Visualization and Interaction Metaphors for Large High-Resolution Displays

Vom Fachbereich Informatik der Technischen Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation
von

Dipl.-Inf. Sebastian Thelen

Datum der wissenschaftlichen Aussprache: 9. Februar 2011

Dekan:	Prof. Dr. Arnd Poetzsch-Heffter
Vorsitzender des Prüfungsausschusses:	Prof. Dr. Peter Liggesmeyer
1. Berichterstatter:	Prof. Dr. Achim Ebert
2. Berichterstatter:	Prof. Dr. Jörg Meyer
3. Berichterstatter:	Prof. Dr. Hans Hagen

© Copyright 2010
by Sebastian Thelen

Acknowledgements

This thesis would not have been possible without the help of many people who supported me during my doctoral studies. First of all, I would like to thank my advisors Achim Ebert and Jörg Meyer. Achim and Jörg gave me the opportunity to freely conduct my research and work on the projects I was most interested in. In particular, I would like to thank Jörg for making my visit to the University of California, Irvine an unforgettable time in both scientific and non-scientific matters. I am especially thankful to Prof. Hans Hagen, who gave me the opportunity to conduct my research as a member of the International Research Training Group "Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling, and Engineering" (IRTG 1131), University of Kaiserslautern, Germany. More than anything else before in my still short academic life, I enjoyed the opportunity to work in an interdisciplinary and international environment.

Furthermore, I would like to thank all my friends and colleagues at the University of Kaiserslautern and the University of California, Irvine. My thanks go to Peter-Scott Olech for the excellent collaboration which resulted in major parts of this thesis. I would also like to thank Aaron Knoll for our common work on direct volume rendering optimizations. Without him, major parts of this thesis would not exist. Further thanks go to Jan Malburg for his help on the Tiled++ prototype and the AnyScreen framework. My thanks go to members of the California Institute for Telecommunication and Information Technology (Calit2); in particular, I would like to thank Stephen Jenks, Sung-Jin Kim, Falko Küster, and Kai-Uwe Dörr for allowing me to use parts of their HIPerWall software. I want to thank my office mate Li-Chang Cheng for the fruitful discussions and nice lunch breaks. Furthermore, I would like to thank Ariane Middel for proofreading my papers and for providing me with the Maricopa County census data set. Special thanks go to Inga Scheler and Mady Gruys for helping me manage all the confusing and sometimes annoying organizational and administrative things that need to be done.

Finally, and most importantly, I would like to thank my family for their confidence and constant encouragement during the long years of my education. I deeply appreciate my mother's unwavering support which allowed me to undertake my studies in the very first place. I thank my father for having an open ear and for encouraging me to carry on when times were hard. My thanks go to my grandfather for keeping me with both feet on the ground. I want to thank Annette for her love, confidence and telling me to follow my heart. She reminded me that there is a life outside academia and helped me to not turn into a complete nerd.

Abstract

Due to remarkable technological advances in the last three decades the capacity of computer systems has improved tremendously. Considering Moore's law, the number of transistors on integrated circuits has doubled approximately every two years and the trend is continuing. Likewise, developments in storage density, network bandwidth, and compute capacity show similar patterns. As a consequence, the amount of data that can be processed by today's systems has increased by orders of magnitude. At the same time, however, the resolution of screens has hardly increased by a factor of ten. Thus, there is a gap between the amount of data that can be processed and the amount of data that can be visualized. Large high-resolution displays offer a way to deal with this gap and provide a significantly increased screen area by combining the images of multiple smaller display devices. The main objective of this dissertation is the development of new visualization and interaction techniques for large high-resolution displays.

In the first part of this thesis a hybrid display setup combining flat panel display and computer projector technology is presented to overcome the issue of discontinuous images in multi-monitor systems. Discontinuities are primarily caused by monitor frames and are the strongest argument against using highly parallel multi-monitor systems. The approach eliminates these discontinuities and is thoroughly evaluated in an extensive user study. Furthermore, a software framework is introduced that enables developers to implement distributed applications for arbitrary display configurations. Since large displays are usually driven by a cluster of render nodes, the framework provides an abstraction layer to free developers from the complexity of the underlying system when implementing applications.

The second part of the manuscript deals with interaction concepts for large high-resolution displays. Due to their size many traditional interaction devices and metaphors whose origins are in single display environments do not scale to the size of large displays or to the size of user groups working with them. A new interaction approach is introduced which is based on a combination of two-dimensional visual tags and camera-enabled smart phones. The approach supports collaboration of

multiple user groups in parallel and is implemented in two example scenarios on a fifty tile display cluster.

The third part of this dissertation deals with direct volume rendering on large high-resolution displays and introduces a technique to visualize giga-scale data sets whose sizes exceed the available, combined resources of all render nodes. The proposed out-of-core technique makes efficient use of the resources available on modern graphics cards and generates renderings of volumetric data sets larger than the texture buffer size of a single graphics card at significantly higher levels of detail than on a single desktop display. Furthermore, the thesis investigates how hierarchies of bounding volumes can be exploited as acceleration structures for direct volume rendering to achieve high frame rates during visualization.

Zusammenfassung

Aufgrund technologischer Fortschritte hat sich die Leistungsfähigkeit von Computersystemen in den letzten Jahrzehnten beachtlich gesteigert. Zieht man das Gesetz von Moore zu Rate, so hat sich die Anzahl der Transistoren in integrierten Schaltungen etwa alle zwei Jahre verdoppelt, wobei der Trend sich fortzusetzen scheint. Ähnliche Tendenzen lassen sich bei der Entwicklung der Speicherdichte, der Netzwerkbandbreite und der Verarbeitungskapazität ausmachen. Als Konsequenz ist die Menge der verarbeitbaren Daten um viele Größenordnungen gestiegen. Gleichzeitig allerdings hat sich das Auflösungsvermögen von Bildschirmen nur etwa verzehnfacht. Folglich klafft eine Lücke zwischen der Menge der Daten, die verarbeitet werden können und der Menge der Daten, die auf einem Bildschirm darstellbar sind. Große hochauflösende Displays bieten eine Lösung, indem sie die Bilder mehrerer kleinere Displays zu einem Gesamtbild vereinen. Der Schwerpunkt dieser Dissertation liegt auf der Entwicklung neuer Visualisierungs- und Interaktionstechniken für große hochauflösende Displays.

Im ersten Teil der Arbeit wird ein hybrider Flachbildschirm-/Projektor-Ansatz vorgestellt, um das Problem von Bilddiskontinuitäten in Multimonitorsystemen zu lösen. Bilddiskontinuitäten werden primär durch Monitorrahmen hervorgerufen und sind ein Hauptargument gegen den Einsatz hoch paralleler Multimonitorsysteme. Der vorgestellte Ansatz beseitigt die Diskontinuitäten und wird in einer ausführlichen Benutzerstudie evaluiert. Des Weiteren wird ein Software-Framework vorgestellt, welches Entwicklern die Implementierung verteilter Anwendungen für beliebige Konfigurationen von Displays ermöglicht. Da große Displays für gewöhnlich mit Hilfe eines Clusters von Renderknoten betrieben werden, stellt das Framework dem Entwickler eine Abstraktionschicht zu Verfügung, um ihn von der zugrunde liegenden Systemkomplexität zu befreien.

Der zweite Teil des Manuskripts beschäftigt sich mit Interaktionskonzepten für große, hochauflösende Displays. Viele Interaktionsgeräte und Metaphern, deren Ursprünge im Bereich der Einzelbildschirmsysteme liegen, lassen sich nicht ohne Weiteres hochskalieren, wenn die Größe der hochauflösenden Displays oder die Anzahl der Nutzer, die mit ihnen arbeitet, wachsen. Es wird ein neuer Interaktionsansatz

präsentiert, der auf einer Kombination von zweidimensionalen Bildmarkierungen und sogenannten Smart Phones mit Kameraunterstützung aufbaut. Der Ansatz unterstützt die gleichzeitige Zusammenarbeit mehrere Benutzergruppen. Er wurde auf einem Displaycluster bestehend aus fünfzig Monitoren implementiert, und seine Anwendbarkeit wurde in zwei Beispielszenarien demonstriert.

Der dritte Teil der Dissertation beschäftigt sich mit der Volumenvisualisierung auf großen, hochauflösenden Displays und stellt eine neue Technik für die Darstellung gigabyte-großer Datensätze vor, deren Speicherbedarf die Ressourcen einzelner Renderknoten bei Weitem überschreitet. Der Ansatz schöpft die zur Verfügung stehenden Ressourcen moderner Grafikkarten vollständig aus und ist in der Lage Datensätze darzustellen, die aufgrund ihrer Größe nicht vollständig im Texturspeicher einer einzelnen Grafikkarte untergebracht werden können. Dabei überschreitet der erzielte Detailgrad auf der gesamten Displaywand den Detailgrad eines Einzelmonitorsystems um ein Vielfaches. Außerdem untersucht die Arbeit, inwiefern sich interaktive Bildwiederholraten bei der Volumenvisualisierung mit Hilfe von Datenstrukturen erzielen lassen, die auf Hierarchien von Begrenzungsvolumina basieren.

