filters can be easily added in later version.

3. Simplicity. Aforementioned, the middleware design of different OptiStore components emphasizes on different aspects. OptiStore client API library aims to provide data service for various visualization applications while OptiStore data filtering server should support high-performance and versatile filtering functions. Therefore, we avoid developing a comprehensive system where every component covers a lot of features. On the contrary, the client API library is

implemented with as few dependencies as possible while we care less about the platform heterogeneity for server programs.

OptiStore client API library, OptiStore query server library and OptiStore data filtering server library are shown in Figure 3-8, Figure 3-9, and Figure 3-10 respectively. QUANTA is for network message passing and data transfer; Pthread libraries (already ported to other operating systems) are for multithreading; and Boost C++ libraries [Boost], are for meta-programming and other features, like smart pointers; and TinyXML is for XML parsing. These four libraries are the common and basic external libraries of the whole system. And furthermore, they are widely applied and compatible with different operating systems. OptiStore client library only relies on these four external libraries. Besides these libraries, OptiStore query server uses Spatial-Index library [Hadjieleftheriou 2005] to index multi-dimensional data. Since OptiStore data filtering server runs on computer clusters, it employs massage passing interface (MPI), the de facto standard communication protocol for distributed memory parallel computing systems, whose current implementation is MPICH2 [Gropp 2002]. In addition, both OptiStore query server and data filtering server implement GDAL/OGR for geo-projection mapping [GDAL].

3. 4 Summary

In a summary, OptiStore middleware is designed and developed as a multi-tier and multi-layer system. It is highly modular and extensible so that new filtering service and data

structure can be added in future. Multithreaded architecture is commonly applied in the whole system. The client API library is portable for different operating system platforms.

Figure 3-8: OptiStore client library and its dependencies

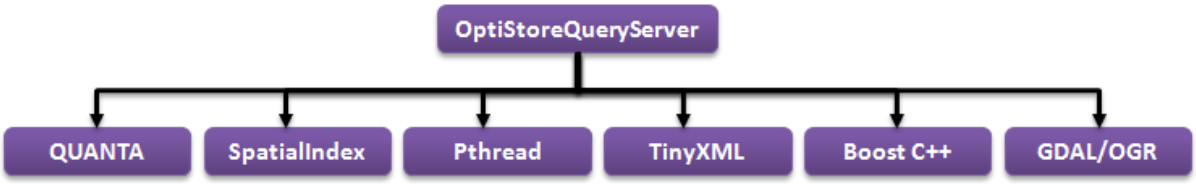


Figure 3-9: OptiStore query server library and its dependencies

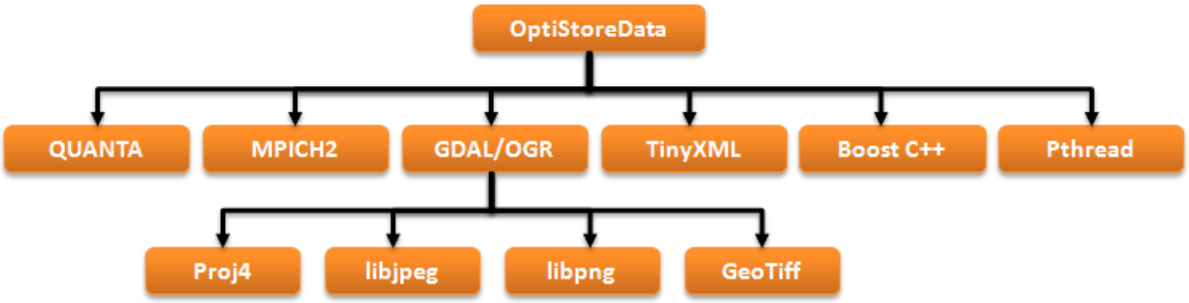


Figure 3-10: OptiStore data server library and its dependencies

CHAPTER 4 DATA PARTITION AND ORGANIZATION

4. 1 Overview

Aforementioned, one of the main goals of parallel system is to achieve high scalability. To reach this goal, it is important for the algorithm to ensure balanced workload.

As a basic LambdaNode in the OptIPuter model, OptiStore filtering server is a distributed-memory multiprocessor system (i.e. a computer cluster). The overhead of the intra-cluster memory access is costly. Furthermore, since the visualization clients, the data filtering servers and the data repositories are distributed in the Wide-Area-Network (WAN), it is a critical task to minimize the runtime communication and synchronization cost so as to reduce the latency in remote data access. On the parallel filtering server in OptiStore system, compared to the data size, the memory resource is extremely limited; and the extra disk I/O access and intra-cluster communication are decisive to the latency in the near real-time data processing. For a high scalable data filtering system under this environment, the workload balance depends heavily on effective and efficient data management and distribution. Therefore, the data management and distribution schema should meet the following criteria [Gao 2004a]:

1. The scheme should avoid data replication to maximize the utilization of limited system resources.

2. The scheme should be predefined to avoid expensive run-time data redistribution.

3. The scheme should keep as high locality as possible so as to only fetch and process necessary data from remote nodes.

4. The scheme should be general enough to ensure balanced workload for different parallel filtering algorithms and different dimensional datasets.

4. 2 Data Partition and Organization

Because the datasets are too large for a single processor, during parallel processing, those datasets have to be divided into small parts and distributed among those processors. Thus, data partition scheme and data distribution scheme are fundamental and crucial. An ill-designed scheme will hurt the overall performance of the whole system. This section will discuss partition and distribution schemes of multi-resolution multi-dimensional datasets.

4.2.1 Data Partition Methods

In that the multi-dimensional data is stored on disk and memory as one dimensional byte array, traditionally, in the multi-dimensional data processing and filtering research, there are two data partition methods [Giess 1998]:

One method is to follow the dimension iteration of the original data and divide the dataset along the highest dimension evenly into chunks, which are called slabs. As Figure 4-1 illustrates, the volume dataset is partitioned into slabs evenly along axis Z, and then the slabs are assigned to the three processors in an interlaced manner to achieve load balancing. Although it is a natural way to partition datasets and for a huge single dataset, there are little overhead to load data from files, the main drawback of this schema is that it cannot preserve the spatial locality of the data within each chunk. Since the visualization application always views the data from different directions, when the dataset is rendered from the Z direction, obviously, all of the chunks have to be accessed and processed, even though only a portion of the X-Y plane is the view frustum. This will lead in high longer process time. In addition, when OptiStore client fetches the data from OptiStore data filtering server, with the restriction of graphics facilities (i.e. each dimension of the texture is limited by graphic cards), since the dataset is considerably large, compared to the dimension limitation, crop or subset is unavoidable on the client side whereas the data fetched is much larger. That results in that either a lot of runtime data redistribution among the client nodes or computation and network utilization runs low on the filtering server. Consequently, the data access time from the visualization application client to data filtering server will increase dramatically.

The other method is to decompose the dataset, along every dimension approximately evenly, into chunks, which are called cells. As Figure 4-2 shows, the volume dataset is divided into eight pieces, two pieces on every dimension. With this partition

schema, each chunk keeps the spatial locality on every dimension. In many cases, the dataset also consists of a set of files which are partitioned on every dimension. And in the visualization applications, the geometry and texture on parallel renderer are also usually partitioned in this way. Thus, this schema can minimize the intra-cluster communication and increase the resource utilization on the parallel system.

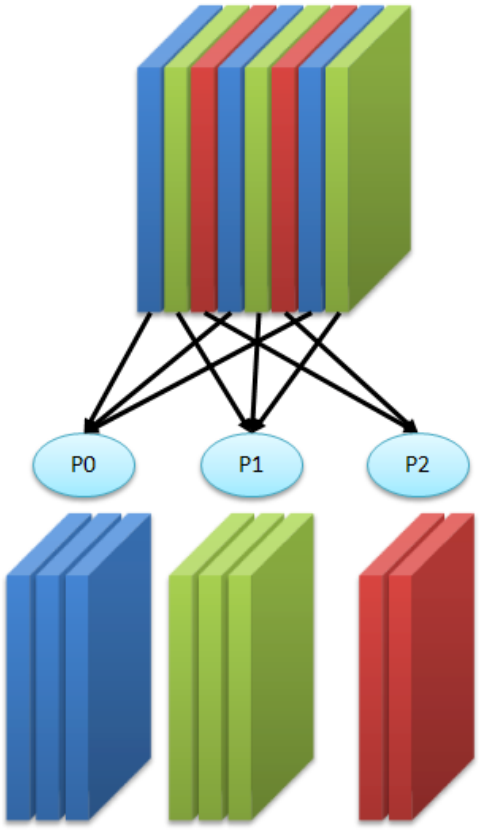


Figure 4-1: Data partition in slabs

In summary, due to the importance of the spatial locality in visualization applications, the data partition schema in which the dataset is decomposed into cells is

preferred in OptiStore system. In the rest of this dissertation, these cells will be called “blocks”.

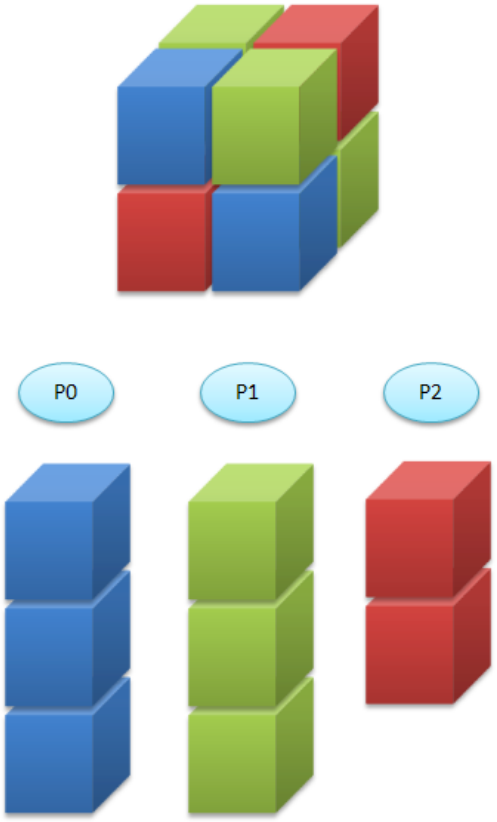


Figure 4-2: Data partition in cells

4.2.2 The Granularity of Blocks

In Figure 4-2, although the locality is preserved, an obvious load unbalancing is noticed. Processor 0 and 1 load 50 percent more data than processor 2, respectively. The unevenness should be blamed on the large size of the blocks. Alternatively, if the block decreases half on every dimension, illustrated as Figure 4-3, the data load ratio on the three processors is 22:22:21. The data loads are very closely even though they are not totally even

yet. In another scenario, if there are 16 nodes on the computing cluster, in the former partition schema, half number of the nodes run without any data; in the latter schema, 4 blocks are loaded evenly on all of the nodes.

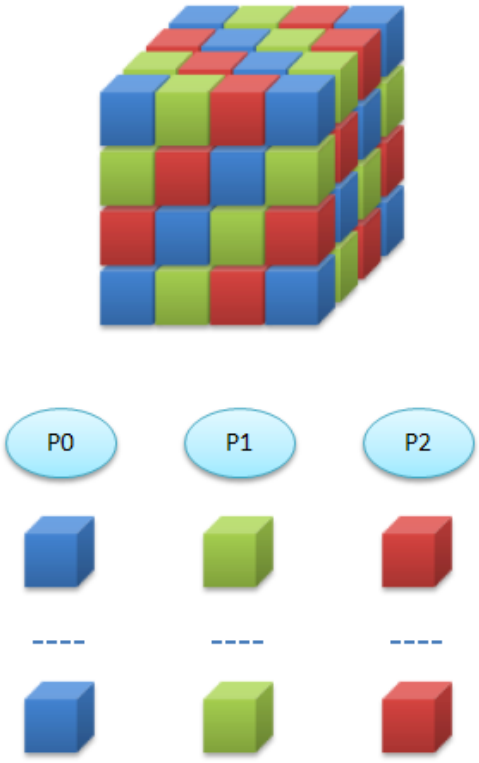


Figure 4-3: Data partition into cells with finer granularity

This contrast demonstrates that the finer granularity of the block can benefit the load balancing in parallel computing. In practice, it is not always the truth. The following disadvantages will emerge if the size of the block is too small because the decrease of the block size leads to the increase of the number of blocks, in the order of two, three, four, or greater:

1. The size of the data structure that indexes the blocks will increase too large. As a result, the time of frequent operations on the data structure (like look-up, insert, delete and etc) will get longer.

2. The cost on distributed data access will grow up. Either long distance data transfer, where small messages needs to be aggregated for bulk data transfer over high-speed optical network, or intra-cluster remote memory access, where synchronization messages have to be posted, the increasing number of message will lower the efficiency of the network transfer.

3. The overhead of data transfer in memory will rise. Since the data is stored in the main memory and video memory linearly, a multi-dimensional data transfer always brings out a lot of operations such as merge and crop. The smaller the block size decreases, the more fragments exist during those operations. That will raise the overhead inevitably.

4. If the block size is too small and multi-resolution filtering algorithm is block-wise (that is multi-resolution transform is applied to each block), the level of the resolution is really limited. Thus too much block size decrease will harm the generality of data partition scheme.

5. The increasing block number will generate more artifacts along the boundaries. If the artifact removal algorithms are employed, the computation cost will rise much highly.

In the implementation of OptiStore, it allows the user to specify the size of blocks before loading the dataset. By default, OptiStore assigns the block size with the size that is closest to the maximum size of graphics card limitation. For example, the two-dimensional dataset of Blue-Marble's west hemisphere, whose dimensions are 21,600 pixels by 21,600 pixels, is partitioned into 64 (8 by 8) blocks, each of which is 2,700 pixels by 2,700 pixels, if the maximum size of two-dimensional texture graphics card is 4,096 pixels by 4,096 pixels.

4.2.3 Data Organization

In OptiStore, each dataset is assigned a Universally Unique Identifier (UUID) number and each block in an N-dimensional hypercube at any resolution level has a coordinate: (D_0, D_1, ..., D_N), which indicates the offset of the block along each dimension from the origin within the hypercube. Then any block in OptiStore system has a unique identifier: (UUID, Level, (D_0, D_1, ..., D_N)). Here, this unique identifier is called universal block coordinate. Suppose the dimension size of the block is W(W_0, W_1, ..., W_N) and the origin coordinate of the hypercube at that level is O(O_0, O_1, ..., O_N). Thus, an arbitrary point with coordinate C(C_0, C_1, ..., C_N) in a dataset can be located via UUID number, level of the resolution, and block coordinate D(D_0, D_1, ..., D_N) and the offset within the block X(X_0, X_1, ..., X_N), where $D_i = C_i / W_i$ and $O_i = C_i \% W_i$.

For each dataset, all of blocks are traversed from the lowest resolution to the highest resolution once. Based on the traverse order, the blocks are distributed among the filtering processors in a round-robin manner. Figure 4-4 demonstrates a three-level resolution volume dataset is partitioned into seventy-three blocks and the blocks are distributed among four processors. In blocks distributed on the same processor are assigned one color.

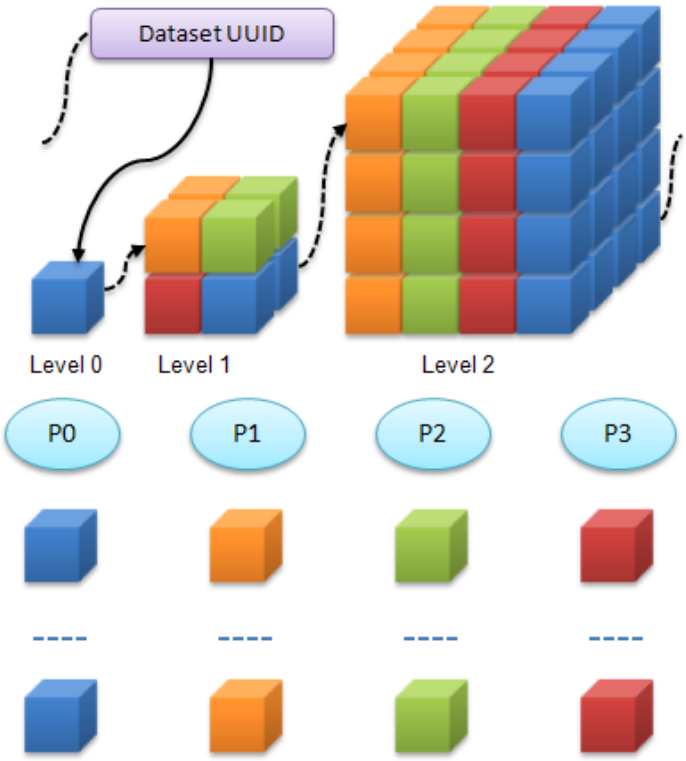


Figure 4-4: Data partition and distribution of a multi-resolution dataset

Therefore, for a block, it has two unique identifiers: one is the universal block coordinate; the other is the pair of processor id and order in the block array. Since that block is unique, these two identifiers are one-on-one mapping between each other. In that

an arbitrary point can be located through universal block coordinate, it also can be located in a processor's memory space via this one-on-one mapping.

At runtime, due to the limited capacity of the memory and the large size of the data, only a few of blocks can be loaded into the processor's cache. The block number is like the page number in operating system's virtual memory. The block swapping in and out will take place by following some replacement algorithms.

Figure 4-5: One-on-one mapping from a dataset block to a page in memory

4. 3 Data Distribution for Load Balancing

Even though the previous data partition schema – partition into blocks, is better at preserving spatial locality and load balancing than the one - partition into slabs, problems still remain to be solved. In Figure 4-4, a familiar situation could be observed: if the top-left orange blocks at resolution level 1 are requested from the client but not exist, a parallel filtering on the higher resolution of level 2 will occur. It can be noticed that only the partial orange and green blocks will be processed. It means that only processor 1 and 3 are filtering data and the other two are idle. This load unbalancing and poor parallel utilization is caused by the scan-line traversal order (shown as Figure 4-6) through all the blocks, which does not preserve much spatial locality enough.

To solve this kind of problem, researchers propose many techniques. In 1998, Gaede and Günther summarized almost of all of multi-dimensional access methods, for regular gridded and irregular unstructured data [Gaede 1998]. Campbell, et al. used space-filling curves to load octrees in balance [Campbell 2003] and Gao did the similar to multi-resolution octrees [Gao 2005]. Lawder extends the space-filling curve methods to much higher dimensional data indexing [Lawder 2000]. Founded on their research and with the hypothesis that the similar techniques can provide balanced work for parallel filtering algorithms, OptiStore takes the advantage of space-filling curve methods for its data distribution of several high dimensional multi-resolution datasets.

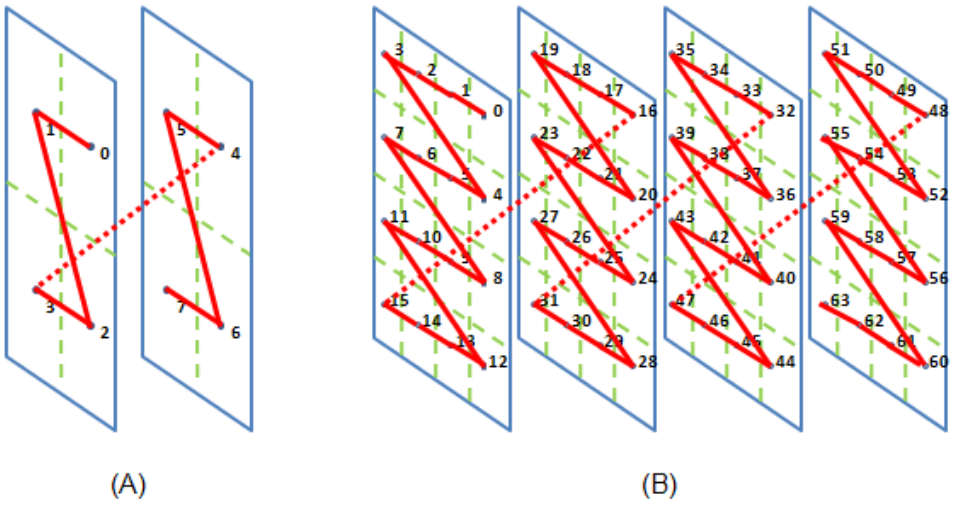


Figure 4-6: Two level three-dimensional scan-line curves, not a space-filling curve: (A) 2 level and (B) 3 levels. The numbers indicate the traverse order.

Space-filling curves [Sagan 1994], also known as Peano curves, are curves whose range contains the entire two-dimensional unit square (or the three-dimensional unit cube).

In 1890, Peano discovered a densely self-intersecting curve which passed through every point of the unit square, which was the first example of a space filling curve. There are many other space filling curves available, such as Hilbert curve, Morton curve (Z-order curve) etc. Two dimensional examples of these curves are illustrated in Figure 4-7.