Contents

List of Figures **v**

List of Tables **x**

1 Introduction **1**

1.1 Motivation 1

1.2 Overview and Contribution 2

2 Setting up Large High-Resolution Displays **5**

2.1 State-of-the-Art and Related Work 6

2.1.1 Hardware Configurations 6

2.1.2 Example Configurations 8

2.1.3 Distributed Rendering Libraries 10

2.1.4 The Current Top 10 Research Challenges 11

2.2 AnyScreen - A Highly Scalable Distributed Rendering Framework for Arbitrary Display and Display-in-Display Configurations 14

2.2.1 Framework Architecture 15

2.2.2 Stereo Modes 20

2.2.3 Rendering Process 22

2.2.4 Examples 24

2.2.5 Discussion 25

2.3 Tiled++ - An Enhanced Tiled Hi-Res Display Wall 26

2.3.1 The Bezel Effect 27

2.3.2 The Tiled++ Method 30

2.3.3 System Calibration 31

2.3.4 Protoypes 37

2.3.5 Examples 37

2.3.6 Interacting with Tiled++ 39

2.3.7 Discussion of Results 41

2.3.8 Evaluation 42

2.3.9 Results 48

2.3.10 Discussion 52

2.4 Conclusions 52

3 Interaction with Large High-Resolution Displays 54

3.1 State-of-the-Art and Related Work 55

3.1.1 Human Performance 55

3.1.2 Interfaces and Interaction Techniques 58

3.1.3 Collaborative Environments 62

3.1.4 Mobile Interaction 64

3.2 Tag-based Interaction with Large High-Resolution Displays 65

3.2.1 Interaction using Visual Tags 65

3.2.2 High Capacity Color Barcodes 66

3.2.3 System Architecture 68

3.2.4 Contribution 69

3.2.5 Google Earth 70

3.2.6 A 3D Human Brain Atlas 74
3.2.7 Discussion 79
3.3 Conclusions 80

4 Volume Visualization on Large High-Resolution Displays 81
4.1 State-of-the-Art and Related Work 82
4.1.1 Volume Data 82
4.1.2 Physically Based Direct Volume Rendering 83
4.1.3 The Graphics Pipeline 85
4.1.4 Direct Volume Rendering Techniques 87
4.1.5 Acceleration Structures for Large-Scale Data Sets 90
4.1.6 Multi-resolution Techniques 92
4.1.7 Large Display Volume Data Systems 97
4.2 Out-of-Core Techniques for Large-Scale Volume Rendering on Distributed Display Clusters 98
4.2.1 Data Structures 99
4.2.2 System Details 101
4.2.3 System Design 102
4.2.4 Implementation 104
4.2.5 Results 107
4.2.6 Analysis 108
4.2.7 Discussion 112
4.3 Fast Direct Volume Rendering Techniques for Distributed Display Clusters 112
4.3.1 BVH-Based Direct Volume Rendering on the GPU 113
4.3.2 BHV-Based Direct Volume Rendering on a Hybrid CPU/GPU System 119

4.3.3 BVH-Based Direct Volume Rendering on the CPU 123
4.3.4 Results 127
4.3.5 Discussion 130
4.4 Conclusions 131

5 Conclusions and Future Work 133

5.1 Conclusions 133
5.2 Future Work 134

A Analysis of Variance (ANOVA) 136

References 138

Biographical Information 151

List of Publications 152

List of Figures

2.1 Possible large high-resolution display configurations: (a) monitor-based tiled display wall, (b) projector-based tiled display wall *Image courtesy of Majumder et al. [MS05]*, (c) Cave Automatic Virtual Environment (CAVE$^{\text{TM}}$). 7

2.2 Tiled wall installations: (a) 3×3 tiled display wall driven by 5 render nodes, (b) HIPerWall, a 200 megalpixel 10×5 display cluster driven by 25 render nodes. 9

2.3 Possible execution modes for distributed rendering libraries: (a) master-slave execution mode, (b) client-server execution mode. 12

2.4 Thread/process model of *AnyScreen* applications executed on a 3×3 tiled display. 15

2.5 The *AnyScreen* configuration tool can be used to generate and edit XML configuration files for arbitrary display configurations. 18

2.6 Logical render pass diagram. 23

2.7 Semantic problems: (a) original scenario, (b) discontinuities due to the offset strategy (image appears distorted/expanded), (c) overlays are hiding the wire joint (missing pixels), (d) missing information provided by Tiled++. 27

2.8 Semantic problems on HIPerWall: (a) original image (Los Angeles Railway System of 1906), (b) discontinuities due to the offset method (diagonal lines appear to be disconnected or incorrectly aligned), (c) overlay technique (pixels under bezels have been eliminated, leading to missing information), (d) missing information provided by Tiled++. 28

2.9 The Poggendorff illusion. Although both oblique lines are in fact collinear, they appear to be offset. 29

2.10 Low-resolution (about 10% of the original resolution) image information is projected only onto the bezel grid. The areas of the LCD panels are blacked out. (Sections of Los Angeles Railway Map.) . . . 31

2.11 Homography-based keystone compensation: (a) selecting corners defines correspondences between the undistorted image and points in projector screen space yielding H, (b) applying H distorts the original image I so that its projection $P(H)$ is free of keystone effects. . . 35

2.12 A CAD example: (a) strong discontinuity effects with the offset strategy disturb perception, (b) with overlay strategy, image information is hidden under the bezels, (c) Tiled++ provides missing information by bezel projection. 38

2.13 Tiled++ provides "higher order" image information, such as, in this case, a company logo. 39

2.14 A HIPerWall user enters the projection area and casts a shadow on the Tiled++ projection, because the projector is mounted too low. . . 40

2.15 Human zooming and the effects of physical movement on Tiled++: (a) context information is available by stepping back, (b) focus information is available by stepping forward, (c) shadow removal using multiple projectors. 41

2.16 The navigation and perception tasks of the evaluation: a) HotWire, b) Pong single-player mode: Squash, c) Poggendorff test, d) animation test. 45

2.17 Results of the user study: (a) perception ratings for Pong, (b) perception ratings for the video test, (c) perception ratings for the Poggendorff test, (c) efficiency ratings for the Poggendorff test. 50

3.1 Examples of interaction devices for large high-resolution displays: (a) 3Dconnexion SpacePilot^{TM} Pro [3dc] *Image courtesy of 3Dconnexion Inc., Tobias Keuthen*, (b) Animazoo Gypsy^{TM} motion capture suit [ani] *Image courtesy of Animazoo, Paul Collimore*, (c) 5DT DataGlove Ultra Wireless^{TM} [5DT], (d) Saitek Cyborg evo Force Feedback^{TM} joystick [sai] *Image courtesy of Saitek Inc.* 61

3.2 Collaborative working environment: (a) a tiled display holding multiple data sources, (b) interaction of multiple users. 63

3.3 High Capacity Color Barcode: triangles of four different colors encode 13 bytes of raw data including error correction in a 5 × 10 grid. . . . 67

3.4 Client-server architecture used for applications based on the HCCB approach. 68

3.5 Tag-based interaction with Google Earth on HIPerWall: (a) a group of users is working in front of the census data set. The picture shows the data set as well as the public screen and the "toggle menu" located on the right (three monitor columns are not captured in this image), (b) a description balloon pops up after scanning a placemark barcode and displays four out of sixty data values. 71

3.6 Presentation and interaction with data: (a) detailed census tract information shown on cell phone displays. The red marked publish button at the bottom of the site loads the content into a web browser on the public screen, (b) tags of the tract selection menu are used to toggle the visibility of different census tract groups based on their ID. 72

3.7 Tag-based interaction with the 3D atlas: (a) user in front of the display, (b) close-up: user prompted to enter data after scanning a tag, (c) input form: instructor can query answers directly by pressing the answer button. 76

4.1 The programmable GPU graphics pipeline. 85

4.2 Sampling schemes for various direct volume rendering approaches: (a) 2D slicing: sampling scheme for object-aligned slices, (b) 3D slicing: sampling scheme for viewport slices, (c) Ray casting: equidistant sampling scheme. 88

4.3 Acceleration structures: (a) recursive octree space subdivision, (b) Bounding Volume Hierarchy (BVH). 91

4.4 The box basis functions for V^2. 95

4.5 The Haar wavelets for W^1 95

4.6 Non-standard decomposition in three dimensions. 96

4.7 The HDD-CPU-GPU memory hierarchy. 98

4.8 Octree space subdivision in combination with multi-resolution wavelet representation. 100

4.9 Mapping information. 106

4.10 Multi-resolution scene on HIPerWall: (a) Toga data set. (b) distribution of wavelet levels among monitors: red corresponds to tiles of level 2 (1/64 = 1.5% of the original quality), green to level 1 (1/8 = 12.5% of the original quality). 108

4.11 Difference between levels of resolution: (a) level 2 before progressive refinement, (b) level 1 after progressive refinement. 109

4.12 Influence of octree depth on average rendering time and average level of detail: (a) average time measured to perform frustum clipping and volume assembling, (b) average percentage of original voxels displayed (including average wavelet level and variance). 110

4.13 (a) Loading time vs. wavelet level, (b) Loading time vs. octree height *Image courtesy of Nguyen [Ngu08].* 110

4.14 Storage scheme for BVHs. 115

4.15 Slab intersection test in two dimensions. Since $t_{near_{max}} > t_{far_{min}}$, $R(t)$ misses the boudning box. $R'(t)$ hits the bounding box and $t'_{near_{max}} \leq t'_{far_{min}}$ holds. 120

4.16 Coherent ray packet traversal: (a) speculative first active descent, (b) frustum test, (c) first active ray tracking. 122

4.17 Culling and pruning metrics based on the pre-integrated transfer function yield significantly smaller subtrees. A similar metric is used to inexpensively integrate constant subvolumes when possible. 124

4.18 Benchmarking data sets: (a) heptane, (b) zebrafish, (c) Richtmyer–Meshkov instability (RMI). 128

List of Tables

2.1 Key features of *AnyScreen* compared to other distributed rendering libraries (table follows Malburg [Mal09]). 14

2.2 Averages and standard deviations of performance time and error rates for HotWire 48

2.3 Averages and standard deviations of returns for Pong. 49

4.1 Data set characteristics and preprocessing times for octree depth 2 (64 bricks) and two levels of wavelet compression. 109

4.2 Data sets of the benchmark test suite. 128

4.3 Benchmark results. 129

Chapter 1

Introduction

1.1 Motivation

Over the last several years, ongoing technological advances led to declining prices for high-performance graphics interfaces, which enabled the assembly of large high-resolution displays at reasonable costs. Whereas in the past powerful workstations and supercomputers were required to control large display systems, today off-the-shelf commodity components can be used to drive multi-monitor or multi-projector display environments. Because of their extended screen real estate and large number of pixels, high-resolution displays have become a commodity in numerous application fields, such as industry design, scientific visualization, collaborative environments, immersive simulation scenarios, and public information systems. However, the challenges of setting up and controlling large high-resolution displays are manifold, and a wide range of issues have been encountered on the technological, interaction, and algorithmic side.