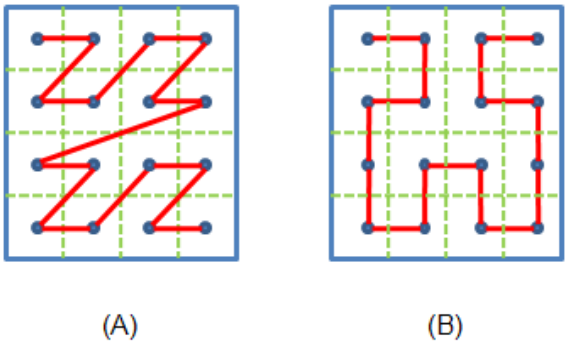


Figure 4-7: Two examples of space-filling curves - (A) a Morton curve; (B) a Hilbert curve

The main reason to use space-filling curves for data distribution is that a space-filling curve preserves spatial locality – a curve always visits adjacent blocks if possible before it leaves the local neighborhood. In our view-dependent and multi-resolution data processing techniques, the spatial locality implies that if one block is accessed and processed, the blocks within its neighbor are likely to be accessed and processed too. While the data distribution among the processors is along the curves in a round-robin manner, then those blocks may be stored on different processors possibly evenly [Gao 2004a]. The previous scan-line traversal path is converted into the ones shown in Figure 4-8 and Figure 4-9, which demonstrate good spatial locality behavior.

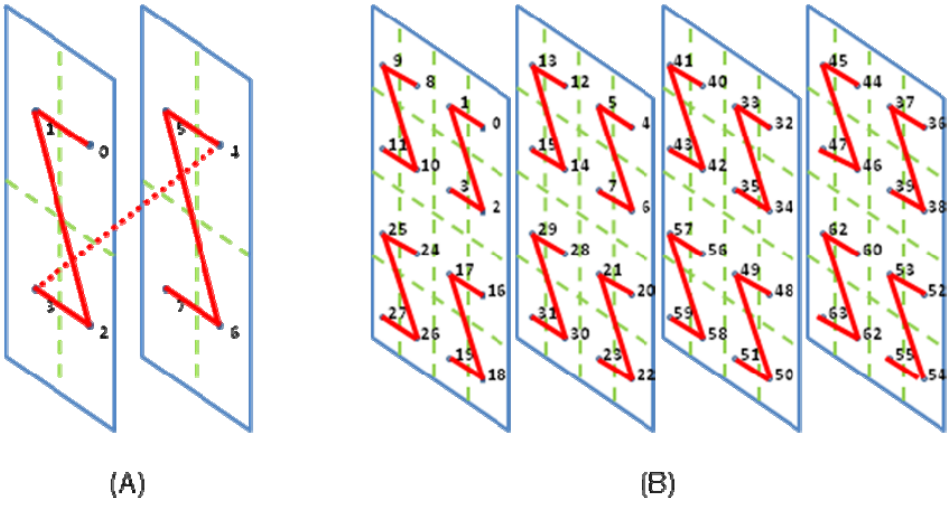


Figure 4-8: Two level three-dimensional Morton (Z-order) curves: (A) 2 levels and (B) 3 levels. The numbers indicates the traverse order. (The dashed lines between planes in (B) are omitted for clarity).

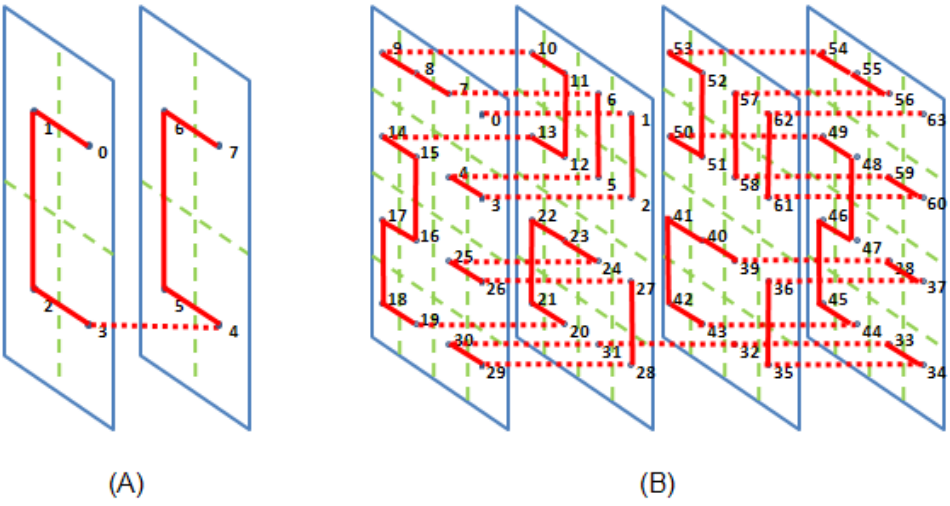


Figure 4-9: Two level three-dimensional Hilbert curves: (A) 2 levels and (B) 3 levels. The numbers indicate the traverse order.

The Hilbert curve has better spatial locality preserving behavior than the Morton curve. But in Hilbert curves, the calculation of multi-dimension to one-dimension mapping is more complicated. Considering that the data partition and block traversal is a onetime process when loading the dataset, we prefer the Hilbert curve for our data distribution.

4. 4 Implementation and Results

We applied the techniques discussed previously to our data partition schemes in the data organizer module of OptiStore data filtering server.

When the initial data loading request from OptiStore client, the metadata information of the datasets, including dataset UUID, total resolution levels, maximum block size, etc., arrives at OptiStore data filtering server, the data organizer will generate multi-resolution pyramid according to the number of total resolution level of each dataset, partition the datasets into blocks based on the maximum block size by each dimension, and assign the unique block IDs to the processors along space-filling curves. When the data multi-resolution processing request arrives from a slave node at the master node, the task dispatcher analyzes the global existing block information on the cluster, and distributes the processing job to the processors that should keep the corresponding blocks assigned at the loading time.

To measure load balance of parallel processing applications, overall processor utilization is a good and intuitive metrics. Let T_i be the computation time on processor i

and I_i the idle time during the run-time of a parallel task on n processors; then overall process utilization is defined as:

$$U = \frac{\sum_{i=1}^{n} T_i}{\sum_{i=1}^{n} (T_i + I_i)} \tag{4-1}$$

According to this definition, it is evident that the upper limit of overall process utilization is 1. That is to say if the workload is perfectly balanced (without any idling time on any processor), U equals to 1. Assume the computing capability of each process equal, the more evenly the workload is distributed, the larger U is.

By using the measurement metrics defined in Equation 4-1, we tested the over process utilization with different schemes and different number of processors (excluding the master node). In Table 4-1, the results are listed, which show that space-filling curve method is much superior to the simple scan-line method and Hilbert curve method is slightly better than Morton curve method.

Scheme	**Processor Number**		
	4	8	16
Scan-line	78.2%	74.5%	69.8%
Morton	94.7%	92.2%	86.9%
Hilbert	94.7%	92.3%	87.2%

Table 4-1: The overall process utilizations with different data distribution scheme

4. 5 Summary

This chapter has discussed several problems about data partition and distribution on OptiStore data filtering server for parallel processing and explained what scheme is adopted to organize the data.

Firstly, in order to achieve the scalability, data should be partitioned and distributed among the server nodes. Secondly, due to the locality feature in visualization applications, data are partitioned into blocks instead of slabs. Thirdly, the granularity of blocks has been considered. It should be avoided to choose too large or too tiny blocks. And the size can be user-defined. Fourthly, the data block of any dataset is uniquely identified. Last but not least, to achieve even load balance, the data block is distributed according to the order of some space-filling curves, such as Morton curves and Hilbert curves.

CHAPTER 5 MULTI-RESOLUTION FILTERING

5. 1 Multi-Resolution Filtering with Wavelet Transform

As afore-stated in previous chapters, multi-resolution filtering is very important to very scale visualization applications. In computer graphics texture filtering, the technique of texture mipmaps is widely applied for rendering acceleration and anti-aliasing. In very large scientific visualization, the multi-resolution analysis with wavelet transform is often employed. Essentially, the interpolation in mipmap filtering is a special case in wavelet transform and the filtering result of the low-pass sub-band of each wavelet transform. For example, the linear interpolation filtering is the low-passing filtering of Harr wavelet transform [Sweldens 1995].

Compared to mipmap filtering and other down-sampling algorithms, the wavelet transform has the advantages as follows:

1. Wavelet transform is fast. Discrete wavelet transform (DWT) takes $O(N)$ time compared to $O(N \log N)$ Fast Fourier Transform (FFT).

2. Wavelet transforms do not have blocking artifacts like other transforms, such as Gibbs artifacts in Discreet Cosine Transform (DCT) at low-resolution [Malvar 1999].

3. Wavelet transforms are naturally embedded and can reconstruct images progressively in resolution. Its intrinsic embedded feature allows for the data structures like quadtree and octree as shows, which are the primary data structures in OptiStore. With the generated quadtree / octree, the high-passing sub-bands can indicate the significance of the details on the sub-blocks. If the visualization applications on the client work with error control methods, some wavelet-based encoding algorithms can provide such mechanism so that when a relatively uniformed block exists, it will save computation and network resource to feed data at higher resolution.

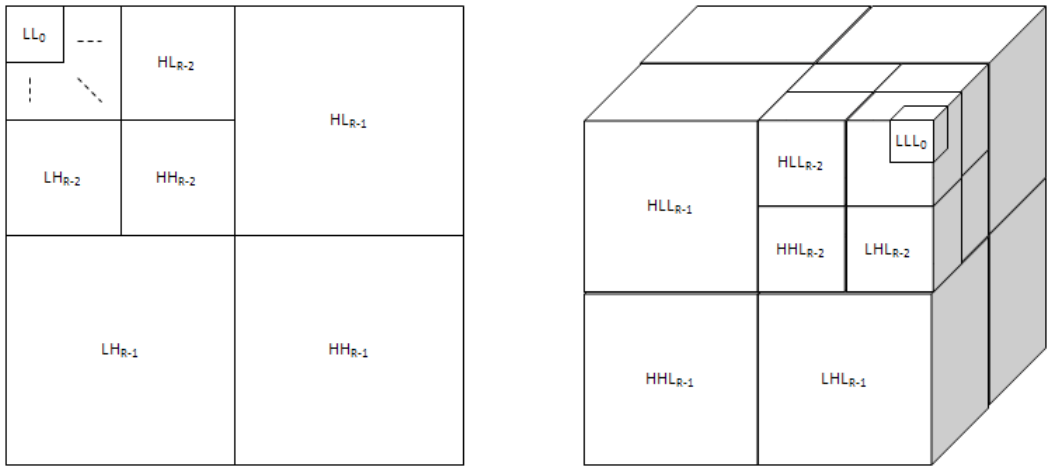


Figure 5-1: Wavelet multi-resolution analysis on multi-dimensional datasets. (A) A two dimensional dataset filtered into R level resolutions as a quadtree; (B) A three dimensional dataset filtered into R level resolutions as an octree.

4. Wavelet transform is widely employed in many image compress codecs due to its better distortion/rate performance resulted from its energy compactness.

5. Tiling technique is maturely utilized in wavelet-base codecs like JPEG2000 for encoding very large images [Christopoulos 2000]. This scheme is similar to the blocking scheme in OptiStore and easily to be tailored in parallel computing systems. Although some artifacts (for example, ring artifact and tile artifact) appear as a result on post-processing of JPEG2000 compressed images, these artifact don't show in OptiStore system for two reasons: one is that some algorithms can be employed to reduce the artifacts [Berkner 2002]; the other is that in OptiStore, no entropy quantization is applied on the transformed data, because the filtering is for using lower resolution intermediate image but not for compression which is the main factor that leads to those artifacts.

6. Wavelet transform is broadly involved with various datasets. At the beginning of wavelet development, the wavelet transform is used in regular grid images. Later, it is also progressive mesh simplification algorithms based on its simple lifting scheme [Khodakovsky 2000]. As the GPU-based displacement method become popular in terrain rendering [Asirvatham 2005; Livny 2007], the utilization of geometry images rather than semi-regular grid can achieve better performance thanks to its simple and GPU-friendly data structure. Thus, the

wavelet transform can also be applied on height-field data for terrain visualization.

Therefore, based on the analysis above, the wavelet transform was chosen for the multi-resolution filtering in OptiStore.

Traditionally, the discrete wavelet represents a signal in terms of shifts and dilations of a low-pass scaling function $\varphi(t)$ and a band-pass wavelets function $\psi(t)$ [Daubechies 1992]. The transform is multi-scale, in that it creates a set of coarse coefficients that represent signal information at the lowest scale, and sets of detail coefficients with increasingly finer resolution. The transform is typically implemented as a filter bank with analysis low-pass filter $H(z)$ and high-pass filter $G(z)$, as shown in Figure 5-1 (A). The inverse transform uses synthesis low-pass $\tilde{H}(z)$ and high-pass $\tilde{G}(z)$, as shown in Figure 5-2 (B). For special choices of H, G, $\tilde{H}$, and $\tilde{G}$, the underlying wavelets and scaling functions form a biorthogonal basis and provide perfect reconstruction. The transform is typically iterated on the output of the low-pass band (c[n]) to create the series of detail coefficients at different scales.

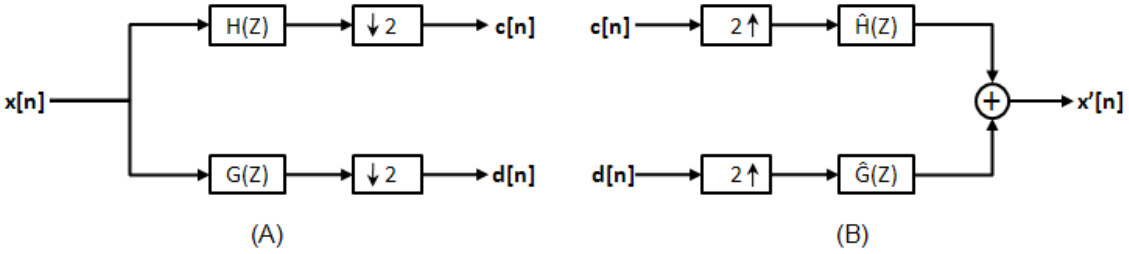


Figure 5-2: Filter bank implementation of discrete wavelet transform. (A) H and G are the analysis low-pass/high-pass pair. c[n] and d[n] are the scaling and wavelet coefficients, respectively. (B) It is the corresponding inverse wavelet transform. $\hat{H}$ and $\hat{G}$ are the synthesis low-pass/high-pass pair.

Later on, the lifting scheme was introduced to construct the second generation wavelet [Daubechies 1998; Sweldens 1995]. A typical lifting stage is comprised of three steps: Split, Predict, and Update (shown as Figure 5-3 (A)).

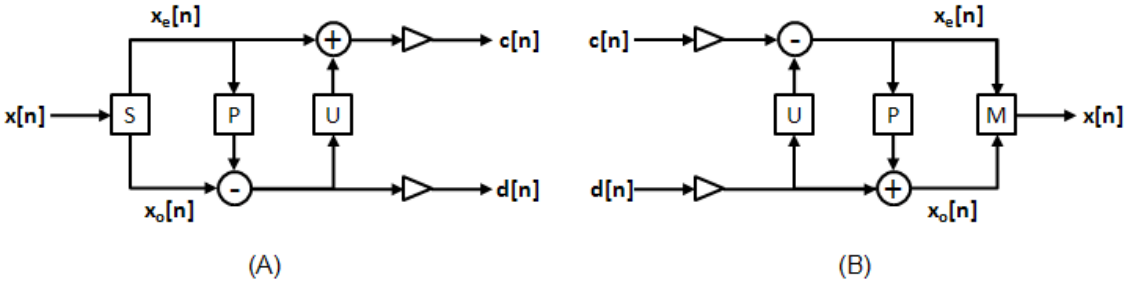


Figure 5-3: The lifting scheme. (A) Typical lifting steps: Split, Predict, and Update; c[n] and d[n] are the scaling and wavelet coefficients, respectively. (B) Typical inverse lifting steps: undo Update, undo Predict, and Merge.

Split: Let $x[n]$ be a signal. We first split $x[n]$ into its even and odd polyphase components $x_e[n]$ and $x_o[n]$, where $x_e[n] = x[2n]$ and $x_o[n] = x[2n+1]$.

Predict: In the interpolating formulation of lifting, the odd polyphase coefficients $x_o[n]$ are predicted from the neighboring even coefficients $x_e[n]$. The predictor for each $x_o[n]$ is a linear combination of neighboring even coefficients:

$$P(x_e)[n] = \sum_l p_l x[n+l] \tag{5-1}$$

Then a new representation of the $x[n]$ replaces $x_o[n]$ with the prediction residual. This leads to the first lifting step:

$$d[n] = x_o[n] - P(x_e)[n] \tag{5-2}$$

If the underlying signal is locally smooth, the prediction residuals d[n] will be small. Furthermore, the new representation contains the same information as the original signal $x[n]$: given the even polyphase $x_e[n]$ and the prediction residuals $d[n]$, the odd polyphase coefficients $x_o[n]$ can be recovered as follows:

$$x_o[n] = d[n] + P(x_e)[n] \tag{5-3}$$

Update: The third lifting step transforms the even polyphase coefficients $x_e[n]$ into a low-pass filtered and sub-sampled version of $x[n]$. This coarse approximation is obtained by updating $x_e[n]$ with a linear combination of the prediction residuals $d[n]$. $x_e[n]$ is replaced with:

$$c[n] = x_e[n] - P(d)[n] \tag{5-4}$$

where U(d) is a linear combination of neighboring d values:

$$U(d)[n] = \sum_l u_l d[n+l] \tag{5-5}$$

Each lifting step is always invertible; no information is lost. Assuming the same P and U are chosen for the analysis and synthesis stages, the lifting construction guarantees perfect reconstruction for any P and U. Given $d[n]$and $c[n]$, we have

$$x_e[n] = c[n] - U(d)[n] \tag{5-6}$$

and $x_o[n]$ from (5-3).

The inverse lifting scheme is shown in Figure 5-3 (B).

Compared to the traditional discrete wavelet transform, the lifting scheme has the following immediately advantages [Uytterhoven 1999]:

1. It is faster (but still *O*(n), of course). In the case of CDF(2, 2), the original filters H(Z) and G(Z) count 5+3=8 filter coefficients, while the lifting decomposition gives us P and U counting only 2+2=4 filter coefficients.

2. The inverse transform is trivial to find. One does not need to explicitly find the corresponding analysis filters.

3. The lifting scheme is easier to understand and implement.

4. The lifting scheme is more generic and allows for the creation of second generation wavelets. All wavelet filters can be implemented using the lifting scheme. Thus with this lifting scheme, we can construct some especial wavelets, like integer-to-integer wavelets for lossless data transform.

Based on the analysis above, we choose the lifting scheme for wavelet filtering. In the rest of this dissertation, we use a DWT box to represent the lifting steps for illustration, shown in Figure 5-4.

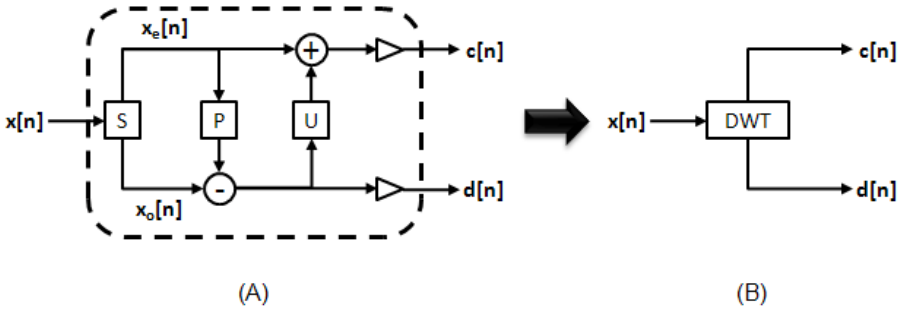


Figure 5-4: The lifting scheme is chose in OptiStore

5. 2 Wavelet Multi-Resolution Filtering for Multi-Dimensional Datasets

In multi-resolution analysis for multi-dimensional data, the wavelet transform can be extended to multiple dimensions by use of separable filters. Each dimension is filtered and down-sampled separately [Bilgin 2000]. Although non-separable wavelets can also be used to filter multidimensional data, such filters are much harder to design than are separable filters. As a result, their use has been limited in multi-resolution applications. On the other hand, due to the anisotropy in some datasets, for example, the temporal volumetric

datasets, sometimes different filters should be applied to different dimensions. For example, for some four-dimensional medical datasets, the variances along the axial and temporal direction are determined by the thickness of slices and imaging speed. The variances among four dimensions may be very different. In general, the similarity of pixel values along the temporal direction is expected to be closer than along the other three directions, and similarity along the X and Y directions is very close. This asymmetric similarity has been shown in [Liu 2007] for four-dimensional medical image data sets. Therefore, it is reasonable to apply transforms along some dimensional in different ways in the separate multi-dimensional wavelet transform. Figure 5-5 illustrates the implementation of two levels of a three-dimensional dyadic decomposition with separable filters. Similar to the one dimensional case, three-dimensional wavelet transform coefficients can also be stored in a data cube that has the same size as the input data. In Figure 5-5, each sub-band is labeled, and the corresponding locations for storing these sub-bands in the output data cube are identified.

5. 3 Implementation of Multi-Resolution Filtering in OptiStore

In practical implementation of the multi-resolution filtering in real-time applications, if the overview of very large data is required at the beginning of data processing and visualization, to reduce the latency cost, we use down-sampling methods to generate coarser resolutions rather than discrete wavelet transform, even though it may introduce more artifacts than discrete wavelet transforms. Then during the idle time of interactive

visualization procedures, OptiStore data filtering server runs discrete wavelet transform to generate coarser resolution data blocks based on previous requests.

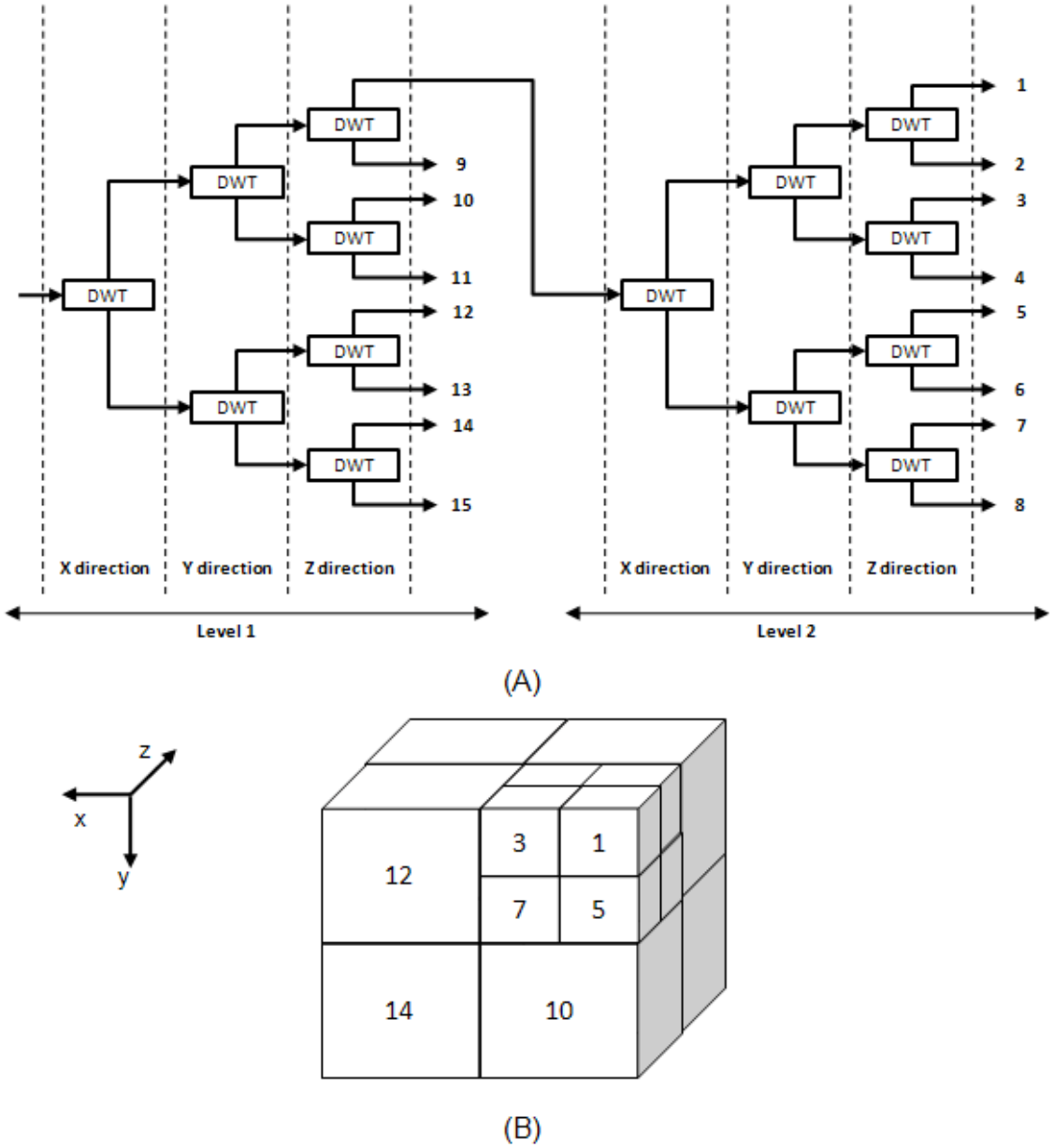


Figure 5-5: Three-dimensional wavelet analysis

If a data block request arrives at the task dispatcher of OptiStore but that block does not exist in the whole cache of the cluster, the task dispatcher will check the children of that block in the hierarchy data structure. A 2D quadtree example is illustrated in Figure 5-6. In the three-level quadtree, the red block on the top is the requested missing block in cache. Three blocks out of its four children exist in cache, with blue color in the figure. If any child is missing, like the block one in the figure, then go downwards to check its children recursively until reaching the leaves that represent the original data blocks. In this example, the descendent of the black block are the green ones that are also original data. Thus, to achieve the red block, we should process seven blocks: four original blocks with green color, each wavelet transformed twice; and three blue blocks, each wavelet transformed once. As discussed in the previous chapter, these data blocks should be distributed evenly among cluster nodes. The filtering is processed parallel. At last, all of the lowest resolution sub-bands are merged into one block as request. More details about data transfer of the filtered blocks between nodes will be described in Chapter 7.

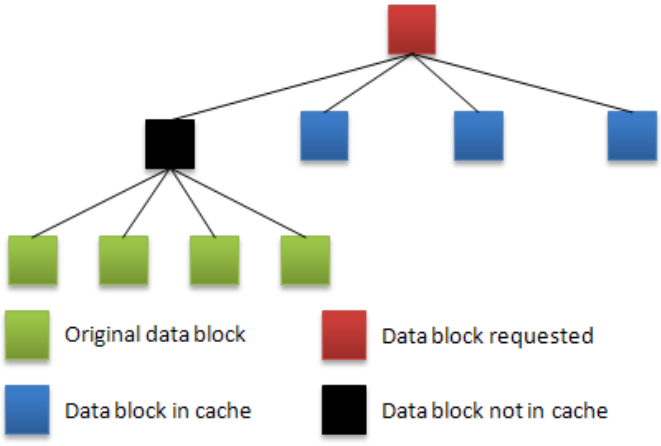


Figure 5-6: Implementation of multi-resolution filtering

In OptiStore, by default, Haar DWT is applied to float- or double-type data and integer Haar DWT to integer- and byte-type data (normally color data in RGB format are treated as three bytes).

5. 4 Summary

In this chapter, we explained the reason to utilize discrete wavelet transform as multi-resolution filter and present the method to apply this filter to multi-dimensional datasets. We also described the practical implementation of this filter integrated with data partition scheme and parallel filtering modules in OptiStore.

CHAPTER 6 VIEW-DEPENDENT DATA PROCESSING

In order to reduce the latency during near real-time processing, the data that does not contribute to the visualization scene should be excluded from being computed. This visualization or data processing based on this scheme is called view-dependent visualization or view-dependent data processing.

6. 1 View-dependent Visualization and Data Processing

Originally, view computation has been the major focus of early computer graphics research. Visibility was a synonym for the determination of the parts/polygons/lines of the scene visible from a viewpoint. Later on, since the data size becomes larger and larger, the view computation is used in other aspects of computer graphics, such as out-of-core graphics, geometry model simplifications and etc [Durand 1999].

Generally, view-dependency is referred to the dependency on the visual properties during visualization, such as visibility, distance, direction, illumination and etc. Besides visibility, the distance and direction properties are widely employed for level-of-detail polygon simplification [Luebke 2002] and the calculation of the contribution to the final projected image is often used in ray-tracing-like or splatting-like visualization applications. However, these calculations are usually closely integrated with rendering pipeline. For

example, when to measure the criteria like a screen-space error threshold, a silhouette test, and a triangle budget in view-dependent LOD simplification [Luebke 2001] or Plenoptic Opacity Function in direct volume rendering [Gao 2004a], multi-pass computation is unavoidable. Most of the intermediate output and input data during the multi-pass computation are the images from hardware's frame buffer (i.e. by OpenGL's glReadPixels function). If the data processing process and the rendering process are distributed, large frame buffer data reading and transferring between the processes will occur so that the latency will eventually rise much higher, especially in wide-area-network systems. On the other hand, the rendering methods in visualization applications vary from one to another. Different criteria or control functions on those parameters may be applied. Too complicated automatic view-dependent determination calculation will also result in unnecessary computation so as to increase the latency as well as generate false view results.

Therefore, under OptIPuter's distributed computing environment, this data processing scheme, which is tightly coupled with data rendering pipeline, is not suitable and contradicts one of our goals: flexibility. In opposition, the view-dependence computation based on visibility culling just requests a few parameters from the rendering process and then all of the view-dependence computation can be completed during the data processing pipeline. Thus OptiStore takes visibility culling as its automatic view-dependent data processing scheme on the data filtering server and leaves other complicated view-dependence computation in the rendering pipeline to the visualization application developers.

6. 2 Visibility Culling

In computer graphics, hidden surface removal (HSR) or visible surface determination (VSD) is a traditionally important topic. A related area to visible surface determination (VSD) is culling, which usually happens before VSD in a rendering pipeline. Primitives or batches of primitives can be rejected in their entirety, which usually reduces the load on a well-designed system.

The advantage of culling early on the pipeline is that entire objects that are invisible do not have to be fetched, transformed, rasterized, or shaded. Here are some types of culling algorithms, shows as Figure 6-1:

1. *Viewing frustum culling*

The viewing frustum is a region of space within that the objects are visible to the notational camera or view point. Of course, objects outside this volume will not be visible in the final image on the screen, so they are discarded.

2. *Backface culling*

For those objects composed of faces or polygons, if the object is closed or back face hiding is enabled, the faces or polygons on the other side other size of the object may never face the camera. Those polygons' normal vectors are pointed way from the view point or the camera.

3. Contribution culling

Often, objects are so far away that they do not contribute significantly to the final image. These objects are thrown away if their screen projection is too small. Whereas this culling algorithm has to deal with the final projected image, it is similar to other view-dependence computation. So it is not included as a culling technique in OptiStore's view-dependence determination computation.

4. Occlusion culling

Objects that are entirely behind other opaque objects may be culled. This is a very popular mechanism to speed up the rendering of large scenes that have a moderate to high depth complexity. There are several types of occlusion culling approaches.

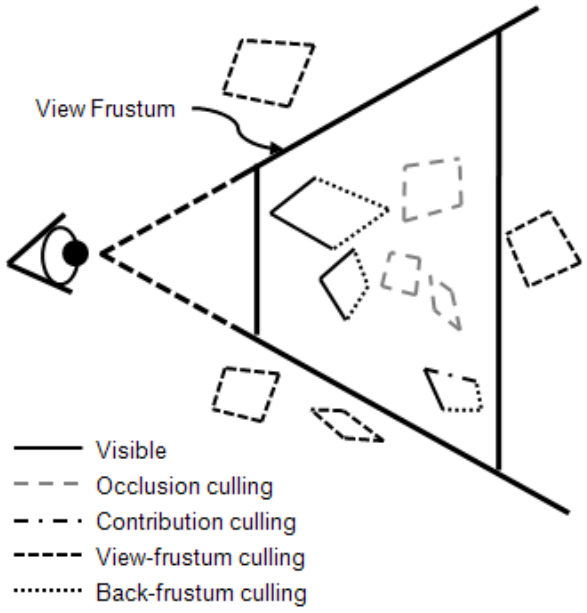


Figure 6-1: Four types of visibility culling techniques

6. 3 Implementation of View-dependent Data Processing

In OptiStore system, for succinctness and generality, the techniques of view-frustum culling and back-face culling were mainly employed. By using hierarchical data structures, like scene-graph, Quadtree, Octree and etc., to organization the partitioned data, OptiStore can cull the geometry data effectively. The culling program just culls the geometry primitives from top to down. If at some level the whole primitive is culled out, then the culling process halts at that level; otherwise continues to descendant level as the client requires. Figure 6-2 shows an example of view-frustum culling on three different resolution levels within a Quadtree in 2D applications.

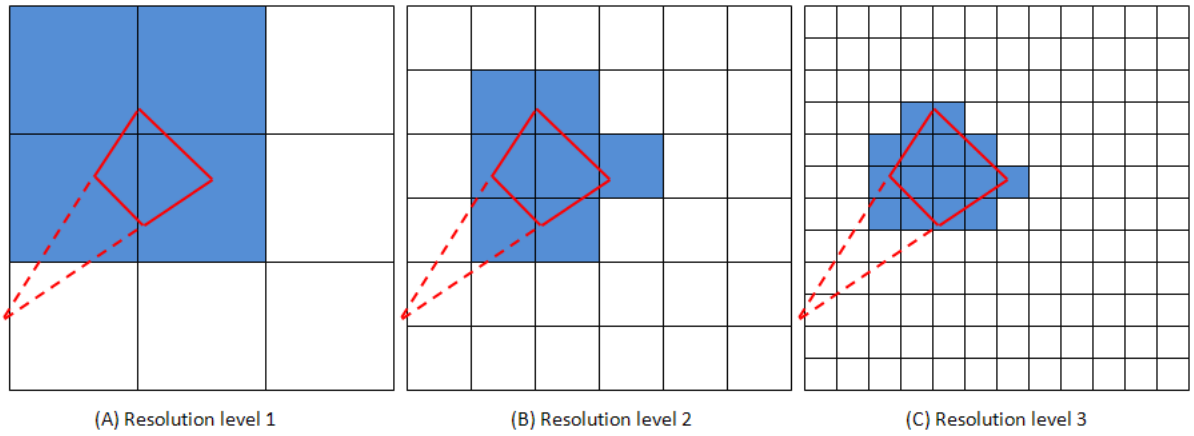


Figure 6-2: View frustum culling at different resolution levels

To fulfill the culling functionality, OptiStore requires visualization applications to pass frustum information through OptiStore client APIs. In most cases, frustum information can be achieved by simply passing model-view matrix $\mathbf{M}$ and projection matrix $\mathbf{P}$ in OpenGL. The model-view matrix and the projection matrix can be retrieved through two

OpenGL functions from the top of model view matrix stack and projection matrix stack respectively:

```
glGetDoublev ( GL_PROJECTION_MATRIX, &m );

glGetDoublev ( GL_MODELVIEW_MATRIX, &p );
```

If a plane in 3D space is noted as a 4-element vector $\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix}$ so that for any point $\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ specified as homogeneous coordinates within the plane, the cross product equals to zero:

$$\mathbf{n} \cdot \mathbf{p} = n_x x + n_y x + n_z x + d = 0 \qquad (6\text{-}1)$$

However, if the point is above the plane, the result of the cross product is positive: $\mathbf{n} \cdot \mathbf{p} > 0$; if below the plane, it's negative $\mathbf{n} \cdot \mathbf{p} < 0$.

Suppose that $\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \\ d \end{bmatrix}$ represents a plane in modeling space and $\mathbf{n}'$ is the corresponding plane in the viewing space after linear transforms multiplied by the model-view matrix $\mathbf{M}$ and the projection matrix $\mathbf{P}$. As proved in Appendix F of [Woo 1999], normal

vectors are transformed by the inverse transpose of the transformation that transforms points. Then we can get the relation between the original plane and the correspondingly transformed plane.

$$\because \mathbf{MPn} = \mathbf{n}'$$

$$\mathbf{P}^{-1}\mathbf{M}^{-1}\mathbf{MPn} = \mathbf{P}^{-1}\mathbf{M}^{-1}\mathbf{n}'$$

$$\therefore \mathbf{n} = (\mathbf{MP})^{T}\mathbf{n}' \qquad (6\text{-}2)$$

Therefore, given the planes bounding view frustum in clipping space, if the model-view-projection matrix is rewritten as $(\mathbf{MP}) = [\mathbf{c}_0,\ \mathbf{c}_1,\ \mathbf{c}_2,\ \mathbf{c}_3]$, the corresponding original planes can be achieved from Equation (6-2) as the result in Table 6-1. Note: all of planes are facing outside the frustum.

Bounding Plane	Bounding Plane Vector	Plane in Modeling Space
Near	$[0,\ 0,\ 1,\ 1]$	$(\mathbf{c}_3 + \mathbf{c}_2)$
Far	$[0,\ 0,\ -1,\ 1]$	$(\mathbf{c}_3 - \mathbf{c}_2)$
Left	$[1,\ 0,\ 0,\ 1]$	$(\mathbf{c}_3 + \mathbf{c}_0)$
Right	$[-1,\ 0,\ 0,\ 1]$	$(\mathbf{c}_3 - \mathbf{c}_0)$
Top	$[0,\ 1,\ 0,\ 1]$	$(\mathbf{c}_3 + \mathbf{c}_1)$
Bottom	$[0,\ -1,\ 0,\ 1]$	$(\mathbf{c}_3 - \mathbf{c}_1)$

Table 6-1: Finding frustum's bounding plane in modeling space

Obvious, if a point is within the view frustum, it implies the point is above all of the frustum's six bounding planes. That means to determine if a point is within the frustum, just check if the cross products of this point with all of the plane vectors are positive. If so, it is

inside the frustum; if all negative, it is outside of the frustum; otherwise, on the frustum plane(s). Then, to determine the relation of an object and the frustum, just check the relation between all of the points within the object and the view frustum. But evidently, this brute-force method is inefficient. Normally, if the bounding primitive (such boxes, spheres, cylinder and etc.) of an object is outside, so the object will be for sure; on the other hand, if the bounding primitive is inside or intersected with the frustum, it will not always be the same for the object. Fortunately, in OptiStore, most of the geometries are box-like; the bounding primitive is the same as the object itself. If the geometry data is not rectangle, we use axis-aligned bounding boxes (AABBs) or oriented bounding boxes (OBBs) to represent those object. Since data structure of these objects is hierarchical, the accurate culling against a view frustum can be gained by testing the descendant sub-boxes recursively like what Assarsson and Möller have done in [Assarsson 2000]. An example of bounding box techniques on a sphere is shown in Figure 6-3. This is implemented for visualizing the geosciences data that is mapped onto the earth with longitude and latitude coordinates.

If back face culling is enabled, the culling process just checks the angle of the normal vector of every geometric primitive within the frustum and the view directional vector. Usually, those normal vectors are assigned when the scene-graph is generated, or calculated based on the shape of the primitive, i.e. the normal vector of a point on the sphere is from the center to that point. And the view directional vector is from the view

point to that point, where the viewpoint $\mathbf{v}' = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ in viewing space. The transformed normal vector in viewing space can be achieved simply by being multiplied by the inverse the inverse transpose of the transformation model-view-projection matrix of the original normal vector. Therefore, we can the cross product of the transformed vectors in view space.

$$\mathbf{c} \cdot \mathbf{n}' = \left(\mathbf{p}^T \mathbf{MP} - \mathbf{v}'\right) \cdot \left(\left(\mathbf{P}^{-1}\mathbf{M}^{-1}\right)^T \mathbf{n}\right) = |\mathbf{c}||\mathbf{n}'| \cos(\alpha) \qquad (6\text{-}3)$$

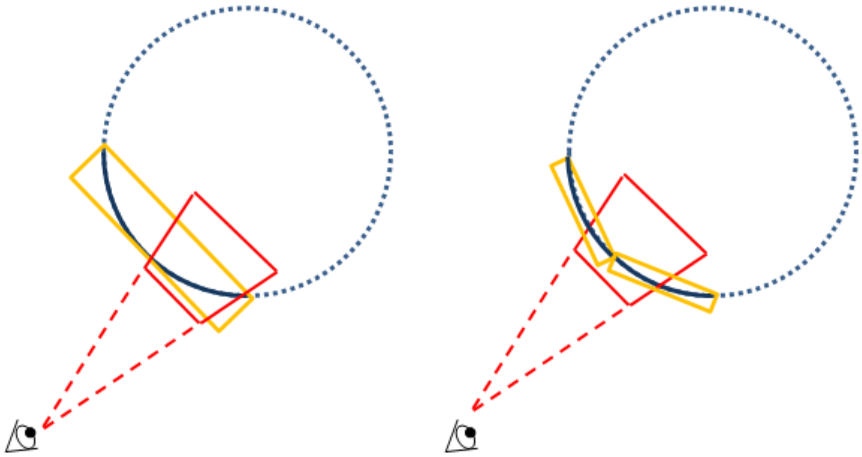


Figure 6-3: Bounding boxes of a sphere for view frustum culling

If all of the angle between all of the normal vectors of the primitive and the view directional vector are greater than 90-degree, then the primitive is visible, which is equivalent to that the cross products are negative; otherwise, not visible (see Figure 6-4).