Since large displays environments are often driven by a cluster of render nodes, their control applications are usually distributed and executed individually on each node of the cluster. Hence, programmers need to face all peculiarities of distributed systems, such as finding effective ways to share data among cluster nodes or keeping the system consistent by synchronizing state information.

Large high-resolution displays combine the images of multiple display devices and require careful calibration to create the impression of a seamless screen area. Calibration can be quite complex, as it must consider the geometric alignment of the individual image tiles, as well as variations in color temperature and luminance. Therefore, the building of truly seamless tiled displays is often considered to be biggest challenge in large display research.

Furthermore, large high-resolution displays are a playground for research in human-computer interaction (HCI). Most commonly used interaction devices and metaphors have their origin in single monitor environments but perform poorly on large displays due to scalability issues. Therefore, new techniques are required to support the intuitive interaction of large user groups with virtual screen content.

Last but not least, a major reason for using large displays is the intention to visualize data sets whose resolution exceeds those of available individual monitors and graphics cards. Dealing with gigabyte- to terabyte-scale data sets at interactive frame rates requires efficient memory layout schemes and fast visualization algorithms that exploit the resources of modern graphics systems.

Although large high-resolution displays are becoming increasingly popular, there are hardly any valid standardized guidelines to set up large screen systems in terms of hardware and software. Most installations and applications are unique, custom solutions by individual research groups or companies. By providing novel insights and solutions to the challenges mentioned above, this dissertation aims at pushing the frontier towards more general design principles for large high-resolution displays.

Results were obtained in all stages of the system integration process over a period of $2\frac{1}{2}$ years as part of the research as a member of the IRTG 1131 at the University of Kaiserslautern, Germany.

1.2 Overview and Contribution

The following parts of this manuscript cover an extensive range of research topics, which on the top level can be divided into three categories: (1) *system*, (2) *interaction*, and (3) *visualization*. Each of these categories is covered by an individual chapter of this thesis. For the reader's convenience, instead of having one extended related work section for the entire dissertation, each chapter starts with its own related work section. Likewise, each chapter ends with its own conclusion section. Chapter 2 deals with system-related aspects of large high-resolution displays. Section 2.2 introduces AnyScreen, a distributed rendering library for high-resolution display clusters. Section 2.3 describes Tiled++, a hybrid display cluster combining liquid crystal display (LCD) technology and computer projectors. In particular, the scientific contributions described in this chapter are:

- Section 2.2
  - Description of AnyScreen, a new and lightweight C++ rendering framework for driving arbitrary display and display-in-display configurations.
  - Successful usage of AnyScreen in two sample applications.
- Section 2.3
  - Description of Tiled++, a hybrid approach to address the bezel issue of monitor-based tiled displays.
  - Successful evaluation of Tiled++ in a thorough user study and demonstration of increased user efficiency.

Chapter 3 deals with interaction techniques for large high-resolution displays. Section 3.1 provides an overview of existing large display interaction methods and challenges in terms of HCI. Section 3.2 introduces a new interaction technique for large displays based on two-dimensional visual tags and camera-enabled smart phones. The specific contributions of chapter 3 are:

- Section 3.1
  - Identification and clarification of design challenges for interaction techniques on large high-resolution displays.
- Section 3.2
  - Description of a highly scalable interaction approach for large displays using smart phones and two-dimensional visual tags.
  - Explicit support of large and diverse user groups.
  - Providing ways for customizing visual outputs and implementing security mechanisms on large high-resolution displays.
  - Successful usage of the interaction approach in two example scenarios.

Chapter 4 covers visualization-related aspects of large displays and in particular deals with volume visualization on high-resolution screens. Section 4.2 describes an out-of-core technique for handling large-scale data sets on distributed display clusters. Afterwards, section 4.3 describes three implementations of direct volume rendering systems exploiting hierarchies of bounding volumes to achieve interactive frame rates on large displays. The scientific contributions in this part of the manuscript are:

- Section 4.2
  - Implementation of an out-of-core technique to overcome the CPU-GPU memory gap on distributed display clusters. The approach exploits octree-based space subdivision in combination with wavelet-based multi-resolution techniques.
  - Successful implementation on a fifty tile display cluster.
  - Rendering of volumetric data sets larger than the texture buffer size of a single graphics card at significantly higher levels of detail than on a single desktop display.
  - Detailed evaluation of the technique.
- Section 4.3
  - Implementation of three direct volume rendering systems exploiting hierarchies of bounding volumes as acceleration structures on the GPU, CPU, and in a hybrid CPU/GPU system.
  - Identification and description of implementation details and challenges.
  - Detailed comparison of performances for various data sets.
  - Highly competitive performance rates of the CPU implementation for large-scale data sets.
  - Identification of guidelines for the implementation of future large display visualization systems.

Chapter 5 concludes the dissertation by giving a summary of the described work. Furthermore, possible directions for future research are pointed out.

Chapter 2

Setting up Large High-Resolution Displays

This chapter provides an introduction to large high-resolution displays and new concepts from a *system* point of view. Section 2.1 discusses hardware-specific options for setting up large high-resolution displays and investigates software requirements for controlling these systems and for developing applications that will run on them. As a part of this thesis work, a lightweight rendering library called *AnyScreen* is introduced in section 2.2. *AnyScreen*'s implementation is based on concepts described in section 2.1 and is able to drive almost arbitrary display and display-in-display configurations. Furthermore, it is capable of handling a variety of different stereo modes and adapts to rather unconventional hardware setups that are not supported by other libraries.

Tiled++ is discussed in section 2.3 and has been implemented using *AnyScreen*. It explicitly addresses the most fundamental problem of monitor-based tiled display walls: screen bezels. In monitor-based tiled walls, screen bezels are responsible for causing discontinuities in the displayed image, known as the French window effect, that disturb the impression of a seamless display area and lead to potential misinterpretation of displayed information. Tiled++ overcomes this shortcoming by combining tiled wall renderings with projector-generated overlay images that are projected onto the bezel grid. The calibration of a Tiled++ system is described in detail, and the results of a user study that was conducted in order to evaluate the usefulness of the approach are presented.

2.1 State-of-the-Art and Related Work

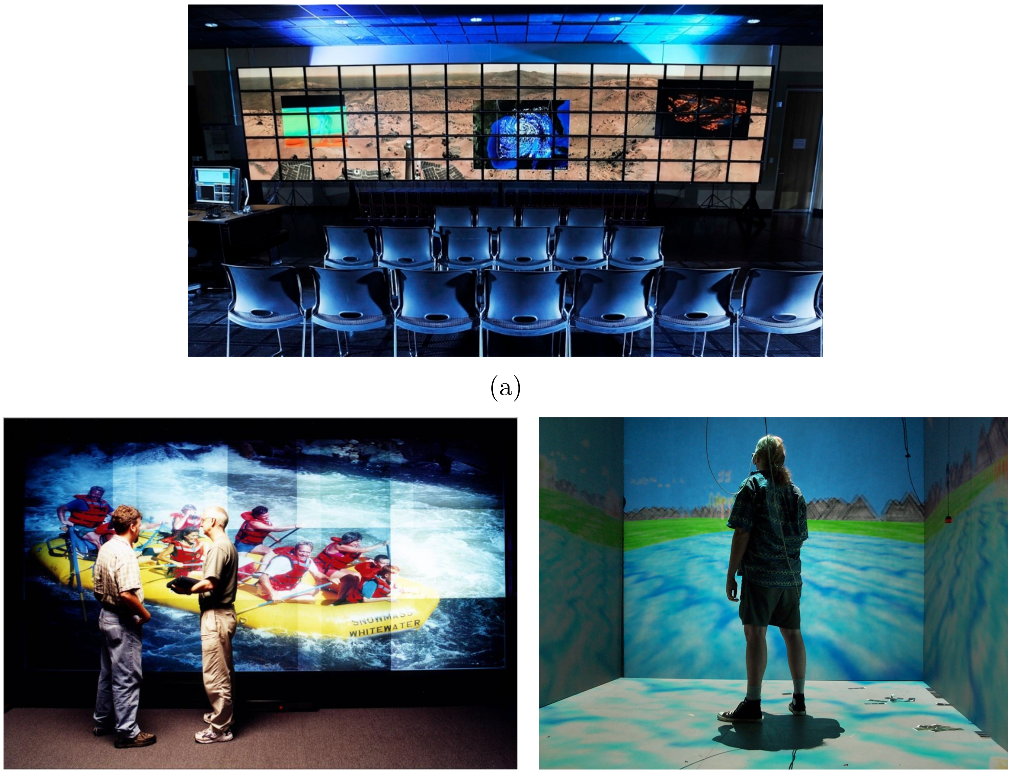
This section provides an overview of the current state-of-the-art in the field of large-high-resolution displays. Possible display configurations as well software packages that enable users to implement applications for these systems are investigated. The top 10 list of research challenges published in 2006 [NSS+06] reveals research in this area is still ongoing and many fundamental questions still need to be solved.

2.1.1 Hardware Configurations

A major characteristic of large high-resolution displays is the fact that they combine a large screen area with a high number of pixels. This is generally achieved by combining the images of multiple smaller display devices to form one unified display area. The pixel count of a large display is often too high for a single computer to drive, and a cluster of rendering nodes that are connected via a high-speed network is required to handle the resolution. The following section describes some of the possible hardware configurations for large high-resolution displays that are currently available and relevant for this thesis. A comprehensive overview of large display technology is given in the detailed survey paper by Ni et al. [NSS+06].

2.1.1.1 Monitor-based Tiled Displays

Monitor-based tiled display walls combine the resolution of multiple LCDs and easily achieve a combined resolution of several hundred megapixels. Since LCD screens are affordable now, monitor-based systems have become a viable way to set up large high-resolution displays, and compared to other approaches, their calibration is rather simple as it mainly consists of mounting all tiles on a proper monitor rack. A major problem of monitor-based systems are screen bezels, which lead to discontinuities in rendered images and result in an ever-present French window



(a)

(b)

(c)

Figure 2.1: Possible large high-resolution display configurations: (a) monitor-based tiled display wall, (b) projector-based tiled display wall *Image courtesy of Majumder et al. [MS05]*, (c) Cave Automatic Virtual Environment (CAVE$^{\text{TM}}$).

effect. The display wall in figure 2.1(a) depicts the world's currently largest monitor-based tiled display called Stallion [sta]. Stallion is set up at the Texas Advanced Computing Center (TACC) and comprises 307 megapixels in a 15 × 5 grid.