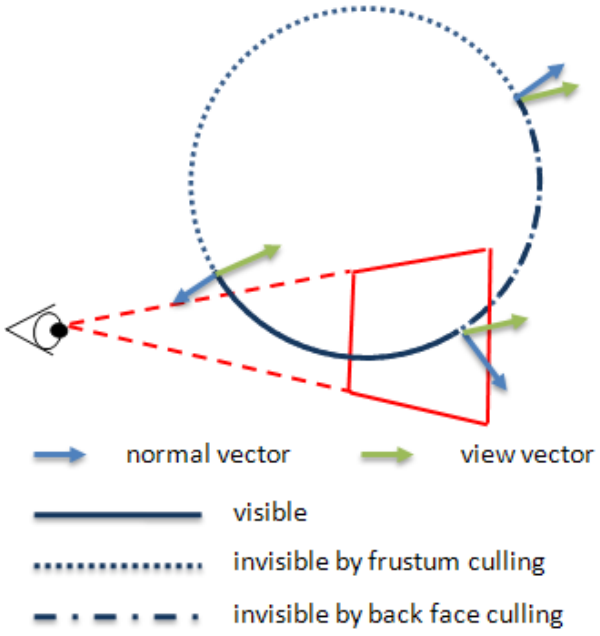


Figure 6-4: Back face culling on a sphere

6. 4 Summary

This chapter explained the reasons that the techniques of view frustum culling and back-face culling are applied to the culling module on OptiStore client and the predictor module for pre-processing and prefetching. These techniques are not only efficient but also less tightly coupled with the rendering pipeline so that the communication overhead between distributed systems can be largely reduced.

CHAPTER 7 REAL-TIME DATA PROCESSING AND CACHING

In distributed applications, reducing the latency is a challenging problem. This chapter mainly describes the techniques in OptiStore that try to minimize the latency cost during data processing and caching.

7. 1 Furthest Object Replacement (FOR) Algorithm

In large scale remote visualization, data is very expensive to access because of several reasons: 1) the capacity of main memory is limited, relative to the extremely large data; 2) the memory size on the graphics card is much smaller and the on-chip bandwidth is very low; and 3) the round-trip communication time is very long. Besides the latency hiding techniques together with other techniques such as multi-resolution analysis and visibility culling, caching techniques are another general strategy to reduce latency, which is similar to the use in processor architecture and web caches.

Caching, as a general concept, has been widely applied throughout computer science. It has proven to be extremely effective in many areas of computing because access patterns in typical computer applications have locality of reference. With well-designed replacing mechanism, caches can help to reduce the data access to the underlying storage or remote buffers by reusing the data. Among the cache replacement algorithms, the least

recently used (LRU) algorithm is still one of the most extensively implemented algorithms for large scale visualization of multi-dimensional data [Bethel 2003; Gao 2005; Guthe 2002; Krishnaprasad 2004; Leven 2002; Li 2002b; Pavlakos 2005; Schneider 2006; Sisneros 2007] as well as the applications in other computer science areas.

Briefly described, LRU algorithm keeps a link-list of data pages in cache. At the back of this list is the least recently used page, and at the front is the most recently used page. If the requested page is found in the linked list, then put the page to the front of the page; otherwise, if the cache has room, the requested page is fetched from the remote buffer or the underlying storage and put to the end of the linked list as the most recently used page; if the cache is full and a new page is requested, the least recently used page will be removed from the linked list and the new block will be fetched and be put to the back of the linked list Because multi-dimensional data is organized as a set of blocks, a hash-table is employed to index those blocks to the pages in the cache. Each block is identified as Figure 4-5. The data structure in LRU algorithm is shown in Figure 7-1: on the left is the hash-table to index blocks and on the right is the linked list of pages with LRU information. The main operations, like insertion, deletion, finding and replacement decision, simply cost $O(1)$ time if the proper data structures are employed. However, it has to treat all of the pages as the same. In real applications, sometimes, blocks or pages are assigned with different priorities, for example, the lowest resolution level block always reside in the memory when it is used for high frequent interaction. Thus, the operations may take much longer time up to $O(n)$. To accelerate these operations, a red-black tree is introduced in some systems [Gao 2005].

With a red-tree tree, the time complexity of find and remove operations are reduced to $O(log(n))$, but the insertion and update operations increased to $O(log(n))$ too.

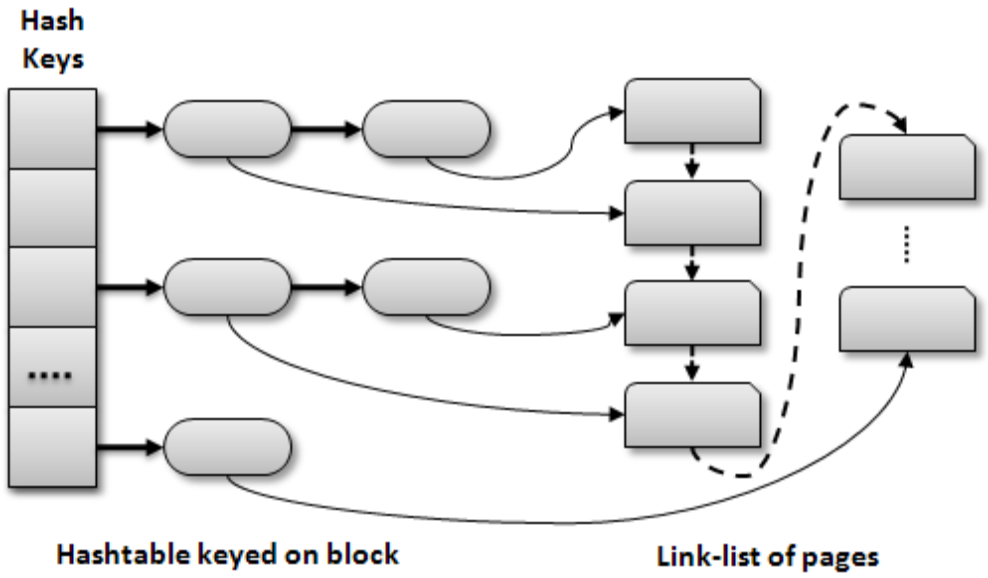


Figure 7-1: The least recently used (LRU) replacement algorithm

Even though LRU algorithm is popular in many computer science areas as well as large scale and distributed visualization, due to the lack of spatial information, it does not work well for caching the multi-dimensional data. Figure 7-2 shows one of the worst scenarios of LRU replacement algorithm for 2D data caching. Suppose that the cache has the capacity of nine pages in memory; the blocks of the bottom three rows are in current view; and the visiting order is from the top-left corner to the bottom-right. Then we can achieve the page visiting order that is noted with numbers in blocks, where the number indicates the rank in the LRU linked list: for example, number 0 indicates the top of the linked list and number 8 the bottom. When the view center moves upwards (from block

with number 4 to the one with number 7 in Figure 7-2), new pages will be requested, and some old pages on the bottom of the linked list should be removed. From the figure, it can be found that nine page faults occur although only three new pages need to be fetched from remote buffer. Similarly, when the view center moves to other directions, except downwards, the LRU algorithm will still cause more page faults than what is actually requested. The high missing ratio will definitely degrade the performance of the cache.

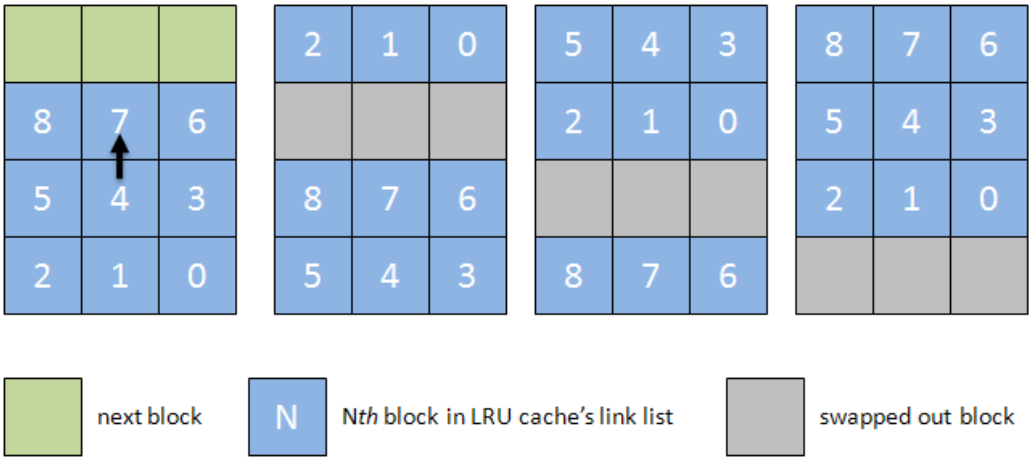


Figure 7-2: The weakness of LRU algorithm with spatial data

Many researchers have already noticed this limitation of LRU replacement algorithm for multi-dimensional data, and proposed different caching algorithms to overcome this limitation. Rhodes et al. [Rhodes 2005] present an iteration aware prefetching algorithm, but the pre-fetching algorithm is only for single dataset and their method requires global iteration information of the dataset so is not suitable for our portioned multi-dataset data organization in parallel computing environment. Other researchers extend semantic caching mechanism to query-oriented spatial data caching [Andrade 2007; Lai 2004; Ren 2000].

Although these algorithms demonstrate much better performance than the LRU algorithm, they are too complicated compared to LRU algorithm because they are dedicated to location or spatial-temporal queries and related operations that occur much less frequently than the data access in visualization applications. Furthermore, these algorithms require a lot of semantic information to determine replacement victims. As OptiStore is a middleware that should be portable enough for different visualization applications, these caching algorithms that are tightly coupled with the query program and depend on considerable information from the client cannot meet the requirement of flexibility.

Thus, we designed a novel caching algorithm – furthest object replacement (FOR) algorithm computation for multi-dimensional data in OptiStore, which has strong spatial locality and fast replacement.

Unlike LRU algorithm, FOR algorithm does not keep a linked list or a red-black tree to maintain the recent use information. On the contrary, it just hash-maps block IDs with pages in memory, shown as Figure 7-3.

As a result, the operations of finding, insertion and deletion will cost only $O(1)$ time. The speed of these operations in FOR algorithm is much faster than those in LRU algorithm. When a cache miss occurs and the cache is full, the algorithm has to determine which data objects to be removed from the cache. Given a requested block ID with information (dataset, resolution-level, d_0, d_1, ..., d_{N-1}), FOR algorithm can determine the replacement victim by the following prioritized criteria: 1) not in the same dataset; otherwise 2) not in

the same resolution level; otherwise 3) the furthest data object. And then the replacement victim block will be evicted from the hash table and the corresponding page from the memory. FOR algorithm provides two types of the geometry distance calculation: Manhattan distance and Euclidean distance.

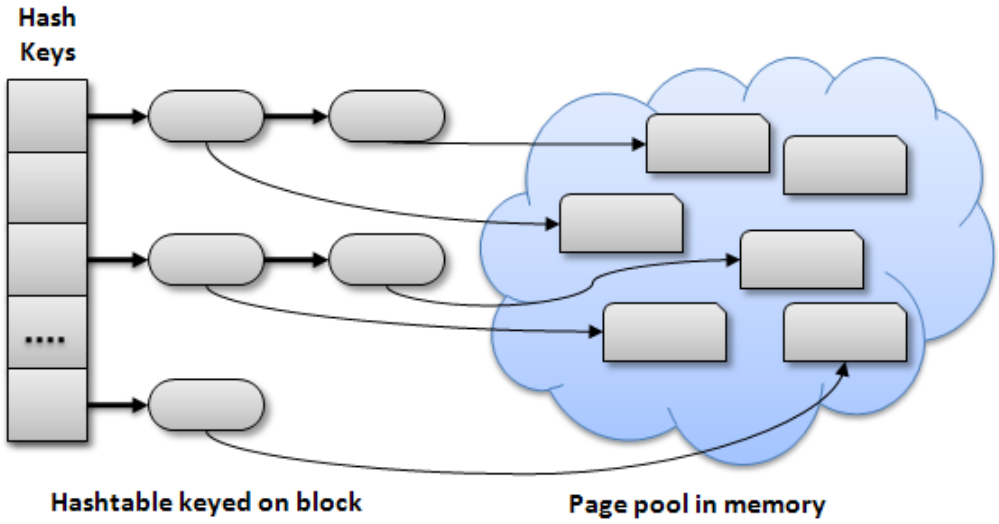


Figure 7-3: Data structure in the furthest object replacement (FOR) algorithm

Manhattan distance between two points in a Euclidean space with fixed Cartesian coordinate system is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. For example, in the 2D plane, the taxicab distance between the point P_1 with coordinates (x_1, y_1) and the point P_2 at (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. The Manhattan distance is also known as taxicab metric, rectilinear distance, or city block distance. A general N dimensional Manhattan distance from block $P = (p_1, p_2, ..., p_N)$ to block $Q = (q_1, q_2, ..., q_N)$ is calculated as:

$$|p_1 - q_1| + |p_2 - q_2| + \cdots + |p_N - q_N| - \Sigma_1^N |l$$

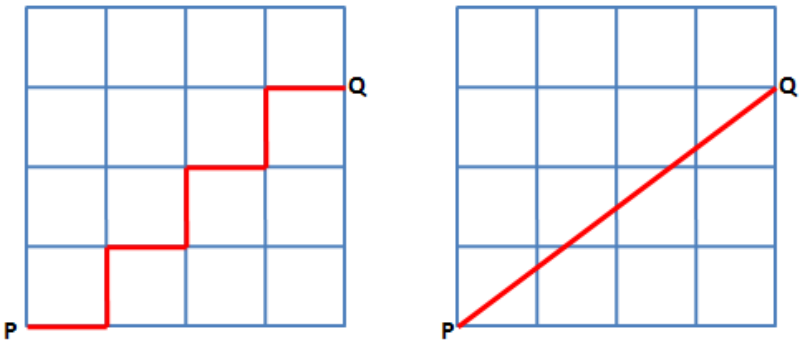
$$|p_1 - q_1| \quad + \quad |p_2 - q_2| \quad + \quad \cdots \quad + \quad |p_N - q_N| \quad = \quad \sum_{i=1}^{N} |p_i - q_i| \tag{7-1}$$

Euclidean distance, also known as Pythagorean metric, between two blocks $P = (p_1, p_2, ..., p_N)$ and $Q = (q_1, q_2, ..., q_N)$ is defined as:

$$|p_1 - q_1| + |p_2 - q_2| + \cdots + |p_N - q_N| = \Sigma_1^N |l$$

$$\sqrt{(p_1 - q_1)^2 \quad + \quad (p_2 - q_2)^2 \quad + \quad \cdots \quad + \quad (p_N - q_N)^2} \quad = \quad \sqrt{\sum_{i=1}^{N} (p_i - q_i)^2} \tag{7-2}$$

A 2D example of Manhattan distance is demonstrated as Figure 7-4 (A) and one of Euclidean distance as Figure 7-4 (B). By default, in OptiStore, due its simplicity and fastness, Manhattan distance is chosen to compute the distance between a pair of blocks within the same dataset and the same resolution level.



(A) Manhattan distance: PQ = 7 (B) Euclidean distance: PQ = 5

Figure 7-4: Examples of 2D Manhattan distance and Euclidean distance

The pseudo code of FOR algorithm is listed as Figure 7-5.

Algorithm FOR:

Input:

B - block ID/coordinate

H - hash-map of blocks and pages

C - cache

Output:

P - corresponding page in memory

```
1:  if H.find(B) == true
2:          return H.get(B);
3:  else {
4:          if C.full() == true {
5:                  min ← ∞;
6:                  foreach b in H {
7:                          if B and b are not in the same dataset or resolution level {
8:                                  victim ← b;
9:                                  break; }
10:                         else {
11:                                 if min > distance(B, b);
12:                                 min←distance(B, b) ;
13:                                 victim ← b; } }
14:                 evict both of victim and its corresponding P; }
15:         C.new(P);
16:         C.getData(P);
17:         H.add(B, P);
18:         return P ;}
```

Figure 7-5: Pseudo-code of FOR algorithm

As listed in Figure 7-5, compared to LRU algorithms with different data structures, the FOR algorithm shows superiority over it on the time complexity of finding, insertion and deletion operations, but it takes more time on the victim determination. In overall measurement, FOR outperforms the LRU algorithm mainly because of the following reasons:

1. The visualization applications access the pages in local cache much more than those out of the cache due to the visualization locality. FOR algorithm spends $O(1)$ time to locate the page, while besides locating page, LRU algorithms has to spend more time to update visiting information.

2. When a block does not exist in the cache, it will take much longer time to access the data on remote server or underlying storage. The eviction decision time is relatively insignificant compared to that in secondary data access. Thus if the missing ratio is lower, longer time eviction decision algorithm will not affect the overall running time too much.

	FOR	LRU with linked list	LRU with tree
Insertion	$O(1)$	$O(1)$	$O(log(n))$
Deletion	$O(1)$	$O(1)$	$O(log(n))$
Locating	$O(1)$	$O(1)$	$O(log(n))$
Victim Determination	$O(n)$	$O(n)$	$O(log(n))$

Table 7-1: Comparison of time complexity of the operations between FOR and LRU algorithms

The following chart shows the result of the experiment that compares FOR algorithm with LRU algorithm (implemented with data structure of red-black tree). The cache size changes from 167MB, to 334MB, to 667MB. The average access time is the latency to fetch a 21MB block from the cache (if the block is missing, fetch it from disk). FOR demonstrates around 20% -30% improvements.

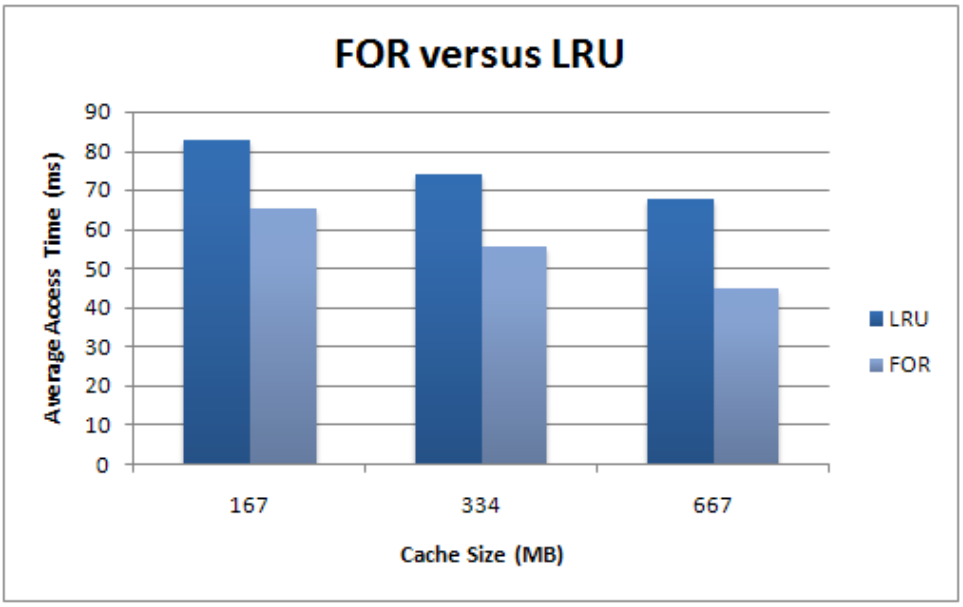


Figure 7-6: Comparison of FOR algorithm with LRU algorithm

7. 2 Data Processing and Caching With Prediction

In previous section, the cache fetches data using a demand-fetch model: when the application calls a load function, the specified datum is fetched from remote server. If the datum is not in the cache, a request is made to the remote system or local shared buffer to fetch the datum. With a good replacement algorithm, the cached data would be likely re-used in a few load function calls. Besides caching data, another general strategy to reduce

latency cost during remote data access is to prefetch the data. By comparison, a data prefetching mechanism attempts to provide data before the application requests that data. As OptiStore processes data on the fly, similarly, a data preprocessing strategy is to filter data before the client application requests to do so. Obviously, if addressed properly, both techniques of data prefetching and data preprocessing can reduce the latency cost during remote visualization largely and both of them can share one prediction mechanism.

Traditionally, the processor cache just prefetches data based on some linear access patterns to overcome the latency within loop instruction blocks. Even though some researchers proposed high accurate prediction methods based on Markov model [Joseph 1999; Nesbit 2005], these models do not perform well to predict multidimensional data access pattern due to the large data size and deep loop instruction blocks. Rhode *et al* present iteration aware prefetching technique for multidimensional data [Rhodes 2005]. But their cache system has to know iteration patterns and access the data within rectangular blocks. On the contrary, most of applications to use OptiStore do not visualize the data in that way. Different prefetching schemes have been employed in many visualization applications [Aliaga 1999; Cohen-Or 2003; Corrêa 2003; Funkhouser 1992; Gao 2005; Gobbetti 2005], but all of them require specific information, like explicit camera position tracks, multi-pass visibility functions, preprocessed occlusion data structures and etc. To provide a succinct and general specific middleware system for different visualization applications, we proposed a new prediction model for data prefetching and preprocessing.

Aforementioned in previous chapters, the visualization application just provides view frustum information for data filtering server via OptiStore client's application interface. Although the camera or view point position can be calculated from the model view matrix inversely, it is still hardly to predict the next camera point without additional information because 1) the movement of the camera is too complicated: given a small incremental movement or angle along each axis, there are twenty-six next possible positions for the camera, another twenty-six for view centers and twenty-six possible view angles of pitch, roll and yaw, let alone the combination of these possibilities; 2) it is too hard to decide those increments: the view result is sensitive to the change of those parameters, due to many factors, like distance, angle, object size and etc. Therefore, to leverage the concept of neighborhood, OptiStore predicts next possibly requested blocks in two ways: one is to predict blocks neighboring to current blocks in the cache within the same resolution level, called intra-level prediction; the other is to predict blocks in adjacent resolution levels, called inter-level prediction.

At the current resolution level, the closest adjacent blocks around the view frustum or currently in the cache should be likely requested later. To find these blocks, we apply the dilation operation on the blocks within the view frustum (or currently in the cache) as in morphological image processing. The indexed block in a *d*-dimensional space can represented as map $f : \underbrace{Z \otimes Z \otimes \cdots \otimes Z}_{d} \to R$, in such a way that every block is assigned a value representing its visibility as a binary multi-dimensional image, where the image set

is $\{0, \quad 1\}$. In the binary image, if the block is in the view frustum (or currently in the cache), it's assigned with 1; otherwise, 0. Let *A* and *B* be sets in Z^d, $d>0$. Let $(B)_x$ denote the translation of *B* by *x* and let $\hat{B}$ denote the reflection of *B* with respect to its origin. The *dilation* of *A* by *B*, $A \oplus B$, is defined as [Gonzalez 2007]:

$$A \oplus B = \left\{ x \mid \left(\hat{B}\right)_x \cap A \neq \Phi \right\} \tag{7-3}$$

Thus, the dilation of *A* by *B* is the set of all *x* displacements of the origin of $\hat{B}$ such that $\hat{B}$ and A overlap by at least one nonzero element. Figure 7-7 illustrates the prediction with 2D dilation of the blocks within the view frustum by a 4-connection operator.

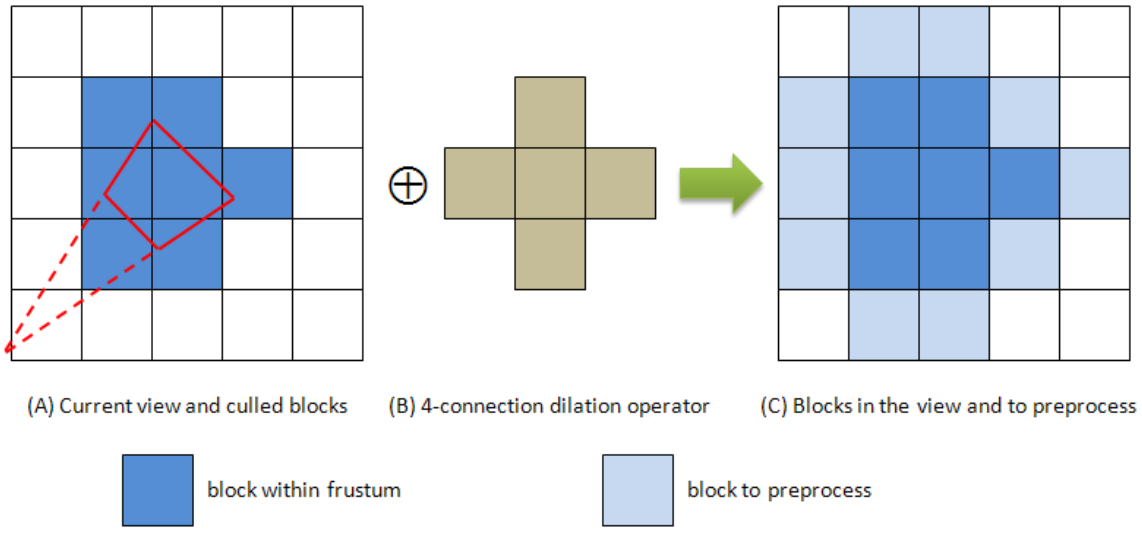


Figure 7-7: Dilation operation for prediction with the same resolution level

To fulfill this operation, the algorithm needs to maintain one array and one hash set (hash key on the block index) to hold the blocks within the view frustum (or currently in the cache) and one array of the predicted blocks. Then for each block in the sorted set, check if each of the corresponding dilation blocks exists in the hash-set; if not, push it to the back of the predicted block array, and insert it into the hash set. Assume the average operation time on the hash set is constant time - $O(1)$ and the size of dilation operator B is constant too with predefined size, the time complexity of this algorithm is linear ($O(n)$) with the number of input blocks. The pseudo-code of this predictor is listed as in Figure 7-5.

Algorithm Intra-Level Predictor:

Input:

A – index array of blocks within the frustum

B – relative index array of the operator block

Output:

P – predicted block array

```
1:  H ←Φ // initialize hash set
2:  P ←Φ // initialize predicted block array
3:  foreach a in A {
4:          foreach b by operation B on a {
5:                  if b not in H {
6:                          P.push_back(b);
7:                          H.insert(b); } } }
```

Figure 7-8: Intra-level predictor

In visualization applications, the next data request may not only occur around current view within the same resolution level, but also in the adjacent resolution levels. In

adjacent levels, above or below (if exist), the possible blocks should also be within the same view frustum. Similar to the intra-resolution-level prediction algorithm, the intra-resolution-level prediction algorithm also examines each block with the view frustum (or currently in the cache). But rather than the blocks around it, the algorithm will check whether its parent blocks for the lower resolution level and child blocks for the higher resolution level are within the same view frustum too by assuming the frustum information is provided. If so, those blocks will be put into the prediction block set. If the frustum information is not available, then all of the parent and child blocks will be added into the set. Figure 7-9 (B) shows the same 2D area of the blocks within the view frustum. Besides the dilated boundary blocks, the blocks in the lower and the higher resolution level within the same frustum are also marked as predicted blocks (see Figure 7-9 (A) and (C)).

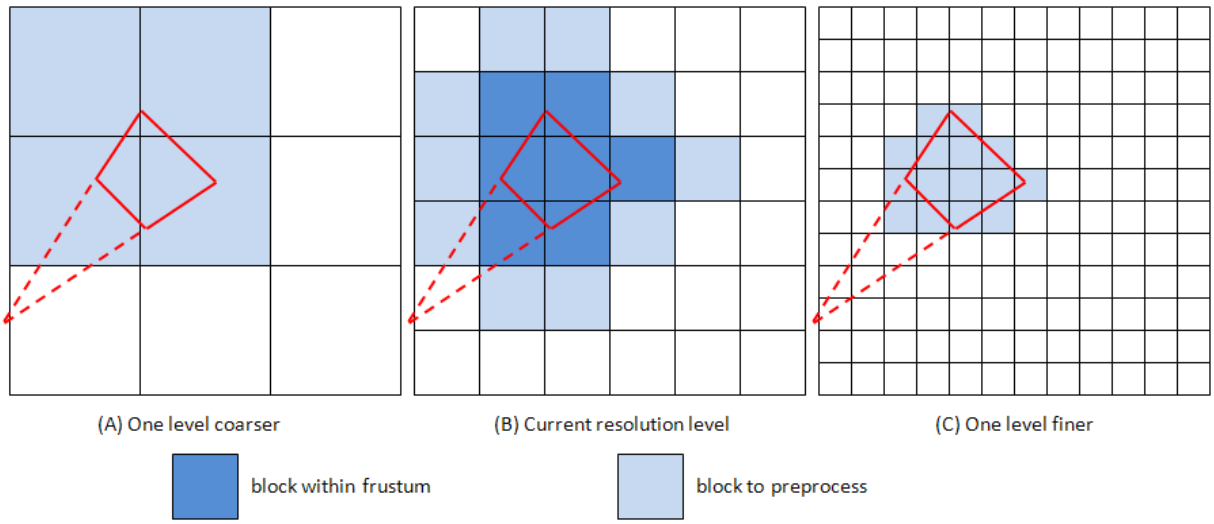


Figure 7-9: Prediction on different resolution levels

Even though both of the algorithms can predict the next blocks to process and fetch, the system has still to decide the blocks of which level to process or prefetch first because of the following reasons:

1. The predictor buffer is limited when compared to the total size of the datasets. If the data to process or fetch exceed the size of the buffer, only the blocks with higher priority will be processed or fetched.

2. The next data request may arrive before the preprocessing or prefetching is completed. So the block with higher probability to be request later will be processed earlier.

3. The data access patterns vary from one application to another. For example, current version Vol-a-Tile renders the volume dataset from the lowest resolution data to the highest progressive, but the pan operation within the same resolution level occurs more often than others in current version JuxtaView when displaying the two dimensional images. The static priority order cannot help to chose the most possible blocks to process or fetch for different datasets, different visualization applications, or different user manners when visualizing data.

In OptiStore, a pair of a dataset UUID and a resolution level within the dataset can be treated as a state in a finite state machine. Consequently, one data request following

another is a transition between two states (if both requests are on the same level of the same dataset, it is defined self transition). Then the prefetching / preprocessing decision problem is transformed to the problem how chooses the most possible next state. Like many similar applications using predictor, such as processor cache [Joseph 1999], web cache[Pons 2003], Google's Page-Rank [Brin 1998] and etc., OptiStore employs Markov model (as known as Markov chain) to predict state transitions.

A discrete time Markov chain is defined in [Bolch 2006]:

Given a finite state space S, a given stochastic process $\{X_0, \quad X_1, \quad \cdots, \quad X_{n+1}, \quad \cdots\}$ at the consecutive points of observation 0, 1 , ... , n + 1 constitutes a discrete time Markov chain if the following relation on the conditional probability mass function (PMF/pmf), that is, the Markov property, holds for all $n \in N_0$ and all $x_i \in S$:

$$\begin{aligned} &\Pr\{X_{n+1} = x_{n+1} \mid X_n = x_n, \quad X_{n-1} = x_{n-1}, \quad \cdots \quad X_1 = x_1\} \\ &= \Pr\{X_{n+1} = x_{n+1} \mid X_n = x_n\} \end{aligned} \tag{7-4}$$

For convenience, discrete time Markov chain is noted as Markov chain in the rest of this document. In the homogeneous case, when the conditional pmf of the process's one-step transition from state *i* to state *j* is independent of epoch *n*, for short notation it can be written as:

$$p_{ij} = \Pr\{X_{n+1} = S_j \mid X_n = S_i\} = \Pr\{X_1 = S_j \mid X_0 = S_i\} \tag{7-5}$$

Starting with state *i*, the Markov chain will go to some state *j* (including the possibility of *j* = *i*), so that it follows that $\sum_j p_{ij} = 1$, where $0 \le p_{ij} \le 1$. The one-step transition probabilities p_{ij} are usually summarized in a non-negative, stochastic transition matrix $\mathbf{P}$:

$$\mathbf{P} = \left[p_{ij} \right] = \begin{pmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ p_{20} & p_{21} & p_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Here, let letters represent datasets and numbers resolution levels, i.e. *A*1 indicating a data request of the blocks on the resolution level 1 of Dataset *A*. Each data request can be considered as a state in the finite state space. Assume the data request sequence on Dataset *A* and Dataset *B* is as the following:

A0, A1, A1, A2, A2, A2, A1, A0, B0, B1, B1, A1, B1, A1, A2, A1, A0

Then from this sequence, we can build a Markov chain transition matrix. In this example, A0 appears three times, but twice followed by other requests; thus the transition probabilities from A0 to A1 and B0 are half respectively, which are shown in the first row of the matrix (see Figure 7-10).

	A0	A1	A2	B0	B1
A0		1/2		1/2	
A1	2/6	1/6	2/6		1/6
A2		2/4	2/4		
B0					1/1
B1		2/3			1/3

Figure 7-10: An example of Markov chain transition matrix

The corresponding Markov chain is also built as a directed graph in Figure 7-11, where the nodes are the data requests and weighted links represent probabilities of transitions.

Figure 7-11: Markov chain representing data access patterns via transition probabilities

Therefore, given the current state (data request) in a Markov chain, the predictor can sort the next possible states, where the weights of the outgoing links from this current state to these successors are descending. If there exists none of outbound links, for instance, a first time data request, the predictor just simply checks the current resolution level in the same dataset and then lower and higher resolution levels.

In OptiStore, because the transition matrix is likely to be a sparse matrix, a data structure of adjacent list is used to build the directed graph of a Markov chain. To increase processing speed and reduce latency cost, we employ process parallelism and thread parallelism in the data filtering server when implementing predicted data processing and caching. The integration of the predictor and the filter modules is illustrated in Figure 7-12.

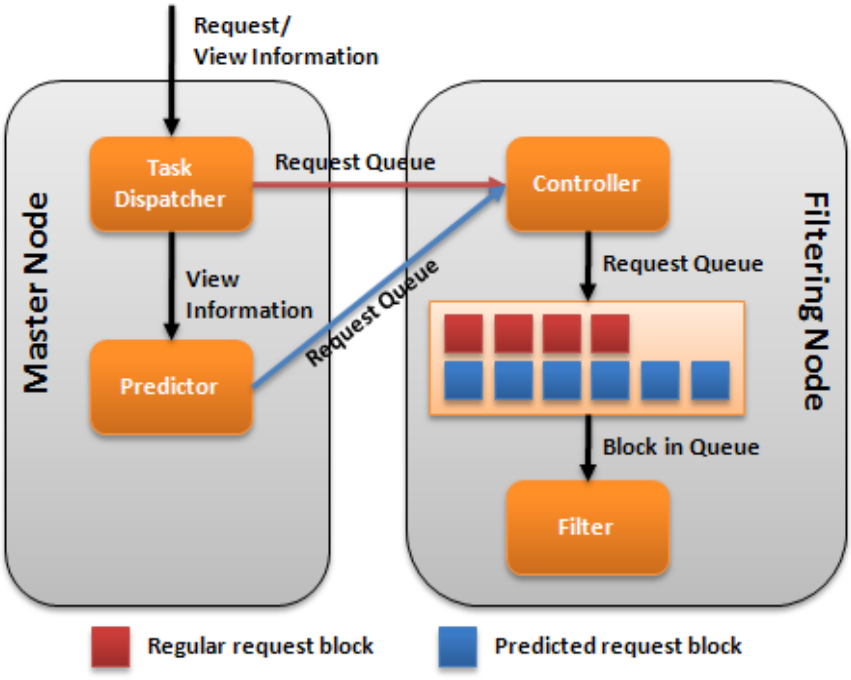


Figure 7-12: Integration of predictor with data filtering and caching

Once the request with view information arrives at the master node, the task dispatcher will find the blocks to filter, push them into a queue and deliver the queue to the filtering nodes. It will also pass the view information to the predictor. The predictor calculates the next possible blocks to filter, updates Markov chain and pushed the predicted request blocks to the filtering nodes. The main controller is in charge of receiving the request queues and updating the local processing queues. The filter module processes the blocks in this local processing queues, where regular request has higher priority that predicted request. The controller and the filter work as a producer and a consumer in the producer-consumer model. The request queue is like the shared buffer in that model.

7. 3 Implementation of Parallel Data Processing with Remote Memory Access

According to the data partition and distribution scheme in OptiStore, once the dataset loading is initialized, the partitioned blocks of the hierarchical data structure are distributed statically among the server nodes. For some in-place operations, which lead to the result as the same block as the source block, such as element conversions (i.e. RGB data to gray data), isosurfaces extraction, etc., the result blocks are just transferred within the same server node. But for the operations like multi-resolution filtering, it is inevitable to transfer the data between server nodes. During this kind of operations, this intra-node data transfer takes place among the nodes so frequently that a remote memory access interface should be applied for these operations.

In OptiStore, the nodes that filter the data and write the result are called source nodes; and the nodes that receive the intermediate result sub-blocks and merge them into the final blocks are called target nodes. We use the same 2D example in Chapter 5, illustrated as Figure 5-6. Four blocks are transformed twice and the result sub-blocks are one sixteenth of the target block; and three blocks are transformed once and the result sub-blocks are one quarter of the target block (see Figure 7-13(A)). It is noticed that each sub-block cannot be written directly to the target block buffer at once due to the fact that multi-dimensional data are stored in memory as one dimensional array. Therefore it will increase much latency cost if writing sub-blocks directly to the remote target block pixel by pixel or line by line because the overheads during remote memory access will increase dramatically.

To minimize the times of remote memory writing operation, we allocate a temporary local buffer on the target node and expose it as a shared memory to the whole cluster. This buffer is partitioned into continuous segments, each of which is to store one sub-block. Figure 7-13(B) shows this buffer. At the end of each segment, a sub-block tag (drawn as a tiny red box) is attached. Each tag includes the sub-block coordinate information, writing complete byte, etc. Besides additional sub-block tags, the size of each segment is the same as the corresponding sub-block. In this way, sub-blocks can be written to the target node at once.

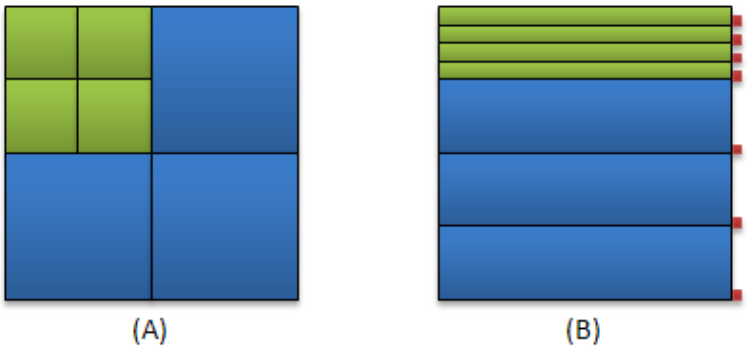


Figure 7-13: Multi-dimensional data sub-blocks mapped to one-dimensional array in memory on the target node

Based on this data structure and remote memory access strategy, the work flow of the remote memory access during data processing is illustrated in Figure 7-14. There are three types of modules related to remote memory access:

1. The modules on the master node. Once the filtering request arrives at the task dispatcher module, it searches the source blocks according to the global hierarchical information (as the example in Figure 5-6) and broadcasts the source and target block information to the corresponding nodes. After filtering finishes, the task dispatcher notifies all of the nodes to free the shared buffer.

2. The modules on the source nodes. As the main controller receives the broadcasted source block information from the task dispatcher, it passes this request to the data fetcher and the filter, and allocates a shared memory, which is just synchronization with the shared memory allocation on the target nodes. After data filtered, the filter writes the sub-blocks together with an additional

tag to the target node via the shared memory sender. The filter and the shared memory sender work parallel in the producer-consumer paradigm.