2.1.1.2 Projector-based Tiled Displays

Projector-based systems, such as the one shown in figure 2.1(b), consist of multiple computer projectors that are usually arranged in a grid and project onto a large screen. Compared to monitor-based tiled walls, projector-based systems are capable of forming a truly seamless display area, which requires careful calibration. The calibration is challenging, since projectors need to be adjusted in terms of inter- and intra-display luminance and color temperature differences. Also, geometric distor-

tions must be considered, which are due to lens artifacts, the location of projectors relative to the screen, and the actual screen geometry.
Stereoscopic powerwalls are a projector-based derivative of large-scale display systems. Stereoscopy is achieved by overlaying two projector images. Users wear polarized stereo glasses in order to separate the images for the left and right eye. Combining multiple displays for stereoscopic projection of larger, animated scenes provides additional challenges, such as frame synchronization and coordination of stereo parallax.

2.1.1.3 CAVEs™

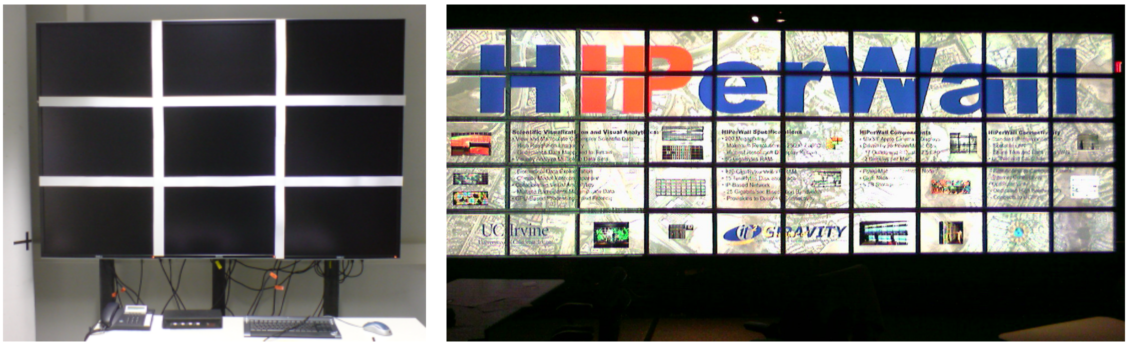
Cave Automatic Virtual Environments (CAVEs™) (see figure 2.1(c)) are non-planar projector-based installations that consist of up to six screens representing the sides of a cube or box. CAVEs™are immersive display systems, in which users are literally surrounded by the virtual environment. Often, they use additional devices, such as magnetic head or hand tracking, tracking suits or stereo glasses, to increase the perceived realism of the virtual environment.

2.1.2 Example Configurations

The focus of this thesis will be primarily on two particular display systems, since a majority of the presented work has been implemented on these systems. Section 2.1.2.2 and section 2.1.2.1 describe these systems in detail as they are referenced throughout the rest of this thesis.

2.1.2.1 3×3 Tiled Display Cluster

The first display is a 3×3 monitor-based tiled display wall (see figure 2.2(a)) set up at the University of Kaiserslautern, Germany. The system is driven by five nodes that are connected via Ethernet and contain Intel Core 2 Duo CPUs with 2.40 GHz, 2GB RAM, and two dual nVidia GeForce 7950 GX2 graphics cards with 1,024MB VRAM. Each monitor is a 30 inch DELL UltraSharp 3007WFP screen with a resolution of $2,560 \times 1,600$ pixels. The combined wall resolution is $7,680 \times 4,800$ pixels.



(a) (b)

Figure 2.2: Tiled wall installations: (a) 3 × 3 tiled display wall driven by 5 render nodes, (b) HIPerWall, a 200 megalpixel 10 × 5 display cluster driven by 25 render nodes.

Each of the two GPUs of a render node is connected to one screen of the wall, respectively. The one remaining graphics card at the master node (see section 2.1.3.2) is used to drive the I/O monitor that users interact with or an additional computer projector, as described for the Tiled++ display in section 2.3.

2.1.2.2 10 × 5 HIPerWall Cluster

The second visualization cluster is a 200 megapixel monitor-based tiled display wall set up at the University of California, Irvine called Highly Interactive Parallelized Display Wall (HIPerWall) [hip] (see figure 2.2(b)). HIPerWall is driven by 25 PowerMac G5 computers, each equipped with nVidia GeForce 6800 and nVidia Quadro FX4500 graphics cards possessing 256MB of video RAM. The 25 compute nodes drive fifty 30 inch Apple Cinema Displays that have been arranged in a 10 × 5 grid. Each screen has a native resolution of $2,560 \times 1,600$ pixels. This means that the wall can display images, videos, or rendered scenes at a maximum resolution of $25,600 \times 8,000$ pixels. A designated front-end computer launches applications and synchronizes the render nodes that communicate via the IP protocol. The current operating system is Mac OS X Tiger.

2.1.3 Distributed Rendering Libraries

As described previously, large high-resolution displays are usually driven by a cluster of render nodes that are connected via a high-speed network. As a consequence, applications developed for these systems are actually distributed and require support through appropriate middleware. *Distributed rendering libraries* and *data streaming software* provide interfaces to application developers that provide a layer of abstraction between the software and the underlying hardware and free them from having to deal with too many cluster-specific implementation details, e.g., how to split-up a scene among the tiles of a large display, how to synchronize nodes, and how to keep a consistent state between cluster nodes. Whereas distributed rendering libraries are used to implement typical rendering tasks, such as volume rendering, ray tracing, and visualizing geometric objects, data streaming software streams any kind of multimedia data, such as videos or photo streams. Unless otherwise mentioned, the remainder of this thesis deals primarily with distributed rendering on large displays. Some of the best known software packages are CAVELib [Mec], VRjuggler [BJH+01], Syzygy [SG03], Jinx [SZ04], OpenSG [Rei02], Chromium [HHN+02], Garuda [NHN07], and CGLX [DK], which is described in section 4.2.2.1. For a comprehensive overview of further software packages see Ni et al. [NSS+06] and Raffin et al. [RS06].

2.1.3.1 Data Distribution

Distributed rendering libraries for large high-resolution displays can be characterized in various ways. One of them is in terms of the graphics pipeline stage, in which geometry is assigned to the tiles of a display cluster [MCEF94]. A detailed description of the graphics pipeline and the conversion of geometry data into pixel data is given in section 4.1.3.

Sort-first. Libraries implementing a sort-first strategy assign raw primitive data in the primitive assembly stage. This stage converts geometry data from three-dimensional object space into normalized screen space. After each node has been assigned its parts of data, the remaining pipeline stages are executed independently.

Sort-middle. Systems implementing a sort-middle approach assign screen-space primitives between the primitive assembly stage and the rasterization stage.

Sort-last. Frameworks based on a sort-last approach assign actual pixel/fragment information in the rasterization stage. The rasterization stage is the last stage of the pipeline in which a series of fragment tests (e.g., scissor, alpha, and depth tests) are performed to determine whether and how a fragment is visualized as a pixel on the screen.

2.1.3.2 Execution Modes

Another way to classify rendering libraries is in terms of the underlying execution mode [CCF+01, CCL+01]. Two different approaches have emerged: master-slave and client-server systems (see figure 2.3).

Master-slave. In the master-slave model every node of the cluster runs one instance of the application and performs the same operations on the data. A designated node, the master, is responsible for handling additional tasks, like processing user input and output, and forwarding this data to the other nodes. Because every node has access to the data, the network overhead of this approach is extremely low.

Client-server. In the client-server model a single instance of the application runs on the client node, which is responsible for sending data packages to the rendering servers. In contrast to the previous approach this model makes better use of the computational power of the render nodes. However, with increasing data size, network bandwidth may pose a limit to this technique.

2.1.4 The Current Top 10 Research Challenges

In 2006 Ni et al. [NSS+06] formulated what they considered to be the top 10 research challenges in the field of large high-resolution displays. Their list is still up-to-date and can be considered a roadmap for future research. The points they identified are:

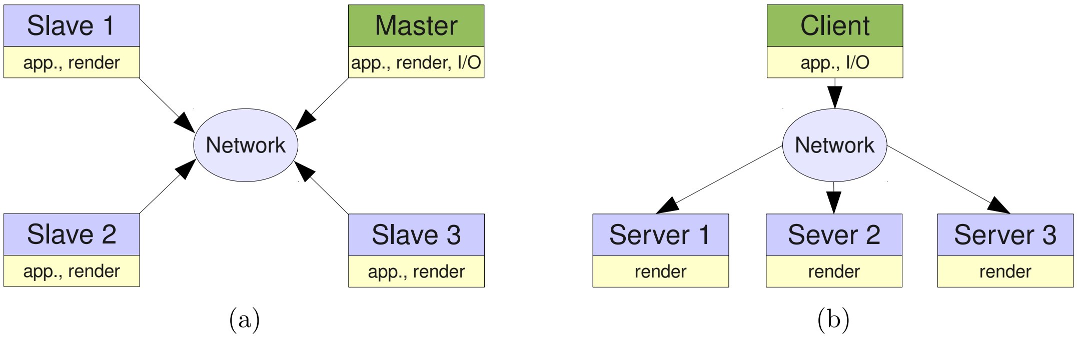


Figure 2.3: Possible execution modes for distributed rendering libraries: (a) master-slave execution mode, (b) client-server execution mode.

1. *Truly seamless tiled displays.* Setting up truly seamless tiled displays is challenging. Monitor-based systems inherently suffer from screen bezels and projector-based systems require a careful calibration.
2. *Stereoscopic large high-resolution displays.* Stereoscopic systems as described in section 2.1.1 enhance the degree of realism in an immersive environment. What are the best ways to set up these environments and what applications benefit from stereoscopy?
3. *Easily reconfigurable large high-resolution displays.* Ideally, large high-resolution display environments can be extended or altered dynamically and reconfigured automatically.
4. *High-performance cluster rendering.* The performance of cluster applications depends on their implementation and the support of the underlying middleware.
5. *Scalability.* Do systems scale to an arbitrary number of tiles?
6. *Design and evaluate large high-resolution display groupware.* How can high-resolution displays be used to support large user groups and collaboration among them?
7. *Effective interaction techniques.* Many traditional interaction techniques are not suited for tiled displays and need to be extended or adapted.