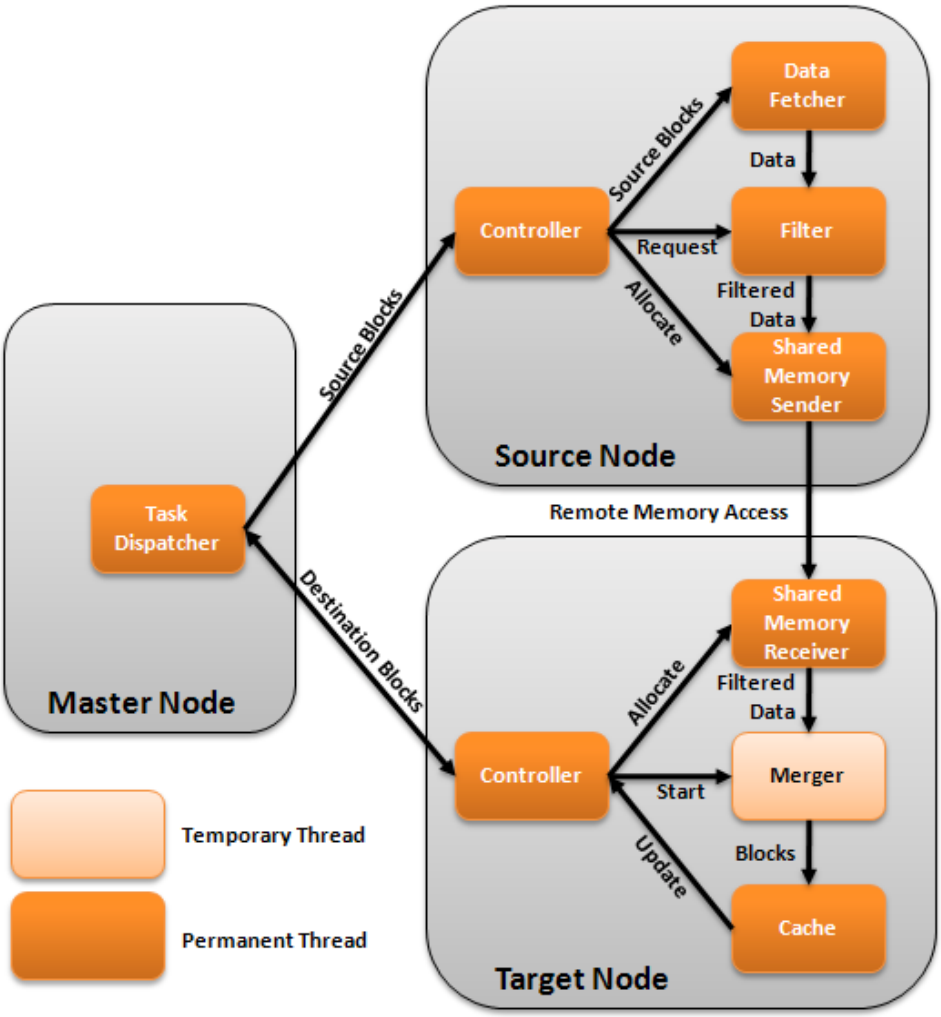


Figure 7-14: Remote memory access on OptiStore data filtering server

3. The modules on the target nodes. When the main controller receives the request from the task dispatcher on the master node, it allocates a buffer as the one in Figure 7-13(B) and starts a new thread - the merger. The merger will check the tags of each segment. If the writing completion byte changes, the merger then

convert the one-dimensional array to multi-dimensional sub-block according to the coordination information of that sub-block. After all sub-blocks have been converted, the target block is inserted into the cache and the cache notifies the master node of the information of the newly filtered blocks and replaced blocks. Again, the merger and the shared memory receiver work parallel and synchronized as the producer-consumer model.

Although current MPI implementation provides data transfer between cluster nodes, we still develop our own remote memory access interface over the standard MPI functions [Gropp 1998] due to the following facts:

1. One-sided communication mechanism should be applied because a source node itself may also be a target node of other nodes; and a target node may also filter data for another node. The other modules do not necessarily care about the parameter, synchronization and timing the data transfer operations. Therefore, the functions, such as MPI_Send / MPI_Recv / MPI_Isend / MPI_Irecv, do not fit in remote memory access.

2. In order to exploit high parallelism, asynchronous and passive data transfer mode is preferred because the synchronous and active data transfer requires synchronization overheads per transfer operation, and both data transfer parties, sender and receiver, have to explicitly involved with the data transfer. So we cannot choose the active paradigms of MPI one-sided communication.

3. The current MPI implementation of one-sided passive communication requires a lot of synchronization, especially for the write operation. Although the functions of MPI_Win_lock and MPI_Win_unlock work in shared and exclusive mode, the lock granularity is too large – the lock is applied to the whole shared buffer on the target node. Considering that the sub-blocks from all source nodes do not overlap on the target buffer, we do not apply any lock on the shared buffer.

In the implementation of this remote memory access interface, when the singleton RMA object is generated, a shared memory sender thread and a shared memory receiver thread will start. All of the nodes are fully connected via these threads. Every time when a new shared buffer is allocated, a unique id associated with the shared buffer will be generated and registered on the RMA object among all of the nodes. RMA read and write operations on the shared buffer then are equivalent to receiving and sending data on that buffer on the remote node with the same buffer id.

We compared OptiStore RMA with MPICH2 [MPICH2] RMA by writing 21MB data from 16 source nodes to 1 target node with 16, 64 and 256 sub-blocks respectively on a cluster interconnected with 1Gpbs Ethernet network. The result in Figure 7-15 shows that OptiStore remote memory access out-performs RMA application with one-sided communication in MPICH2 implementation, especially when the number of sub-blocks increases (that is when the number of write operation becomes large).

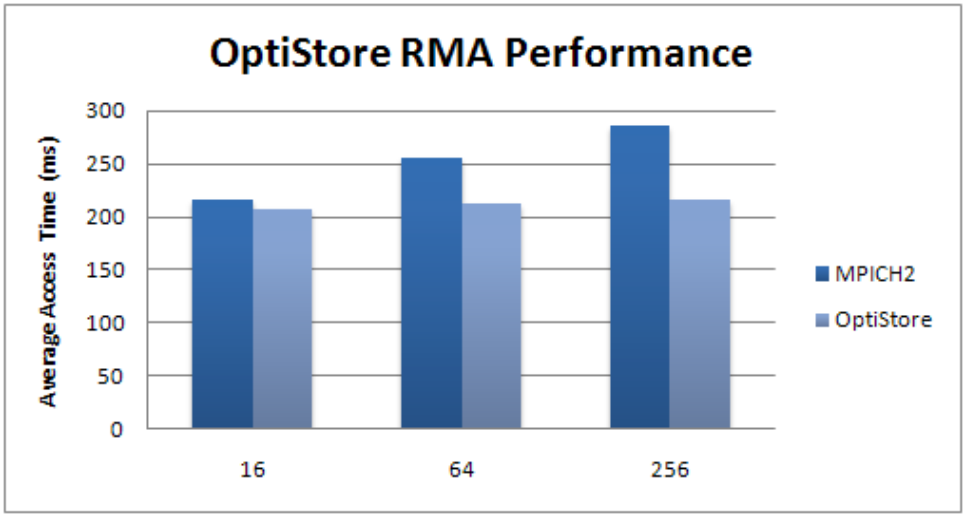


Figure 7-15: OptiStore RMA performance versus MPI RMA

7. 4 Summary

In this chapter, we elaborate the core techniques in OptiStore: data caching and data processing with prediction, parallel data processing, and remote memory access.

A novel caching replacement algorithm - Furthest Object Replacement algorithm is presented and the experiment is tested to compare it with the popular algorithm – least recently used replacement algorithm.

A Markov chain predictor can predict several next request data based on prior operation history and data access pattern.

A new remote memory access implementation has been introduced. It outperforms the counterpart of the standard MPI implementation – MPICH2.

CHAPTER 8 EXPERIMENTAL STUDY

In this chapter, we will present the hypotheses and performance models for OptiStore architecture and describe the details of the experiments for these models. Some experiment results will be presented and studied. There are mainly two objectives of the experiments: one is to analyze how OptiStore architecture can meet the requirement of scalability and interactivity for large scale visualization applications; the other is to find the major factors that affect the on-demand data processing performance.

8. 1 Hypotheses & Predictive Models

The goal of OptiStore is to provide on-demand data processing service for very large scale interactive visualization. Compared to the predominant data preprocessing paradigm, before we build up an experiment model, we introduce the following hypotheses:

1. It is possible to conduct near real-time filtering fast enough to eliminate the need of data preprocessing for very large scale interactive visualization.

2. This system will be scalable enough as the size of dataset and/or the number of rendering node increases

3. This system can process distributed datasets.

Therefore, to sum up these hypotheses, we can have one question: given specific computation power and network bandwidth, how fast is the data processing? That can be written as a formula to indicate the relationship of access time, dataset size, filtered data size, computation power and network bandwidth:

$$T = F\left(S_{data},\quad S_{fdata},\quad P,\quad BW_{network}\right) \tag{8-1}$$

As described in the previous chapters of this dissertation, the middleware system consists of many components. For the sake of simplicity and generality, we create an abstract model for the whole data processing and caching system, illustrated as Figure 8-1.

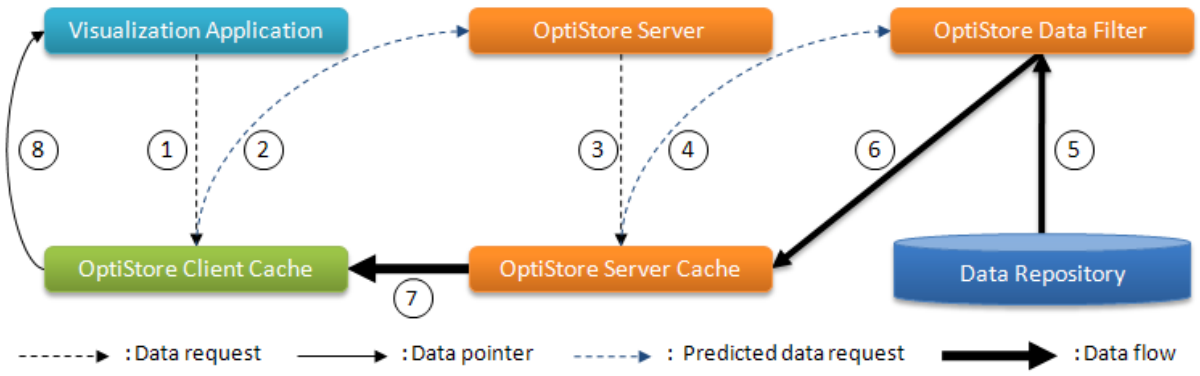


Figure 8-1: Abstract data access model of OptiStore

This abstract model can be considered as a three-level caching system. The work flow can be described as follows:

1. The visualization application requests data from OptiStore client cache via OptiStore client interface. It takes T_{ccache} time. If data exists, called a hit, go to

step 8; otherwise, it will be a miss and then go to step 2. The missing ratio is noted as R_c, which satisfies $0 \le R_c \le 1$.

2. OptiStore client sends data request to OptiStore Server. Suppose it takes T_{cs} time.

3. Server requests data through server cache. The time cost is T_{scache}. If data exists, called a hit, go to step 8; otherwise, it will be a miss and then go to step 7. The missing ratio is noted as R_s, which satisfies $0 \le R_s \le 1$.

4. OptiStore server cache sends data filtering request to OptiStore filtering server nodes. Suppose it take T_{sf} to pass the request from the server cache to the filter on other nodes in the same cluster.

5. OptiStore filter gets the data. The data access time is $T_{repository}$ with missing ratio R_r, which satisfies $0 \le R_r \le 1$;.

6. OptiStore filter processes the data and writes the requested data into server cache. The total time cost include filtering time T_{filter} and intra-cluster remote memory access time T_{fs}. Since the pipeline of filtering and remote memory access is implemented parallel, it is hard to separate them. Thus, we use T_f to represent both of them.

7. OptiStore server sends back the requested data to OptiStore client. The time cost is T_{sc}.

8. Local cache returns data pointer to visualization application. The time cost is T_{return}.

Thus, to this three-level cache system, we can conclude a formula for average access time, similar to the two-level cache in [Hennessy 2006]:

$$T_{average} = \left(T_{ccache} + T_{return}\right) + R_c \times \left(\left(T_{cs} + T_{scache} + T_{sc}\right) + R_s \times \left(T_f + T_{sf} + R_r \times T_{repository}\right)\right) \quad (8\text{-}2)$$

In this model, because the operations with relatively small data structure occur locally, the time consumed on step 1, 3, 6 and 8 is far less than others. That says that $T_{ccache} << T_{average}$, $T_{scache} << T_{average}$, $T_{sf} << T_{average}$ and $T_{return} << T_{average}$. They will be omitted in Formula 8-2 later. Additionally, T_{cs} could be considered as a constant that nearly equals to round-trip time (RTT).

As to T_{sc} and $T_{repository}$, they are basically the size of data (original or filtered data) and the bandwidth function:

$$T_{sc} = \frac{S_{fdata}}{BW_{sc}} \quad (8\text{-}3)$$

$$T_{repository} = \frac{S_{data}}{BW_{rs}} \tag{8-4}$$

Suppose a single processor can filter the whole data S_{data} with the data size P in unit time, which represents the computation power $P(S_{data})$, then a parallel computing cluster with N same processes can speed up the processing capability to $\alpha(N)\cdot P$. $\alpha(N)$ satisfies Amdahl's law [Grama 2003] so that it would not be greater than a linear function. Then from Equation 8-3 and 8-4, Formula 8-2 can be rewritten as follows:

$$T_{average} = R_c \times \left(\left(C + \frac{S_{fdata}}{BW_{sc}} \right) + R_s \times \left(\frac{S_{data}}{\alpha(N)\cdot P} + R_r \times \frac{S_{data}}{BW_{rs}} \right) \right) \tag{8-5}$$

We noticed that in Formula 8-5, R_r and R_s are related to the aggregated cache size of the whole cluster (in other words, the number of processors), the performance of caching replacement algorithm and the accuracy of prefetching and processing prediction.

Furthermore, we have two assumptions for this model:

1. Since OptiStore client and visualization applications are implemented in a pipelined parallel model. And the data size is restricted in visualization application due to the limitation of the size of main memory and graphic memory. With this latency hiding techniques, we can replace S_{fdata} in Formula 8-5 with the size of the first several available blocks: $\beta \cdot S_{block}$ if the total requested data size is controlled.

2. The latency cost and bandwidth utilization in wide area network is considered to be stable with the implementation with new OptIPuter software components, such as LambdaStream [Xiong 2005] and LambdaBridge [Wang 2006]. In most cases, the constant C, approximated as RTT, is at several hundred millisecond level.

Hence, Formula 8-5 is also suitable for wide-area-network applications, and transformed to:

$$T_{average} = R_c \times \left(\left(C + \frac{\beta \cdot S_{block}}{BW_{sc}} \right) + R_s \times \left(\frac{S_{data}}{\alpha(N) \cdot P} + R_r \times \frac{S_{data}}{BW_{rs}} \right) \right) \quad (8\text{-}6)$$

Therefore, due to the immaturity of those network software components and hardware facilities, we will build up experiments within local area network and project results into the application in wide area network.

Table 8-1 indicates how our proposed approaches affect the variables in this predictive latency model and ultimately improve overall performance of OptiStore.

Approaches	Effects
Load balancing	$R_S \downarrow, \quad \alpha(N) \cdot P \uparrow, \quad R_r \downarrow$
Multi-resolution	$R_c \downarrow, \quad \beta \cdot S_{block} \downarrow, \quad S_{data} \downarrow \Rightarrow R_S \downarrow, \quad R_r \downarrow$
View-dependent	$R_c \downarrow, \quad \beta \cdot S_{block} \downarrow, \quad S_{data} \downarrow \Rightarrow R_S \downarrow, \quad R_r \downarrow$
Run-time Preprocessing	$R_c \downarrow, \quad R_S \downarrow, \quad R_r \downarrow$

Table 8-1: Proposed approaches and their effects on the predictive model

8. 2 Experiments

8.2.1 Experiment hardware, software and data

To study the performance and scalability of OptiStore, we tried several experiments upon OptiStore system on the cluster – YORDA.

YORDA consists of 30 nodes. Each node has one AMD® Opteron™ 2GHz Processor, 3GB memory and the nodes are interconnected with 1Gbps Ethernet network cards. The operating system is SUSE® 10.3 and the kernel is Linux of version 2.6.16.27-0.9-smp.

The middleware of OptiStore has two parts: OptiStore Client and OptiStore Server. Both API is compiled, built and running with POSIX Threads library and Boost C++ library. OptiStore server also runs on MPI library. In our experiments, the MPI environment is MPICH2.1.04 with multiple thread support.

Since our experiments were mainly to measure the latency of multi-resolution filtering that takes O(n) time linearly to the size of different dimensional datasets, we just tested the two dimensional datasets in our experiments. The original dataset is around 11GB Blue Marble images from USGS. The dataset consists of 32 by 16 cells, each with dimension of 2700 by 2700 and 3 bytes (RGB) per pixel. The whole single image could be 86400 by 43200 by 3. The total size of the dataset is of 11,197,440 kilo-bytes. In order to validate the scalability of the system, in our experiments, we did up-sampling over the original data up to 89.6GB.

8.2.2 Experiments

The OptiStore server program runs on the master node of YORDA and four and sixteen slave nodes (for different experiments, the slave node number varies). For different experiments, OptiStore data filtering server processes and caches different size datasets. In our experiments, there are mainly three sizes of dataset: the original dataset of Blue Marble dataset, one quarter subset of the images from the original dataset and one sixteenths out of the original dataset.

On the client side, a parallel visualization program (a parallel version virtual-earth-like program) runs on a client cluster (still on YORDA, a portion of the nodes) and accesses the dataset on remote server during the rendering through OptiStore client API per blocks (from aforementioned calculation, each block is around 21MB) in a real-time processed multi-resolution pyramid.

Therefore, the experiment model becomes:

$$T_{average} = R_c \times \left(\left(C + \frac{\beta \cdot S_{block}}{BW_{sc}} \right) + R_s \times \left(\frac{S_{data}}{\alpha(N) \cdot P} + R_r \times \frac{S_{data}}{BW_{disk}} \right) \right) \qquad (8\text{-}7)$$

8. 3 Results and Discussion

1. Overall access latency versus data size

First we measured the overall access time versus original data size with 16 processing nodes, compared against the results in the worst case and the base case. This

allows us to prove that even though the data size is increasing, the overall access time is still acceptable. The worst case is that all of the local cache assesses miss; that is all the missing ratios in Formula (8-7) equal 1.00. On the contrary, the best case means that it does not take any time to filter data on the server side. Correspondingly, the missing ratios on server side are all zeros. In the experiments, a 4-node visualization application ran on the client side and fetched data through OptiStore client with 512MB client cache. The OptiStore data filtering server consisted of 16 processing nodes and each node contributed 1GB cache to the whole cluster shared buffer.

In Figure 8-2 and Table 8-2, we can see that when the dataset sizes are below 11GB, the access time is almost constant and almost equals to the results in the best case; even if the dataset size is above 11GB, the average access time is still close the best case. This is because that when the whole original data can be held in the allocated buffer of filtering server, it takes little time to fetch the original data. If the data size exceeds the total size of the shared buffer on the filtering server, the overall latency will increase but still stay close to the results in the best case and much better than those in the worst case.

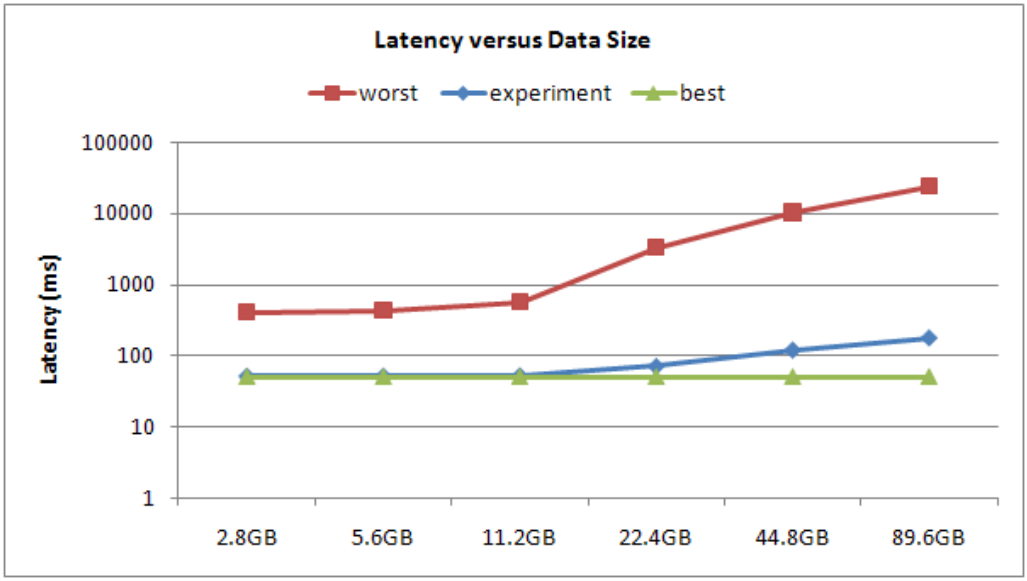


Figure 8-2: Access latency versus data size

	average access time (ms)					
data size (GB) / scenarios	2.8	5.6	11.2	22.4	44.8	89.6
Best	50.0	50.0	50.0	50.0	50.0	50.0
Experiment	52.2	52.6	53.0	71.6	120.9	178.9
Worst	409.7	434.1	566.7	3339.9	10269.5	24000.3

Table 8-2: Average latency versus data size

2. The number of filtering processors versus data size

When we set the maximum access time to 1 second so that the user can get 1 frame-per-second response, we measured how many nodes at least should be employed to run the data filtering services on the server. The experiments ran as the same as the previous experiment. The similar result as previous experiment occurs for the in this scenario – for example, when data size is small so that the server buffer can stored the original data and processed data, the numbers of processing node at least require are as

same as those in the best case; on the other hand, as data size grows, the number required will remain close to the numbers the best case for OptiStore while the number in the worst case increases promptly. This means that OptiStore scales well (see Figure 8-3 and Table 8-3).