8. *Perceptually valid ways of presenting information on a large display.* How can applications make effective use of the increased pixel count and present information adequately?
9. *Empirical evidence for the benefits of large high-resolution displays.* Which tasks can be accomplished faster on tiled displays than on desktop PCs or laptops?
10. *Integrating large high-resolution displays into a seamless computing environment.* How can large high-resolution systems be integrated into an already existing infrastructure?

The work presented in this thesis addresses a majority of points from this list. The *AnyScreen* framework presented in section 2.2 is a distributed rendering library that scales to displays of arbitrary size (top 5) and has also been used to drive stereoscopic powerwalls (top 2). Various display configurations and display types are supported through the easy to use configuration tool (top 3) that allows to save and load particular sets of configuration parameters as XML files.

The Tiled++ system discussed in section 2.3 is a projector-based technique that deals with the bezel problem of monitor-based tiled walls in order to make them more seamless (top 1). The user study conducted in this context reveals some of the benefits and drawbacks of tiled display systems in navigation- and perception tasks (top 9).

Chapter 3 investigates techniques for large display interaction (top 7) that scale with the display- and user group size (top 5). The tag-based interaction approach presented in section 3.2 demonstrates ways of presenting information that is customized based on the role of a user and the respective group membership of that user on large high-resolution displays (top 8).

Chapter 4 entirely focuses on high-performance algorithms for volume rendering on large high-resolution displays (top 4). The question of how to handle giga-scale data sets on a compute cluster is addressed from a resource point of view as well as based on the issue of how to render them interactively on all tiles of a display wall.

Feature	**Chromium**	**Garuda**	**VRJuggler**	**CGLX**	**AnyScreen**
OpenGL	yes	encapsulated	yes	yes	yes
Side-by-side	no	limited	yes	no	yes
Interlaced	no	no	no	no	yes
z-map	no	no	no	no	yes
Anaglyph	no	yes	no	no	yes
Windows	yes	yes	yes	no	yes
Linux	yes	N/A	yes	yes	yes
MacOSX	no	N/A	yes	yes	yes
Cluster	yes	yes	yes	yes	yes
Keystone	no	no	no	no	yes
Masking	no	no	no	no	yes

Table 2.1: Key features of *AnyScreen* compared to other distributed rendering libraries (table follows Malburg [Mal09]).

2.2 AnyScreen - A Highly Scalable Distributed Rendering Framework for Arbitrary Display and Display-in-Display Configurations

Despite the increasing popularity and wide variety of large high-resolution displays, the ultimate general purpose solution to drive all different types and configurations does not exist yet. Various commercial and non-commercial software solutions are available, differing mainly in the way they perform distributed rendering and in the display configurations they are able to control. Compared to other libraries, *AnyScreen* [DTS$^+$09], the library introduced in this section, was specifically designed as a lightweight architecture with a small and simple to use interface. It consists of a set of C++ classes, available through static and dynamic libraries. The focus of *AnyScreen* is on platform-independence, the ability to drive arbitrary display configurations (monitor-based, projector-based, and hybrid), and the support of various stereo modes. Table 2.1 compares *AnyScreen* to some other libraries and shows how it integrates into the landscape of distributed rendering software (the various stereo modes as well as the keystone and masking feature are described in subsequent sections).

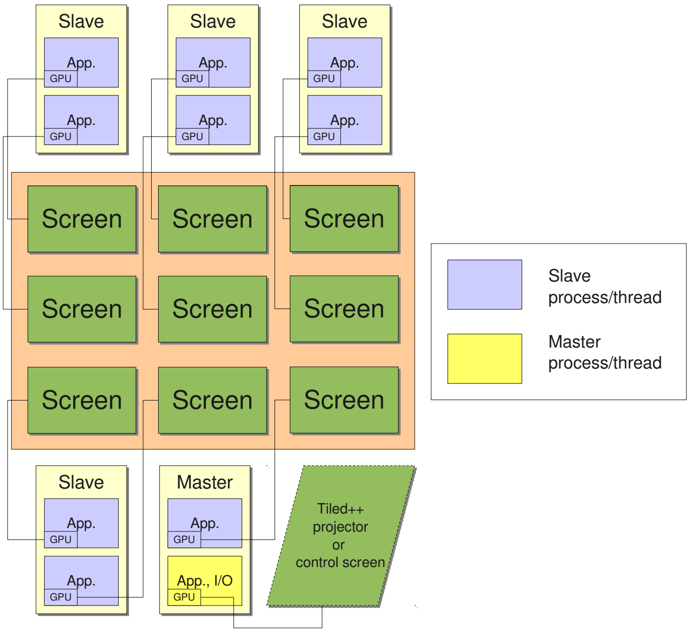


Figure 2.4: Thread/process model of *AnyScreen* applications executed on a 3 × 3 tiled display.

2.2.1 Framework Architecture

AnyScreen is designed as a master-slave system that features a static sort-first strategy during the primitive assembly stage. Figure 2.4 illustrates a schematic view on the thread/process model of the 3 × 3 display wall when executing an *AnyScreen* application.

Each tile is driven by one instance of the application, i.e., an application process or thread. Since each personal computer (PC) is equipped with two graphics cards, two application processes are executed in parallel on each compute node. During program execution, all threads determine which part of the final image they have to render. This information is stored in an XML file for a fixed configuration. The file provides the render parameters for each tile and is read when the application is launched. Since the configuration is independent from the actual application, the same XML file can be used for different applications. Conversely, each application can switch between different display setups simply by switching between configuration files. This allows for easy porting of applications to other display systems.

A designated node runs the master thread while all other processes are slave processes. The master thread keeps track of application-specific variables and distributes data to the rendering nodes. In particular, this includes any kind of user input, like mouse or keyboard events. In case of changes to the display setup, the master broadcasts the new configuration, as well as the index of the current render function. It also synchronizes buffer swaps by waiting for a *frame ready*-signal from each process, before triggering a simultaneous swap. Since only nine out of ten GPUs are required to drive the 3 × 3 display, the remaining graphics card can be used to drive a control monitor that is not actively included in the display array, or, as in the case of Tiled++ (see section 2.3), an additional computer projector.

2.2.1.1 Implementation and Configuration

The graphics- and I/O related functions of the framework have been implemented using *Simple DirectMedia Layer* (SDL) [Sim]. SDL is a cross-platform multimedia library with bindings to most common programming languages, including C++. SDL is similar to the OpenGL Utility Toolkit (GLUT) [SWND07] but more comprehensive, and it provides low level access to audio, keyboard, mouse, joystick, and 3D hardware via OpenGL.

The *AnyScreen* network- and communication module is based on the MPICH2 [Arg] implementation of the *Message Passing Interface* (MPI), a library specification for message passing that is commonly used in high-performance computing. MPICH2 provides various functions to support the implementation of parallel cluster applications. The OpenGL buffer swap, for example, can be synchronized through a barrier synchronization at the end of the render function as shown in listing 2.1:

Listing 2.1: Synchronizing OpenGL Buffer Swaps with MPI

```
int TileRenderer::render( void* data, unsigned int size )
{
  ...
  //barrier synchronization
  MPI_Barrier( MPI_COMM_WORLD );
  //swap buffers
  SDL_GL_SwapBuffers();
}
```

All application processes stall and do no proceed until the last process has reached the barrier.

Likewise, listing 2.2 illustrates how a broadcast is implemented that lets the master process send an array of data to all of its slave processes. The array can contain any kind of data, such as current mouse coordinates, names of files to load, command line parameters, or random numbers that were generated at the master and have to be broadcast in order to keep the system in a consistent state.

Listing 2.2: MPI Data Broadcast

```
//data array, size of data array (number of bytes)
int TileRenderer::distributeData(void* data,
  unsigned int size)
{
  if( this->isMaster() ) //master
  {
    //broadcast size
    MPI_Bcast(&size, 1, MPI_UNSIGNED,
      0, MPI_COMM_WORLD);
    //broadcast data array
    MPI_Bcast(data, size,MPI_BYTE,
      0, MPI_COMM_WORLD);
    return size;
  } else //slave
  {
    //number of bytes to be received
    unsigned int count;
    //get number of bytes
    MPI_Bcast( &count, 1, MPI_UNSIGNED,
      0, MPI_COMM_WORLD );
    if( count > size ) //not enough space
    {
      //temporary buffer
      char* tmp = new char[count];
      //get data
      MPI_Bcast( tmp, count, MPI_BYTE, 0,
        MPI_COMM_WORLD );
      //delete data
      delete[] tmp;
      //return error code
      return -1;
    } else //enough space
    {
      //get data
      MPI_Bcast( data, count, MPI_BYTE,
      0, MPI_COMM_WORLD );
      //return number of bytes successfully received
      return count;
    }
  }
}
```

The final image on the display wall is composed of various tiles. Each application process on a compute node must know which part of a scene it has to render. This information is retrieved from the configuration parameters that are stored in the aforementioned XML configuration files. Configuration files can be generated and edited with a Java-based configuration tool whose graphical user interface is depicted in figure 2.5[1]. The window is vertically divided into two areas. The right area depicts

---

[1] The user interface is in German language. Labels translate as follows: "Bildschirm = Screen", "Datei = File", "Bearbeiten = Edit", "Konfiguration = Configuration"

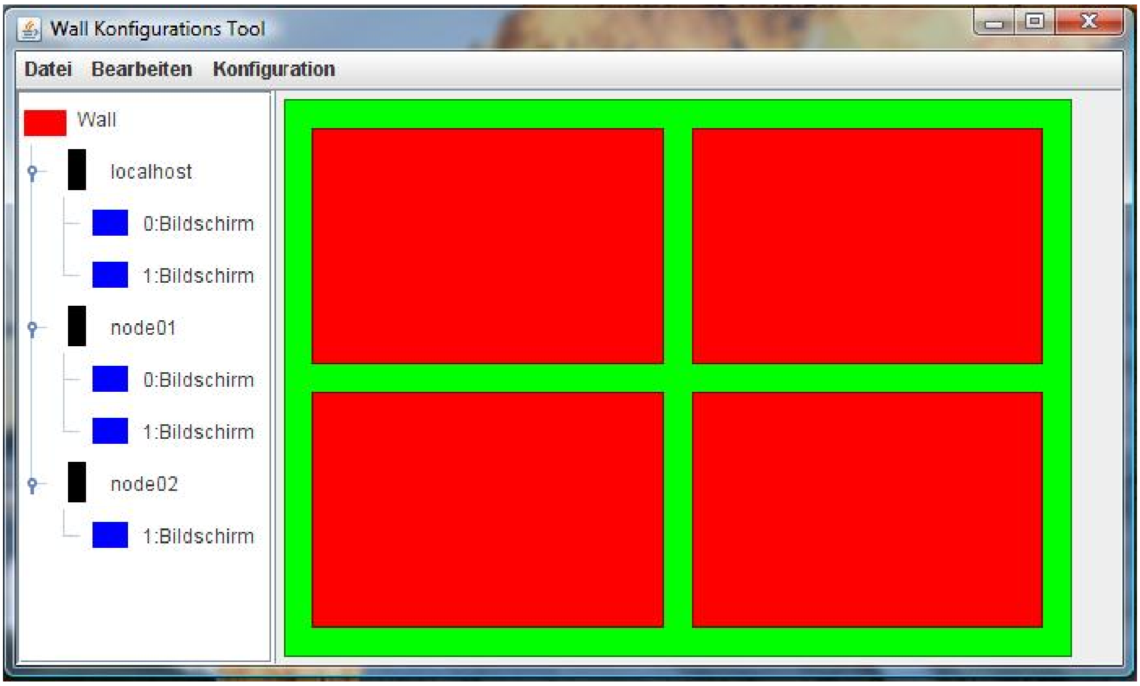


Figure 2.5: The *AnyScreen* configuration tool can be used to generate and edit XML configuration files for arbitrary display configurations.

the current display configuration, including the size, position, and orientation of all tiles. The left area contains a tree-like hierarchy, showing all nodes of the cluster including the monitors that are attached as primary or secondary displays. An item can be selected and edited by right-clicking it in the tree hierarchy. Alternatively, one can select single tiles from the illustration on the right side. XML support is provided by the *Xerces* XML library [Xer].

Listing 2.3 shows an example XML configuration file for a tiled display setup. As can be seen in the example, the superior structure comprising all displays is a *Wall*, describing the complete (virtual) visualization area. Its attributes are *SizeX* and *SizeY*, the size of the entire display area in x and y direction. Any individual display must have the attributes *DisplayNr* and *NodeNr*. *DisplayNr* is the zero-based index of the physical display connected to a machine, e.g., 0 = primary monitor, 1 = secondary monitor.

Listing 2.3: XML Config File

```
<Wall SizeX=”290.0” SizeY=”230.0”>
  <Display DisplayNr=”0” NodeNr=”0”
    PosX=”0.0” PosY=”0.0” SizeX=”290.0”
    SizeY=”230.0” Angle=”0.0”
    IsLowRes=”false”>
    <WallPosition>
      <BottomLeft X=”−1.0” Y=”−1.0”/>
      <BottomRight X=”1.0” Y=”−0.8”/>
      <TopRight X=”1.0” Y=”1.0”/>
      <TopLeft X=”−1.0” Y=”0.6”/>
    </WallPosition>
    <Correction>
      <BottomLeft X=”−0.6” Y=”−0.8”/>
      <BottomRight X=”1.0” Y=”−0.6”/>
      <TopRight X=”0.7” Y=”0.4”/>
      <TopLeft X=”−0.7” Y=”0.7”/>
    </Correction>
    <Stereo Type=”SideBySide” FocalLength=”15.0”
       EyeSeparation=”0.08” Eye=”Left”
       DisplayNr=”3” NodeNr=”1”>
    </Stereo>
  </Display>
</Wall>
```

NodeNr refers to the identifier of the application process that is responsible for generating the corresponding tile rendering. The identifiers are assigned sequentially among nodes/processes by the MPICH2 library. A display with only these parameters is an *inactive* display, i.e., it is part of the configuration but does not display any content. This can be useful if the master process is supposed to be used for synchronization and input control only, but not for generating visual output. If a display is supposed to be *active*, the attributes *PosX*, *PosY*, *SizeX*, *SizeY* and *Angle* have to be provided, specifying the position, size and orientation with respect to the entire display area. In most cases, the angle value of a monitor is a multiple of 90 degrees though in general arbitrary rotation angles can be defined. A display can also be specifically defined as a *LowRes* display. This is used for hybrid configurations where high-resolution displays are embedded into low-resolution projector images. In this case, parts of the low-resolution images overlap some of the high-resolution areas, and the common parts are masked out in the low-resolution image in order to avoid interference. Furthermore, it is possible to add *WallPosition* elements that define non-rectangular displays by the positions of their corners. To accommodate display areas that are smaller than the physically possible display size, an additional keystone correction parameter can be defined, containing the corners of the actually used display area in screen coordinates of the display. An example using a combination of all these features is described in detail in section 2.3.

2.2.1.2 Resource Management

One of the primary application fields for large high-resolution displays is the visualization of large data sets. The question is how to manage and store data sets that consist of thousands of files and comprise gigabytes of data. Depending on the operating system support, *AnyScreen* applications can chose between two options:

1. Files can be stored and accessed on a *centralized* volume, such as a network drive. The advantage of this strategy is that files have to be saved only once but can be accessed from any node in the cluster. However, the application and operating system have to deal with potentially concurrent file access. While this is not so much of a problem when reading files, simultaneous write-operations can lead to irreversible destruction. Furthermore, network bandwidth may become a bottleneck in the system.
2. Alternatively, each PC can store copies of the data on its own *local* hard drive. This increases the total amount of disc space that a data set occupies in the cluster, but restricts the problem of simultaneous file access to the threads that are executed on a node of the cluster. File access is faster and network traffic is reduced, as data does not have to be transferred via the network anymore.

For most applications a mix between these two strategies is used. Small system-related files, such as configuration files or MPI machine files (see section 2.2.3.1), are stored on a network volume that is hosted on the master PC. Data sets are stored either centralized on the network volume or locally on the cluster nodes, depending on their size and the performance requirements of the application.

2.2.2 Stereo Modes

The *AnyScreen* framework supports several stereo modes which can be configured for a use on single display systems, such as stereoscopic desktop displays, as well as for large high-resolution displays, such as powerwalls or CAVEs™. Since the stereo settings are not influenced by the tiled settings and vice versa, it is also possible to setup less common configurations, like stereoscopic tiled walls using anaglyph rendering, or even tiled walls consisting of several autostereoscopic displays. The stereo formats that are supported by the framework are:

Anaglyph. This mode renders stereo image pairs with modulated, chromatically opposite colors. *AnyScreen* supports two different formats of anaglyph stereo: grayscale- and color mode. In grayscale mode, the images are rendered with corresponding colors for each eye, resulting in a gray stereoscopic image. In color mode, the colors of each of the two stereo images are modulated, yielding a color image of the visualization, though with slightly falsified colors. Despite the color deviations, anaglyph stereo mode has the advantage that it can be used for any kind of display.

Vertical/Horizontal interleaving. Vertical interleaving is a rendering mode often used by autostereoscopic desktop monitors. Here, the stereo image pairs are interlaced in alternating columns. A lenticular lens in front of the actual screen diffracts the left and right stereo images to different angles in front of the display. When viewed from the correct position, each eye can only see one of the images, resulting in a stereoscopic effect. When using horizontal interleaving, the stereo pairs are displayed in alternating rows. In this case, a filtering foil is attached to front of the screen with different polarization for even and odd numbered rows, and users are required to wear corresponding filter glasses.

Z-map. This is a specific stereo mode used by Philips WOWvx Technology [Phi08]. The generation of the stereo views is handled by the display itself. However, to achieve this, the display requires each frame as a combination of a normal color image and a corresponding grayscale depth map.

Side-by-side. The side-by-side mode utilizes not one, but a pair of displays to generate stereo images. One display exclusively shows the image for the left eye, while the other one shows the image for the right eye. The images are then displayed by two projectors with polarized filters mounted in front of them. By projecting both images on the same canvas and observing them through matching filter glasses, the stereoscopic effect is achieved. A special silver screen must be used to preserve polarization.

2.2.3 Rendering Process

This section explains how previously described features integrate into the *AnyScreen* rendering process. The initialization stage is responsible for preparing the system to enter the main rendering loop, i.e., setting up network connections, retrieving display parameters, and establishing a consistent state between application threads. The steps of the rendering pass mainly depend on the current display mode and whether stereoscopy is supported or not.

2.2.3.1 Initialization

From a technical point of view, *AnyScreen* applications are MPI applications. The system startup is initiated by launching the master process on the master PC and passing an MPI-specific *machinefile* containing a list of involved PCs, their IP addresses, and number of threads that need to be run on each machine. After establishing network communication between all machines, the application processes are started via MPI. Each thread first reads the display settings from the XML configuration file, creates application windows, and initializes its OpenGL rendering context. The information read from the XML configuration file is used by each thread to determine its own viewing frustum and projection matrix, which is derived from the wall-specific overall frustum defined by the framework's `setPerspective()` function. Before entering the rendering loop, the master broadcasts any further required parameters to its slave processes. After that, the framework is ready to begin rendering.