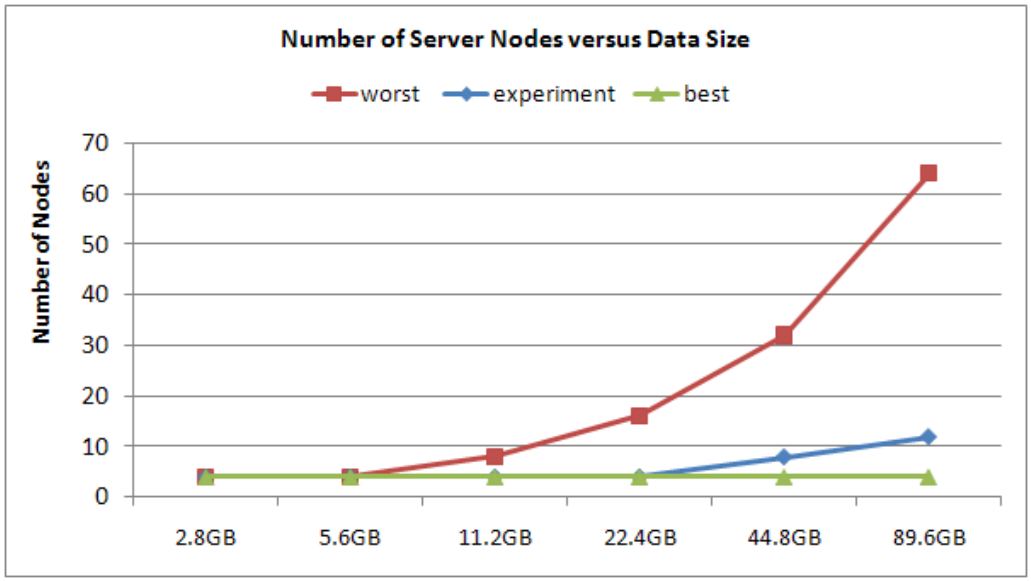


Figure 8-3: Number of filtering processors versus data size

	Number of processing node required					
data size (GB) / scenarios	2.8	5.6	11.2	22.4	44.8	89.6
Best	4	4	4	4	4	4
Experiment	4	4	4	4	8	12
Worst	4	4	8	16	32	64

Table 8-3: Number of filtering processors versus data size

3. Average access time versus cache size on filtering server

The previous two experiments demonstrate the jumps in access time when the cache is small. To watch the relationship of data access time and cache size, we run a 16-node OptiStore server program and a 4-node OptiStore client program. The server feeds and filters 2.9GB dataset – the medium size dataset in the previous experiment. On the server side, the total cache of the whole server changes from 50 percent of the dataset size to 150 percent. Since the newly filtered data also resides in the cache, the cache sizes over 100 percent of the dataset is still included. In Figure 8-4 and Table 8-4, the average access times are close. This result can be explained that the simultaneous heavy processing and disk IO access increases miss ratio and the intra-server access time dramatically.

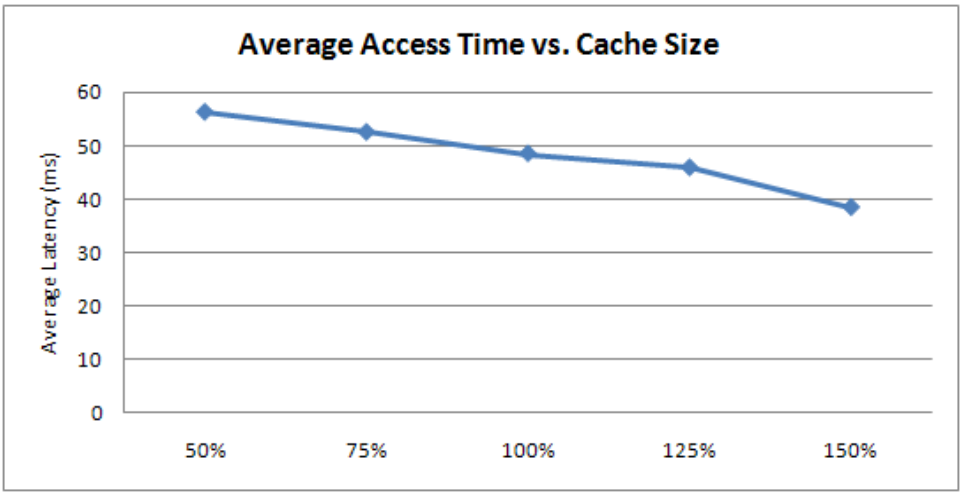


Figure 8-4: Average access latency versus cache size

cache size (compared to data size)	50%	75%	100%	125%	150%
average access time(ms)	56	53	48	46	38

Table 8-4: Average access time versus data server cache size

4. Average access time versus number of clients

In order to measure the access time when client number increasing, eight filtering server nodes provide the 2.8GB dataset service. From four to sixteen node clients connect to the server. Figure 8-5 plots the relation of the average access times with different client node numbers (see the result in Table 8-5). Since the data filtering is processed in batch – the cells that the whole client cluster requests are processed parallel at once; the main increment of the access time relies on the network traffic. When the client number overpasses the server too much, the network traffic may affect the access time obviously because two nodes of the client may connect to one server node at the same time.

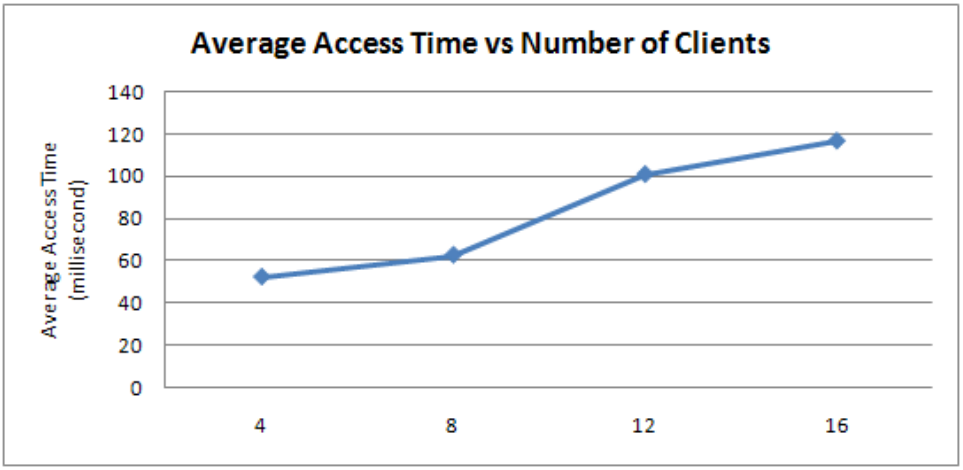


Figure 8-5: Average access latency versus number of clients

client processor number	4	8	12	16
average access time (ms)	65	76	114	130

Table 8-5: Average access time versus number of client processors

8. 4 Summary

In this chapter, first we introduce three hypotheses that assume OptiStore meet the requirements of current very large scale visualization and data processing. Then we deduce a theoretical model for this system. At last but not least, a series of experiments have been performed on this model. These experiments have proved that our model is valid and effective.

CHAPTER 9 CONCLUSION AND FUTURE WORK

In order to satisfy the requirement of scalability, interactivity, timeliness and flexibility in large scale visualization, OptiStore – a data management middleware is designed, developed and implemented. Compared to current predominate data preprocessing systems, OptiStore is an on-demand runtime processing framework for large scale data intended to minimize the need to manage extraneous pre-processed copies of the data that will become a major problem as we continue to amass vast amounts of data. This dissertation describes the theories, models and techniques employed on this system, and demonstrates the experiment results to prove that it is an effective and efficient system.

9. 1 Contributions of OptiStore

OptiStore is designed and implemented as an on-demanded data management system for very large scale interactive visualization. It addresses several problems of data service for visualization application in OptIPuter architecture: scalability, interactivity, timeliness and flexibility. Compared to other data management systems, this dissertation features the following contributions:

- Design a flexible data filtering middleware model for visualization applications. This dissertation presents a distributed high-performance data filtering model. In

this distributed computing model, high-performance data processing system may be separate from both data repository systems and visualization systems so that it affords more flexibility and power to visualize very large datasets. Especially, OptiStore is such a system that also supports interactive scientific visualization applications.

- Develop OptiStore as an on-demand data processing system. OptiStore aims to provide data querying, filtering and processing services on the fly so that various interactive visualization applications can benefit from this near real-time system. In this dissertation, we have implemented several techniques to minimize the latency: load-balancing data partition and organization, multi-resolution filtering, view-dependent data processing, fast remote memory access, a Markov chain predictor, etc.

- Realize OptiStore as a scalable data filtering system. The techniques, such as load-balancing data partition, multi-resolution filtering, multi-dimensional caching, etc., were designed to load and process very large high-dimensional datasets.

- Introduce a novel cache replacement algorithm for multi-dimensional datasets. The new algorithm was designed for multi-dimensional data with the spatial indexing information in the whole system. It shows superiority over the popular Least Recently Used (LRU) algorithm.

- A predictive model was built to measure the performance of OptiStore system. A series of experiments have been conducted to prove the effectiveness and efficiency of OptiStore.

- Implement OptiStore as a flexible extensible framework. The middleware system is designed with design patterns and developed with object-oriented programming language. Various data formats, data filters and caching algorithms can fit into the framework. New data models, data filters, caching mechanisms and networking transfer protocol can be easily added.

9. 2 Future Work

As OptiStore is an extensible data processing framework, many new data models and filters can be added to this system.

- To extend the multi-resolution filtering to other datasets: geometries, unstructured grids/points, vectors and so on.

- Programmable Graphics Processing Unit (GPU) hardware and software is becoming mature. The techniques of General-Purpose computing on GPU (GPGPU) can be applied as data filters to scientific data. Due to its inherent parallelism and computing power, it outperforms CPU on many scientific data computation. The research on GPU cluster and streaming is a hot area in

computer science. It will be an interesting topic to apply GPGPU to OptiStore filtering server.

- Future high-speed OptIPuter networking middleware components, such as LambdaStream and LambdaBridge can be integrated into OptiStore for the visualization applications over wide area network.

CITED LITERATURE

[Abgrall 1998] R. Abgrall. "Multiresolution Representation in Unstructured Meshes." *SIAM journal on numerical analysis* **35**(6): 2128, (1998).

[Ackerman 1998] M. J. Ackerman. "The Visible Human Project." *Proceedings of the IEEE* **86**(3): 504-511, (1998).

[Adelson 1991] E. H. Adelson and J. R. Bergen. "The Plenoptic Function and the Elements of Early Vision". Computational Models of Visual Processing. M. L. a. J. A. Movshon, MIT Press, (1991).

[Agarwal 1997] P. K. Agarwal, L. J. Guibas, et al. "Cylindrical static and kinetic binary space partitions". In *Proceedings of the thirteenth annual symposium on Computational geometry*, Nice, France, ACM Press, (1997)

[Agerwala 2006] T. Agerwala and M. Gupta. "Systems research challenges: a scale-out perspective." *IBM Journal of Research and Development* **50**(2/3): 173-180, (2006).

[Aliaga 1999] D. Aliaga, J. Cohen, et al. "MMR: an interactive massive model rendering system using geometric and image-based acceleration". In *Proceedings of the 1999 symposium on Interactive 3D graphics*, Atlanta, Georgia, United States, ACM Press, (1999)

[Allcock 2001] B. Allcock, J. Bester, et al. "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing". In *MSS '01: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, Washington, DC, USA, IEEE Computer Society, (2001)

[Allcock 2002] B. Allcock, J. Bester, et al. "Data management and transfer in high-performance computational grid environments." *Parallel Comput.* **28**(5): 749-771, (2002).

[Allcock 2003] W. Allcock, J. Bester, et al. "GridFTP: Protocol Extensions to FTP for the Grid." *Global Grid Forum GFD-RP* **20**, (2003).

[Andrade 2007] H. Andrade, T. Kurc, et al. "Active semantic caching to optimize multidimensional data analysis in parallel and distributed environments." *Parallel Computing* **33**(7-8): 497-520, (2007).

[Asirvatham 2005] A. Asirvatham and H. Hoppe. "Terrain rendering using GPU-based geometry clipmaps". GPU Gems 2. M. Pharr and R. Fernando, Addison-Wesley, (2005).

[Assarsson 2000] U. Assarsson and T. Möller. "Optimized view frustum culling algorithms for bounding boxes." *J. Graph. Tools* **5**(1): 9-22, (2000).

[Baer 2004] T. Baer and P. Wyckoff. "A parallel I/O mechanism for distributed systems". In *IEEE International Conference on Cluster Computing 2004*, (2004)

[Bailey 2000] S. Bailey, E. Creel, et al. "A High Performance Implementation of the Data Space Transfer Protocol (DSTP)". In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, London, UK, Springer-Verlag, (2000)

[Barclay 1998] T. Barclay, R. Eberl, et al."Microsoft TerraServer." *Microsoft MSR-TR-98-17*, (1998)

[Baru 1998] C. Baru, R. Moore, et al. "The SDSC storage resource broker". In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada, IBM Press, (1998)

[Behnke 2005] J. Behnke, T. H. Watts, et al. "EOSDIS Petabyte Archives: Tenth Anniversary". In *Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05) - Volume 00*, IEEE Computer Society, (2005)

[Berkner 2002] K. Berkner and E. L. Schwartz. "Removal of tile artifacts using projection onto scaling functions for JPEG2000". In *Proceedings of 2002 International Conference on Image Processing*, (2002)

[Bethel 2003] E. W. Bethel, G. Humphreys, et al. "Sort-First, Distributed Memory Parallel Visualization and Rendering". In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, Washington, DC, USA, IEEE Computer Society, (2003)

[Bevirt 2004] B. Bevirt. "NCAR's MSS surpasses 2 petabytes of data stored." from http://www.cisl.ucar.edu/news/04/fotoweek/0729.mss2pb.html, (July 29, 2004).

[Beynon 2000] M. Beynon, R. A. Ferreira, et al. "DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems". In *Eighth NASA Goddard Conference on Mass Storage Systems and Technologies/Seventeenth IEEE Symposium on Mass Storage Systems*, (2000)

[Beynon 2002] M. Beynon, C. Chang, et al. "Processing large-scale multi-dimensional data in parallel and distributed environments." *Parallel Computing* **28**(5): 827-859, (2002).

[Bilgin 2000] A. Bilgin, G. Zweig, et al. "Three-dimensional image compression with integer wavelet transforms." *Applied Optics* **39**(11): 1799-1814, (2000).

[Bittner 1998] J. Bittner, V. Havran, et al. "Hierarchical Visibility Culling with Occlusion Trees". In *Proceedings of the Computer Graphics International 1998*, IEEE Computer Society, (1998)

[Bolch 2006] G. Bolch, S. Greiner, et al. "Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications", Wiley-Interscience, (2006).

[Boost] Boost. "http://www.boost.org."

[Brière 1996] N. Brière and P. Poulin. "Hierarchical view-dependent structures for interactive scene manipulation". In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, (1996)

[Brin 1998] S. Brin and L. Page. "The anatomy of a large-scale hypertextual Web search engine". In *Proceedings of the seventh international conference on World Wide Web 7*, Brisbane, Australia, Elsevier Science Publishers B. V., (1998)

[Bryson 1996] S. Bryson and S. Johan. "Time management, simultaneity and time-critical computation in interactive unsteady visualization environments". In *Proceedings of the 7th conference on Visualization '96*, San Francisco, California, United States, IEEE Computer Society Press, (1996)

[Callahan 2005] S. P. Callahan, J. L. D. Comba, et al. "Interactive rendering of large unstructured grids using dynamic level-of-detail". In *Visualization, 2005. VIS 05. IEEE*, (2005)

[Campbell 2003] P. M. Campbell, K. D. Devine, et al."Dynamic octree load balancing using space-filling curves." *Technical Report CS-03-01*, Williams College Department of Computer Science, (2003)

[Chang 2002] C. Chang, Z. Li, et al. "Hierarchical Image-based and Polygon-based Rendering for Large-Scale Visualizations". In *Scientific Visualization '02*, Springer-Verlag, (2002)

[Chen 2005] C. Chen. "Top 10 Unsolved Information Visualization Problems." *IEEE Computer Graphics and Applications* **25**(4): 12-16, (2005).

[Christopoulos 2000] C. Christopoulos, A. Skodras, et al. "The JPEG2000 still image coding system: an overview." *IEEE Transactions on Consumer Electronics* **46**(4): 1103-1127, (2000).

[Cohen-Or 2003] D. Cohen-Or, Y. L. Chrysanthou, et al. "A survey of visibility for walkthrough applications." *Visualization and Computer Graphics, IEEE Transactions on* **9**(3): 412-431, (2003).

[Corrêa 2002] W. T. Corrêa, J. T. Klosowski, et al. "Out-of-core sort-first parallel rendering for cluster-based tiled displays". In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, Blaubeuren, Germany, Eurographics Association, (2002)

[Corrêa 2003] W. T. Corrêa, J. T. Klosowski, et al. "Visibility-Based Prefetching for Interactive Out-Of-Core Rendering". In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, IEEE Computer Society, (2003)

[Daubechies 1992] I. Daubechies. "Ten lectures on wavelets", Society for Industrial and Applied Mathematics, (1992).

[Daubechies 1998] I. Daubechies and W. Sweldens. "Factoring Wavelet Transforms into Lifting Steps." *J. Fourier Anal. Appl.* **4**(3): 245-267, (1998).

[De Floriani 2000] L. De Floriani, P. Magillo, et al. "Dynamic view-dependent multiresolution on a client-server architecture." *Computer-Aided Design* **32**(13): 805-823, (2000).

[DeFanti 2002] T. DeFanti, J. Leigh, et al. "Teleimmersion and Visualization with the OptIPuter". In *Proceedings of the 12th International Conference on Artificial Reality and Telexistence (ICAT 2002)*, The University of Tokyo, Japan, (2002)

[DeFanti 2003] T. DeFanti, C. d. Laat, et al. "TransLight: a global-scale LambdaGrid for e-science." *Commun. ACM* **46**(11): 34-41, (2003).

[DeWitt 1994] D. J. DeWitt, N. Kabra, et al. "Client-Server Paradise". In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., (1994)

[Doshi 2003] P. R. Doshi, E. A. Rundensteiner, et al. "Prefetching for Visual Data Exploration". In *DASFAA '03: Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, Washington, DC, USA, IEEE Computer Society, (2003)

[Durand 1999] F. Durand "3D Visibility: analytical study and applications". University Joseph Fourier, Grenoble, France, **Ph. D. Thesis**, July

[El-Sana 1999] J. El-Sana and A. Varshney. "Generalized View-Dependent Simplification." *Computer Graphics Forum* **18, No. 3**: 83 - 94, (1999).

[Ellis 2004] L. Ellis."CDF and Granite: A Comparison." Department of Computer Science, University of New Hampshere, (2004)

[Folk 1999] M. Folk, R. E. McGrath, et al. "HDF: an update and future directions". In *IEEE 1999 International Proceeding on Geoscience and Remote Sensing Symposium, IGARSS '99*, Hamburg, Germany, (1999)

[Foster 1998] I. Foster and C. Kesselman. "The Grid: Blueprint for a New Computing Infrastructure." *Morgan-Kaufmann: San Francisco*, (1998).

[Fournier 1988] A. Fournier and D. Fussell. "On the power of the frame buffer." *ACM Trans. Graph.* **7**(2): 103-128, (1988).

[Franke 1999] E. Franke and M. Magee. "Reducing Data Distribution Bottlenecks by Employing Data Visualization Filters". In *Proceedings of The Eighth IEEE International Symposium on High Performance Distributed Computing*, IEEE Computer Society, (1999)

[Freitag 2001] L. A. Freitag and R. M. Loy. "Comparison of remote visualization strategies for interactive exploration of large data sets". In *15th International Parallel and Distributed Processing Symposium (IPDPS'01) Workshops*, (2001)

[Fuchs 1980] H. Fuchs, Z. M. Kedem, et al. "On visible surface generation by a priori tree structures." *SIGGRAPH Comput. Graph.* **14**(3): 124-133, (1980).

[Funkhouser 1992] T. A. Funkhouser, C. H. Séquin, et al. "Management of large amounts of data in interactive building walkthroughs". In *Proceedings of the 1992 symposium on Interactive 3D graphics*, Cambridge, Massachusetts, United States, ACM Press, (1992)

[Gaede 1998] V. Gaede and O. Günther. "Multidimensional access methods." *ACM Computing Surveys (CSUR)* **30**(2): 170-231, (1998).

[Gao 2004a] J. Gao "Visibility Acceleration for Large-scale Volume Visualization". Ohio State University, Department of Computer and Information Science, **Doctor of Philosophy Thesis**

[Gao 2004b] J. Gao, C. Wang, et al. "A Parallel Multiresolution Volume Rendering Algorithm for Large Data Visualization." *Parallel Computing* **31**(2): 185-204, (2004b).

[Gao 2005] J. Gao, J. Huang, et al. "Distributed data management for large volume visualization." *Visualization, 2005. VIS 05. IEEE*: 183-189, (2005).