2.2.3.2 Rendering Pass

The various steps of the framework's render cycle are depicted in figure 2.6. At the beginning of each cycle, the OpenGL buffers, i.e., color and depth buffer, are cleared from the contents of the previous cycle. Because program parameters, such as mouse coordinates, might have changed, all necessary parameters are broadcasted by the master process in the distribution- and synchronization step in order to ensure all threads are in a consistent state at the beginning of the render process. *AnyScreen*

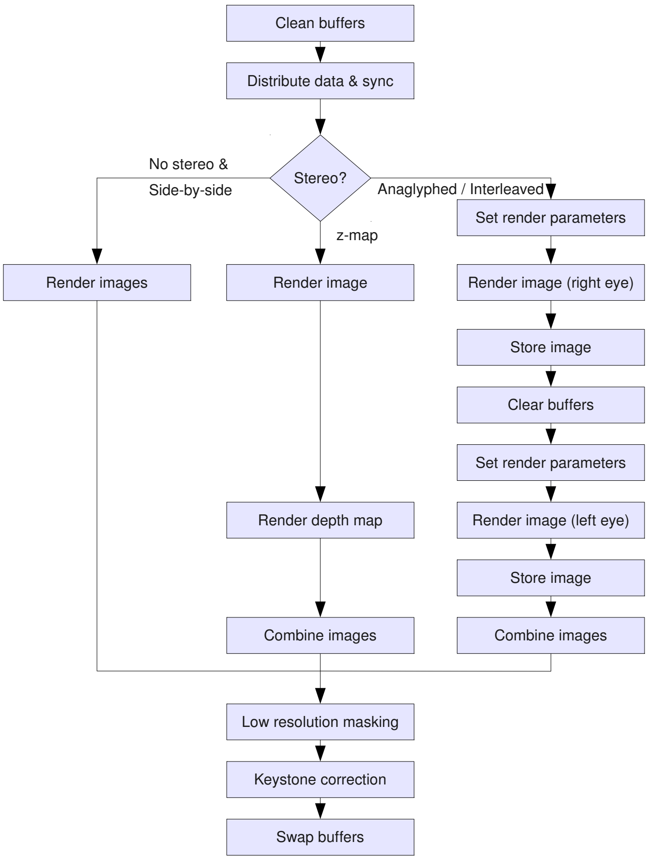


Figure 2.6: Logical render pass diagram.

only handles information concerning the tiled display configuration automatically. Therefore, other data, like user input, has to be distributed explicitly via broadcast by calling the `distributeData()` function discussed in listing 2.2.

The further execution depends on the render mode set for each display. For side-by-side stereo or if no stereo mode is set, the system renders each tile directly without any additional steps. Since projection matrices for both modes have been determined during the initialization, no further actions are required.

If z-map stereo mode is enabled, the system also starts by rendering the current frame. Additionally, the z-buffer is rendered to a depth map texture and combined with the original image to yield the correct WOWvx display format.

The passes for anaglyph and interleaved stereo mode are a bit more complex, because each frame has to be rendered twice as described in section 2.2.2. In both cases, the system first sets the according parameters for the right eye, such as projection matrices and color settings. The frame is rendered to the framebuffer which is then stored in a texture object. After clearing the buffers, the process is repeated for the left eye image. In the final stage, the two images are combined by overlaying (anaglyph) or interleaving them.

Once the image is complete, masking operations are performed for hybrid systems to prevent the images of low-resolution displays from overlapping with high-resolutions display areas.

Finally, a keystone correction according to the parameters defined in the XML file is applied. After that, the render node sends a buffer ready signal and waits for the synchronized buffer swap.

2.2.3.3 Application Development

AnyScreen is designed to put as few restrictions and requirements on application developers as possible. To maintain well-defined states among nodes, rendering is done by registering callback functions, similar to GLUT's `glutDisplayFunc(void (*func)(void))` function. The user has to provide an *initialization callback* that is called directly after initialization to set basic OpenGL parameters, as well as defining the perspective settings for rendering. Also, the application has to provide at least one *rendering callback* that is executed once (or for some stereo modes twice) per frame. Optionally, the user can provide a set of additional render functions and switch between them, for example, when the state of the application changes. Furthermore, it is possible to set different rendering functions for different displays, thereby allowing to use single displays in a tiled configuration to show additional information, e.g., status information of the application.

2.2.4 Examples

Steffen et al. [SEDD09] describe an application whose implementation is based on the *AnyScreen* framework. Their hybrid approach is an extension of the focus+context screen by Baudisch et al. [BGS01] into multiple dimensions. Instead of embedding a 2D high-resolution area into a 2D low-resolution context, as done for focus+context screen, they embed a 2D high-resolution projector image into a 3D context on a stereoscopic powerwall. The motivation for this hybrid setup is the fact that the ability of stereo projection systems to display detail and especially text is limited, because the wide viewing angle required for immersion comes at the expense of detail. However, with the embedded 2D high-resolution image it is possible to view those contents once they appear at the center of the powerwall screen, where the focus area is located.

Another application based on *AnyScreen* is the Tiled++ system described in section 2.3. *AnyScreen* is the ideal platform for driving such unusual display configurations, since the system combines high-resolution and low-resolution display devices, and requires a careful projector calibration, as well as exact masking of certain areas to reduce visual interference between contents displayed at different levels of resolution.

2.2.5 Discussion

AnyScreen is a lightweight, yet flexible rendering framework that provides a set of C++ classes and functions for the development of OpenGL applications on large high-resolution displays. With minor changes to standard OpenGL applications, *AnyScreen* is able to drive a wide variety of display configurations, including monitor-based tiled systems, projector-based displays, and hybrid installations. *AnyScreen* scales to displays of arbitrary size and tile configuration and supports various stereoscopic render modes. As shown in the examples, the library is able to drive rather unconventional display setups of mixed dimensionality and resolution. Future improvements of the library are concerned with an automatic system calibration using cameras to assist the creation of configuration files. Initial attempts in this direction have been made and are described in section 2.3.3.3.

2.3 Tiled++ - An Enhanced Tiled Hi-Res Display Wall

Monitor-based tiled displays offer a way to set up systems that comprise a high number of pixels at affordable costs. Whereas the installation of such display clusters is pretty straightforward, the strongest argument against these systems are monitor bezels that disturb the impression of a seamless display.

A lot of research has focused on how to make truly seamless projector-based tiled displays, however, similar efforts regarding monitor-based display walls have been rather limited, and most conventional solutions either ignore the bezels and display all pixels, causing objects to become distorted, or eliminate the pixels that would normally fall under the bezels, causing pixels to be missing in the display of static images. For wideband displays, Mackinley et al. [MH04] presented novel interface techniques and seam-aware applications that take the position of bezel bars into account. Ball et al. [BVC$^+$05] literally removed parts of the monitor frames in order to minimize gaps between neighboring displays. This approach might not always be a valid solution for obvious reasons, e.g., when the monitor bezel serves as a heat sink for the backlight. In navigation tasks it is important that mouse pointers can cross bezels when moving between screens without irritating the user. Baudisch et al. [BCHG04] compensate offset and warping effects by applying appropriate transformations to the movement of the cursor.

This section introduces a new and scalable approach called Tiled++ [ETO$^+$10] that neither leads to discontinuities nor significant loss of image information. By projecting directly onto screen bezels, it is possible to significantly reduce the negative effects of monitor frames. The approach combines LCD-based tiled displays with projected image information.

The following describes the motivation for coming up with new bezel strategy and explains the setup and calibration of a Tiled++ system in detail. Furthermore, the results of a user study are presented that was conducted in order to evaluate the usefulness of the approach.

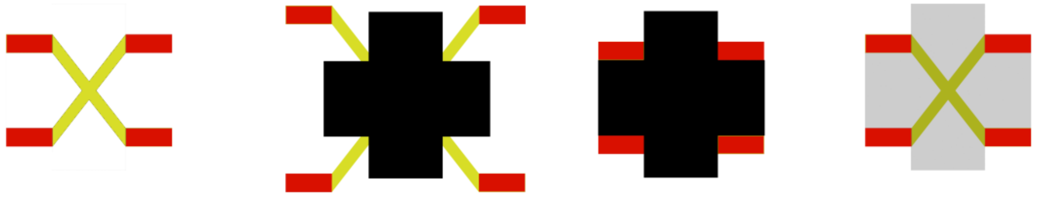

(a) (b) (c) (d)

Figure 2.7: Semantic problems: (a) original scenario, (b) discontinuities due to the offset strategy (image appears distorted/expanded), (c) overlays are hiding the wire joint (missing pixels), (d) missing information provided by Tiled++.

2.3.1 The Bezel Effect

The bezel effect of monitor-based tiled displays is caused by monitor frames that cut across the screen area and disturb the impression of a seamless display. The frames are necessary, since they contain control elements that are needed to drive the display and serve as heat sinks. Bezel-free screens cannot be manufactured with current display technology, although thin-bezel monitors reduce their effect in tiled display setups. As a result, a French window effect occurs, which is still one of the major disadvantages of monitor-based large high-resolution display systems[2]. Traditionally, there are two ways to deal with the effect:

1. The **offset technique** simply ignores the bezels and their effect on the continuousness of a scene. A mouse cursor that crosses the border of a monitor abruptly jumps into the neighboring tile. Objects that fall into that region appear to be stretched by an offset that is equal to the bezel size plus the distance between monitors. This creates strange effects and appears intolerable at a first glance. However, the advantage of the offset technique is that no image information is lost at the border between monitors. This mode is particularly useful when text documents are displayed. No text will be missing in this mode.

---

[2] In the future, it might be possible to build bezel-free screens with Organic Light Emitting Diodes (OLEDs). First models have been presented, however, at the current point, the technology is still in the early stages of development.

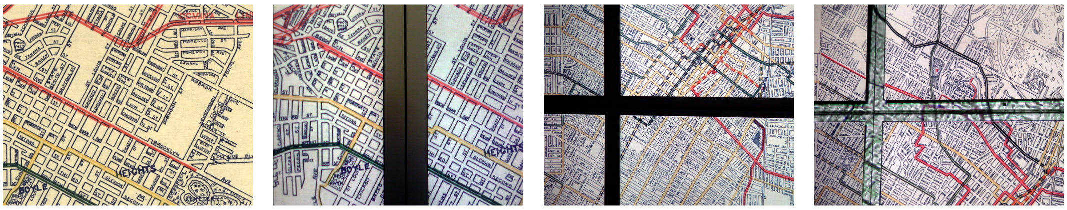

(a) (b) (c) (d)

Figure 2.8: Semantic problems on HIPerWall: (a) original image (Los Angeles Railway System of 1906), (b) discontinuities due to the offset method (diagonal lines appear to be disconnected or incorrectly aligned), (c) overlay technique (pixels under bezels have been eliminated, leading to missing information), (d) missing information provided by Tiled++.