[Garland 1999] M. Garland. "Multiresolution modeling: Survey & future opportunities". In *Eurographics 1999 -- State of the Art Reports*, (1999)

[GDAL] GDAL. "http://www.gdal.org."

[Giess 1998] C. Giess, A. Mayer, et al. "Medical Image Processing and Visualization on Heterogeneous Clusters of Symmetric Multiprocessors using MPI and POSIX Threads". In *12th. International Parallel Processing Symposium* (1998)

[Gobbetti 2005] E. Gobbetti and F. Marton. "Far voxels: a multiresolution framework for interactive rendering of huge complex 3D models on commodity graphics platforms." *ACM Trans. Graph.* **24**(3): 878-885, (2005).

[Gonzalez 2007] R. C. Gonzalez and R. E. Woods. "Digital Image Processing", Prentice Hall, (2007).

[Grama 2003] A. Grama, A. Gupta, et al. "An Introduction to Parallel Computing: Design and Analysis of Algorithms", Addison Wesley, (2003).

[Gray 2003] J. Gray. "What next?: A dozen information-technology research goals." *J. ACM* **50**(1): 41-57, (2003).

[Greene 1993] N. Greene, M. Kass, et al. "Hierarchical Z-buffer visibility". In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM Press, (1993)

[Gropp 1998] W. Gropp, S. Huss-Lederman, et al. "MPI: The Complete Reference - 2nd Edition: Volume 2 - The MPI-2 Extensions", The MIT Press, (1998).

[Gropp 2002] W. Gropp. "MPICH2: A New Start for MPI Implementations". In *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer-Verlag, (2002)

[Grossman 2003] R. L. Grossman, Y. Gu, et al. "Experimental studies using photonic data services at IGrid 2002." *Future Gener. Comput. Syst.* **19**(6): 945-955, (2003).

[Guthe 2002] S. Guthe, M. Wand, et al. "Interactive rendering of large volume data sets". In *Proceedings of the conference on Visualization '02*, Boston, Massachusetts, IEEE Computer Society, (2002)

[Hadjieleftheriou 2005] M. Hadjieleftheriou, E. Hoel, et al. "SaIL: A Spatial Index Library for Efficient Application Integration." *Geoinformatica* **9**(4): 367-389, (2005).

[Haimes 1994] R. Haimes. "pV3: A distributed system for large-scale unsteady CFD visualization". In *AIAA paper*, (1994)

[Hamdi 1997] M. Hamdi and C. K. Lee. "Dynamic load-balancing of image processing applications on clusters of workstations." *Parallel Computing* **22**(11): 1477-1492, (1997).

[He 2003] E. He, J. Alimohideen, et al. "Quanta: a toolkit for high performance data delivery over photonic networks." *Future Gener. Comput. Syst.* **19**(6): 919-933, (2003).

[Heckbert 1994] P. S. Heckbert and M. Garland. "Multiresolution Modeling for Fast Rendering". In *Proc. Graphics Interface '94*, Banff, Canada, Canadian Inf. Proc. Soc., (1994)

[Hennessy 2006] J. L. Hennessy and D. A. Patterson. "Computer Architecture: A Quantitative Approach ", Morgan Kaufmann, (2006).

[Herzen 1987] B. V. Herzen and A. H. Barr. "Accurate triangulations of deformed, intersecting surfaces". In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, (1987)

[Hey 2003] T. Hey and A. Trefethen. "The Data Deluge: An e-Science Perspective". Grid Computing. F. Berman, G. Fox and T. Hey**:** 809-824, (2003).

[Hoppe 1996] H. Hoppe. "Progressive meshes". In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, (1996)

[Hoppe 1997] H. Hoppe. "View-dependent refinement of progressive meshes". In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., (1997)

[Hua 2003] L. Hua "Vector Wavelet Transforms For The Coding Of Static And Time-varying Vector Fields". Mississippi State University, Department of Electrical and Computer Engineering**Thesis**

[Hudson 1997] T. Hudson, D. Manocha, et al. "Accelerated occlusion culling using shadow frusta". In *Proceedings of the thirteenth annual symposium on Computational geometry*, Nice, France, ACM Press, (1997)

[Jobard 2001] B. Jobard and W. Lefer. "Multiresolution Flow Visualization". In *WSCG'01*, (2001)

[Johnson 2004] C. Johnson. "Top Scientific Visualization Research Problems." *IEEE Comput. Graph. Appl.* **24**(4): 13-17, (2004).

[Johnston 1997] W. Johnston, G. Jin, et al. "Real-Time Digital Libraries based on Widely Distributed, High Performance Management of Large-Data-Objects." *International Journal of Digital Libraries - Special Issue on Digital Libraries in Medicine* **1**(3): 241-256, (1997).

[Jones 2006] B. Jones, D. Kranzlmüller, et al. "Lessons from Europe's International Grid Initiatives: Grid Technology in Africa". In *IST-Africa 2006 conference*, Pretoria, South Africa, (2006)

[Joseph 1999] D. Joseph and D. Grunwald. "Prefetching using Markov predictors." *IEEE Transactions on Computers* **48**(2): 121-133, (1999).

[Khodakovsky 2000] A. Khodakovsky, P. Schröder, et al. "Progressive Geometry Compression". In *SIGGRAPH2000*, (2000)

[Kim 2004] K. H. Kim. "Wide-area real-time distributed computing in a tightly managed optical grid-an OptIPuter vision." *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on* **1**, (2004).

[Kirkpatrick 2006] D. Kirkpatrick. "Microsoft's cash versus Google". *Fortume*. **153**, (2006).

[Klosowski 2000] J. T. Klosowski and C. T. Silva. "The Prioritized-Layered Projection Algorithm for Visible Set Estimation." *IEEE Transactions on Visualization and Computer Graphics* **6**(2): 108-123, (2000).

[Kniss 2001] J. Kniss, P. McCormick, et al. "Interactive Texture-Based Volume Rendering for Large Data Sets." *IEEE Comput. Graph. Appl.* **21**(4): 52-61, (2001).

[Krishnaprasad 2004] N. Krishnaprasad, V. Vishwanath, et al. "JuxtaView – a Tool for Interactive Visualization of Large Imagery on Scalable Tiled Displays". In *Proceedings of IEEE Cluster 2004*, San Diego, CA, (2004)

[Kurc 2000] T. M. Kurc, M. Beynon, et al."DataCutter and A Client Interface for the Storage Resource Broker with DataCutter Services." (2000)

[Lai 2004] K. Y. Lai, Z. Tari, et al. "Location-aware cache replacement for mobile environments". In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, (2004)

[Lalonde 1999] P. Lalonde and A. Fournier. "Interactive rendering of wavelet projected light fields". In *Proceedings of the 1999 conference on Graphics interface '99*, Kingston, Ontario, Canada, Morgan Kaufmann Publishers Inc., (1999)

[LaMar 2003] E. LaMar and V. Pascucci. "A Multi-Layered Image Cache for Scientific Visualization". In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, IEEE Computer Society, (2003)

[Lawder 2000] J. K. Lawder and P. J. H. King. "Using Space-Filling Curves for Multi-dimensional Indexing". In *Proceedings of the 17th British National Conferenc on Databases: Advances in Databases*, Springer-Verlag, (2000)

[Lee 1995] C.-k. Lee and M. Hamdi. "Parallel image processing applications on a network of workstations." *Parallel Computing* **21**(1): 137-160, (1995).

[Leigh 2003] J. Leigh, L. Renambot, et al. "An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization". In *3rd Workshop on Advanced Collaborative Environments (in conjunction with the High Performance Distributed Computing Conference)*, Seattle, WA, (2003)

[Leigh 2007] J. Leigh, A. Johnson, et al. "Emerging from the CAVE: Collaboration in Ultra High Resolution Environments". In *Proceedings of the First International Symposium on Universal Communication*, Kyoto, Japan, (2007)

[Lester 2003] L. Lester. "NCAR's MSS exceeds 1 petabyte." from http://www.cisl.ucar.edu/news/03/features/0227.petabyte.html, (February 27, 2003).

[Leven 2002] J. Leven, J. Corso, et al. "Interactive visualization of unstructured grids using hierarchical 3D textures". In *VVS '02: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, Boston, Massachusetts, IEEE Press, (2002)

[Levenberg 2002] J. Levenberg. "Fast View-Dependent Level-of-Detail Rendering Using Cached Geometry". In *13th IEEE Visualization 2002*, Los Alamitos, CA, USA, IEEE Computer Society, (2002)

[Li 2002a] P. P. Li. "Supercomputing Visualization for Earth Science Datasets". In *Proceedings of 2002 NASA Earth Science Technology Conference*, (2002a)

[Li 2002b] X. Li and H.-W. Shen. "Time-critical multiresolution volume rendering using 3D texture mapping hardware". In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, Boston, Massachusetts, IEEE Press, (2002b)

[Liang 2005] K. Liang, P. Monger, et al. "Interactive parallel visualization of large particle datasets." *Parallel Computing* **31**(2): 243-260, (2005).

[Lindstrom 1996] P. Lindstrom, D. Koller, et al. "Real-time, continuous level of detail rendering of height fields". In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, (1996)

[Liu 2007] Y. Liu and W. A. Pearlman. "Four-Dimensional Wavelet Compression of 4-D Medical Images Using Scalable 4-D SBHP". In *Data Compression Conference, 2007. DCC '07*, (2007)

[Livny 2007] Y. Livny, Z. Kogan, et al. "Seamless Patches for GPU-Based Terrain Rendering". In *WSCG*, (2007)

[Luebke 2002] D. Luebke, M. Reddy, et al. "Level of Detail for 3D Graphics", Morgan Kaufmann, (2002).

[Luebke 2001] D. P. Luebke and B. Hallen. "Perceptually-Driven Simplification for Interactive Rendering". In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, Springer-Verlag, (2001)

[Ma 1998] K.-L. Ma, D. Smith, et al."Efficient Encoding and Rendering of Time-Varying Volume Data." *ICASE Report No. 98-22*, NASA/CR-1998-208424, (1998)

[Ma 2003] K.-L. Ma. "Visualizing Time-Varying Volume Data." *Computing in Science and Engineering* **05**(2): 34-42, (2003).

[Malvar 1999] H. S. Malvar. "Fast progressive wavelet coding". In *IEEE Data Compression Conference*, Snowbird, UT, (1999)

[McCormick 1987] B. H. McCormick, T. A. DeFanti, et al. "Visualization in Scientific Computing." *Computer Graphics (Proc. Siggraph 1987)* **21**(6), (1987).

[McCormick 2005] P. McCormick and J. Ahrens. "Large-Scale Data Visualization and Rendering: A Problem-Driven Approach". The Visualization Handbook. C. D. Hansen and C. R. Johnson. Boston, Elsevier-Butterworth Heinemann**:** 533-550, (2005).

[McMillan 1995] L. McMillan and G. Bishop. "Plenoptic modeling: an image-based rendering system". In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, (1995)

[Meerwald 2002] P. Meerwald, R. Norcen, et al. "Parallel JPEG2000 Image Coding on Multiprocessors". In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Washington, DC, USA, IEEE Computer Society, (2002)

[Moreland 2001] K. Moreland, B. Wylie, et al. "Sort-last parallel rendering for viewing extremely large data sets on tile displays". In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, San Diego, California, IEEE Press, (2001)

[MPICH2] MPICH2. "http://www-unix.mcs.anl.gov/mpi/mpich2/."

[Munzner 2000] T. Munzner "Interactive Visualization of Large Graphs and Networks". Stanford University, Computer Science Department, **Ph. D. Thesis Thesis**

[Nesbit 2005] K. J. Nesbit and J. E. Smith. "Data Cache Prefetching Using a Global History Buffer." *IEEE Micro* **25**(1): 90-97, (2005).

[Ni 2006] T. Ni, G. S. Schmidt, et al. "A Survey of Large High-Resolution Display Technologies, Techniques, and Applications." *Virtual Reality Conference, 2006*: 223-236, (2006).

[Nicolescu 2002] C. Nicolescu and P. Jonker. "A data and task parallel image processing environment." *Parallel Computing* **28**(7-8): 945-965, (2002).

[Oldfield 2002] R. Oldfield and D. Kotz. "Armada: a parallel I/O framework for computational grids." *Future Gener. Comput. Syst.* **18**(4): 501-523, (2002).

[Paterson 1990] M. S. Paterson and F. F. Yao. "Efficient binary space partitions for hidden-surface removal and solid modeling." *Discrete Comput. Geom.* **5**(5): 485-503, (1990).

[Pavlakos 2005] C. Pavlakos and P. D. Heermann. "Issues and Architectures in Large-Scale Data Visualization". The Visualization Handbook. C. D. Hansen and C. R. Johnson. Boston, Elsevier-Butterworth Heinemann**:** 551-567, (2005).

[Pons 2003] A. P. Pons. "Web-application centric object prefetching." *J. Syst. Softw.* **67**(3): 193-200, (2003).

[Prohaska 2004] S. Prohaska, A. Hutanu, et al. "Interactive Exploration of Large Remote Micro-CT Scans". In *VIS '04: Proceedings of the conference on Visualization '04*, Washington, DC, USA, IEEE Computer Society, (2004)

[Ren 2000] Q. Ren and M. H. Dunham. "Using semantic caching to manage location dependent data in mobile computing". In *Proceedings of the 6th annual international conference on Mobile computing and networking*, Boston, Massachusetts, United States, ACM Press, (2000)

[Renambot 2004] L. Renambot, A. Rao, et al. "SAGE: the Scalable Adaptive Graphics Environment". In *Proceedings of WACE 2004*, (2004)

[Rew 1990] R. Rew and G. Davis. "Data Management: NetCDF: an Interface for Scientific Data Access." *IEEE Comput. Graph. Appl.* **10**(4): 76-82, (1990).

[Rhodes 2002] P. J. Rhodes, R. D. Bergeron, et al. "A Data Model for Distributed Multiresolution Multisource Scientific Data". Hierarchical and Geometrical Methods in Scientific Visualization. G. Farin, H. Hagen and B. Hamann. Germany, Springer-Verlag, Heidelberg, (2002).

[Rhodes 2003] P. J. Rhodes, R. D. Bergeron, et al. "A data model for adaptive multi-resolution scientific data." In *Data Visualization: The State of the Art*, (2003)

[Rhodes 2005] P. J. Rhodes, X. Tang, et al. "Iteration Aware Prefetching for Large Multidimensional Scientific Datasets". In *Statistical and Scientific Database Management (SSDBM) 2005*, Santa Barbara, California, (2005)

[Sagan 1994] H. Sagan. "Space-filling curves". New York, Springer-Verlag, (1994).

[Saltz 2003] J. H. Saltz. "Capstone: Middleware Infrastructure for Large-Scale Data Management". In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, IEEE Computer Society, (2003)

[Schneider 2006] J. Schneider and R. Westermann. "GPU-Friendly High-Quality Terrain Rendering." *Journal of WSCG* **14**(1-3): 49-56, (2006).

[Schwarz 2004a] N. Schwarz, P. v. Keken, et al. "Distributed volume rendering of global models of seismic wave propagation." *AGU Fall Meeting Abstracts*: A702+, (2004a).

[Schwarz 2004b] N. Schwarz, S. Venkataraman, et al. "Vol-a-Tile - a Tool for Interactive Exploration of Large Volumetric Data on Scalable Tiled Displays". In *Proceedings of IEEE Visualization 2004*, Austin, Texas, (2004b)

[Seinstra 2002] F. J. Seinstra, D. Koelma, et al. "A software architecture for user transparent parallel image processing." *Parallel Computing* **28**(7-8): 967-993, (2002).

[Serot 2002] J. Serot and D. Ginhac. "Skeletons for parallel image processing: an overview of the SKIPPER project." *Parallel Computing* **28**(12): 1685-1708, (2002).

[Shen 1999] H.-W. Shen, L.-J. Chiang, et al. "A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree". In *Proceedings of the conference on Visualization '99: celebrating ten years*, San Francisco, California, United States, IEEE Computer Society Press, (1999)

[Shneiderman 1996] B. Shneiderman. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations". In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, IEEE Computer Society, (1996)

[Singh 2006] R. Singh, N. Schwarz, et al. "Real-time multi-scale brain data acquisition, assembly, and analysis using an end-to-end OptIPuter." *Future Gener. Comput. Syst.* **22**(8): 1032-1039, (2006).

[Sisneros 2007] R. Sisneros, C. Jones, et al. "A Multi-Level Cache Model for Run-Time Optimization of Remote Visualization." *Transactions on Visualization and Computer Graphics* **13**(5): 991-1003, (2007).

[Smarr 2003] L. L. Smarr, A. A. Chien, et al. "The OptIPuter." *Communication of the ACM* **46**(11): 58-67, (2003).

[Stockli 2005] R. Stockli, E. Vermote, et al. "The Blue Marble Next Generation-A true color earth dataset including seasonal dynamics from MODIS."

[Sweldens 1995] W. Sweldens. "The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions". In *Wavelet Applications in Signal and Image Processing III*, Proc. SPIE 2569, (1995)

[Uytterhoven 1999] G. Uytterhoven "Wavelets: software and applications". Katholieke Universiteit Leuven, Department of Computer Science, **Ph.D. Thesis**, April 1999

[Wang 2005] C. Wang, J. Gao, et al. "A Multiresolution Volume Rendering Framework for Large-Scale Time-Varying Data Visualization". In *International Workshop on Volume Graphics*, Stony Brook, NY, (2005)

[Wang 2006] X. Wang, V. Vishwanath, et al. "LambdaBridge: A Scalable Architecture for Future Generation Terabit Applications". In *Broadnets 2006 - Third International Conference on Broadband Communications, Networks, and Systems*, San Jose, CA, (2006)

[Woo 1999] M. Woo, J. Neider, et al. "OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2", Addison-Wesley Longman Publishing Co., Inc., (1999).

[Xiong 2005] C. Xiong, J. Leigh, et al. "LambdaStream – a Data Transport Protocol for Streaming Network-intensive Applications over Photonic Networks". In *Proceedings of The Third International Workshop on Protocols for Fast Long-Distance Networks*, Lyon, France, (2005)

[Yang 2005] M. Yang, R. E. McGrath, et al. "HDF5 - A High Performance data Format for Earth Science". In *21st International Conference on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology*, (2005)

[Yu 1997] J.-B. Yu and D. J. DeWitt. "Query Pre-Execution and Batching in Paradise: A Two-Pronged Approach to the Efficient Processing of Queries on Tape-Resident Raster Images". In *SSDBM '97: Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, Washington, DC, USA, IEEE Computer Society, (1997)

[Zhang 2003] C. Zhang, J. Leigh, et al. "TeraScope: distributed visual data mining of terascale data sets over photonic networks." *Future Gener. Comput. Syst.* **19**(6): 935-943, (2003).

[Zhang 1997] H. Zhang, D. Manocha, et al. "Visibility culling using hierarchical occlusion maps". In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., (1997)

VITA

NANE Chong Zhang

EDUCATION

2000 - 2007 Ph.D., Computer Science, University of Illinois at Chicago, Chicago, Illinois

1997 - 2000 M.S., Electrical Engineering, Wuhan University, Wuhan, China

1993 - 1997 M.S., Electrical Engineering, Wuhan University, Wuhan, China

HONORS

Summa Cum Laude Graduate Student

Outstanding Master Thesis

PUBLICATIONS

1. L. Renambot, Rao, A., Singh, R., Jeong, B., Krishnaprasad, Naveen, Vishwanath, V., Chandrasekhar, V., Schwarz,N., Spale, A., **Chong Zhang**, Goldman, G., Leigh, J., Johnson, A., SAGE: the Scalable Adaptive Graphics Environment, Proceedings of WACE 2004 09/23/2004 - 09/24/2004

2. J. Leigh, Renambot, L., DeFanti, T., Brown, M., He, E, Krishnaprasad, N., Meerasa, J., Nayak, A., Park, K., Singh, R., Venkataraman, S., **Chong Zhang**, Livingston, D., McLaughlin, M., An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization, 3rd Workshop on Advanced Collaborative Environments (in conjunction with the High Performance Distributed Computing Conference), Seattle, WA 06/22/2003 - 06/22/2003

3. **Chong Zhang**, Jason Leigh , Thomas A. DeFanti , Marco Mazzucco , Robert Grossman, TeraScope: distributed visual data mining of terascale data sets over photonic networks, Future Generation Computer Systems, v.19 n.6, p.935-943, August 2003

4. Thomas A. DeFanti, Jason Leigh, Maxine D. Brown, Daniel J. Sandin, Oliver Yu, **Chong Zhang**, Rajvikram Singh, Eric He, J. Alimohideen, N. Krishnaprasad, Robert Grossman, Marco Mazzucco, Larry Smarr, Mark Ellisman, Phil Papadopoulos, Andrew Chien, John Orcutt, Teleimmersion and Visualization with the OptIPuter, Proceedings of the 12th International Conference on Artificial Reality and Telexistence (ICAT 2002), Ohmsha/IOS Press.