2. The **overlay technique** tries to minimize the bezel effect by pretending the bezel grid is an overlay of the image. In contrast to the offset technique, mouse cursors and other objects disappear between screen borders. While this seems to be preferable to the discontinuities caused by the offset technique, potentially important information can be ”hidden” between tiles.

A choice between both methods depends on the application, but in any case, user perception is affected by the chosen technique. This circumstance is referred to as *loss of semantics* and further clarified in the electric circuit example in figure 2.7.
With the techniques described above, the visualization of a pair of crossing wires (see figure 2.7(a)) results in the images shown in figure 2.7(b) and figure 2.7(c). Users working in front of a tiled display where the offset method is applied, are able to make out the wire joint. However, they must keep in mind that what they perceive is a distorted image of the original wire joint. Applying the overlay strategy does not even give users a hint of the crossing. This is a potential source for failures and misinterpretation of scenes.
Figure 2.8 illustrates the same issue in a real world scenario. The image shows a map of the Los Angeles Railway System of 1906 displayed on the 200 megapixel HIPerWall tiled display. As with most road maps, the image contains many horizontal, vertical and diagonal lines. Using the offset approach (figure 2.8(b)) leads to misinterpretations, because the human brain tends to extend lines that are invisible in a straight line and assumes that a line on the other side of the bezel that is in

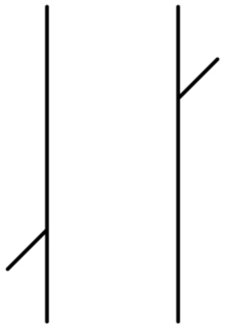


Figure 2.9: The Poggendorff illusion. Although both oblique lines are in fact collinear, they appear to be offset.

the same line of sight, must be connected, which is not the case here. In fact, since many streets on the map run parallel, the brain tries to connect roads that are in fact not connected. In figure 2.8(c), the pixels under the bezels have been removed, assuming the bezels serve as an overlay which covers these pixels and renders them invisible.

Another more subtle effect can lead to additional misinterpretation, especially when using the overlay method. In neurosciences it is known that two collinear oblique lines separated by two vertical parallels can cause failures of perception in the human brain. The collinear lines appear to be offset, even if in fact they are aligned (see figure 2.9). This effect was discovered by the physicist J. C. Poggendorff in 1860 and is known as *Poggendorff illusion*. Although the reasons for the effect are not yet well understood, it has an impact on the visualization with LCD-based tiled displays, as Poggendorff situations arise at the interface between two monitors. The evaluation conducted in section 2.3.8 investigates the extend of the effect on the performance of users more closely.

Further potential misinterpretations that may arise due to the chosen bezel strategy can be characterized by Ware's [War04] classification of *preattentively processable features*. Ware defines four categories of optical attributes whose visual identification is performed in a very short time lapse: color, movement, spatial localization and form. Especially form features, such as line length, line collinearity, size and spatial grouping are affected and demand for a precise evaluation.

Tiled++ is a technique that neither distorts objects across the boundary of monitors nor covers important information without giving a hint of what is happening beneath the bezels. This is achieved by combining the availability of all pixel information, which is characteristic to the offset method, with the continuousness of the scene, given by the overlay strategy. Tiled++ provides the missing context information by projecting directly onto the bezels using additional computer projectors. The idea is sketched in figure 2.7(d). With Tiled++, the crossing is visible without distortions or full loss of pixels. The enhanced bezel areas display the otherwise hidden information in a lower resolution and prevent misinterpretation in that region. Figure 2.8(d) already shows an improvement in the railway scenario without modifying the brushed aluminum monitor frames of HIPerWall.

2.3.2 The Tiled++ Method

The bezel effect can be reduced by using thin-framed monitors in tiled display setups. The screens of HIPerwall and the 3 × 3 display have bezels of approximately one inch size, so applications have to cope with offsets of about two inches between neighboring monitors. Gaps caused by the monitor rack increase the offset. To make a virtue of necessity, monitor frames are used as display areas for additional computer projectors. The monitors display scenes using the overlay strategy, while at the same time, missing information is provided by the projectors. Projectors only render those parts of a scene that would be hidden by the lattice (see figure 2.10). Since black monitor bezels are barely reflecting and hence not suited for projecting directly onto them, they are covered with diffuse reflecting white cardboard to create a grid-like reflective screen as illustrated in figure 2.2(a). For the 10 × 5 HIPerWall configuration, no modifications to the bezels were required, as the brushed aluminum bezels already provide a good reflective surface for projection. The image parts displayed on the grid provide the missing information in a resolution that depends on the projectors' resolution, e.g., $1,920 \times 1,080$ pixels, as well as their distance to the wall. It is usually lower than that of the monitors (e.g. $2,560 \times 1,600$), so that a scene can be divided into regions of very high and rather low resolution. When using more than one projector, each of them can be restricted to a predefined subset of monitors, thereby increasing the resolution on the bezel grid.

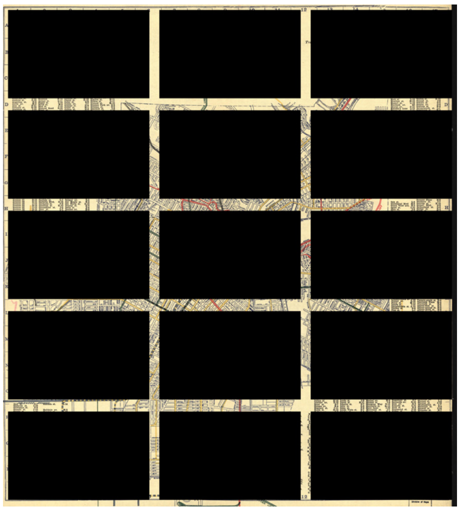


Figure 2.10: Low-resolution (about 10% of the original resolution) image information is projected only onto the bezel grid. The areas of the LCD panels are blacked out. (Sections of Los Angeles Railway Map.)

Methods that are based on combinations of various display technologies with different resolutions have been proposed previously. The focus+context screen proposed by Baudisch et al. [BGS01], for instance, embeds a high-resolution focus monitor into a low-resolution context area that is generated by back-projection. As with Tiled++, the high-resolution parts are masked in the context image in order to prevent interference. The 2D+3D focus+context screen presented by Steffen et al. [SEDD09] is an extension of the original focus+context screen that embeds a 2D focus area into a stereoscopic context. Though these methods use similar techniques, a combination, such as the one described above, has never been used before with the intention of addressing the issues caused by the monitor bezels of a tiled display.

2.3.3 System Calibration

In order to yield results as expected, Tiled++ systems are calibrated in a two step process:

1. The first step involves the determination of the *tiled display geometry*, which basically includes the exact location of each tile in the LCD grid, its dimensions in horizontal and vertical direction, and the bezel size including gaps between neighboring monitors. These parameters are stored in an XML configuration file as described earlier in section 2.5. Referring back to figure 2.5, red indicates the actual LCD panels, whereas everything that is green serves as a display area for the computer projectors and includes monitor bezels and possible gaps.
2. The second step involves the *calibration of computer projectors*. They need to be calibrated so that they seamlessly align with each other and only cover the bezel grid.

AnyScreen is highly configurable and able to drive display configurations of arbitrary complexity, including the rather complicated Tiled++ setup. The calibration parameters can be determined semi-automatically as explained in section 2.3.3.2, which currently yields the best results regarding the precision of alignment, or fully automated with a camera-enabled calibration tool, whose function is described in section 2.3.3.3.

With the parameters, the library is able to assign cut-outs of the final image to the individual tiles of the display and determine which parts have to be left out because they are hidden under the bezels. Furthermore, the library can tell which cut-outs are part of a projector image. Since projectors only provide low-resolution information, they are marked as LowRes displays (see section 2.2.1.1). The library prevents them from directly projecting onto the high-resolution areas of the LCD tiles and spares these regions by masking them in the projector image, as shown in figure 2.10. The lattice-like projector image prevents interference that would naturally arise when projecting onto the LC panels. The overall quality of a scene can be improved by using projectors with good black light properties, which are not always given, especially not with LC-based projectors.

However, projectors cannot simply display unmodified stenciled overlay images without taking the position relative to the display wall into account. Since projectors can be placed at arbitrary positions in front of the tiled display, *keystone effects* can arise that cause trapezoidal distortions in the final images. This effect must be con-

sidered in order to achieve good calibration results. Modern projectors come with built-in corrections that distort images so that the keystone effect is compensated for. Unfortunately, these corrections are too coarse for Tiled++.
The following section describes how a calibration based on homographies compensates for keystone effects by pre-distorting images, so that their projection looks correct on the screen.

2.3.3.1 Keystone Correction using Homographies

Following description is partly based on the excellent introduction by Nielsen [Nie05]. A *homography* H is a linear mapping in the projective space $\mathbb{P}$ that preserves collinearities of points and concurrencies of lines. Considering the projective plane $\mathbb{P}^2$, a homography maps points of one quadrilateral to corresponding points of another quadrilateral. In order to eliminate keystone effects, a homography must be computed that compensates the geometric distortions due to not placing the projector perpendicularly to the screen. H is uniquely defined by four two-dimensional point pair correspondences $\{L_i \leftrightarrow R_i\}_i$ in general position, i.e., no three points are collinear, which is a reasonable assumption for keystone correction. Assuming L_i and R_i are represented by two-dimensional homogeneous vectors l_i and r_i, one can write the following system of equations

$$r_i = \begin{bmatrix} x'_i \\ y'_i \\ w'_i \end{bmatrix} = Hl_i = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}, \qquad (2.1)$$

where

$$\overline{l_i} = \begin{bmatrix} \frac{x_i}{w_i}, \frac{y_i}{w_i} \end{bmatrix}^T \text{ and } \overline{r_i} = \begin{bmatrix} \frac{x'_i}{w'_i}, \frac{y'_i}{w'_i} \end{bmatrix}^T \qquad (2.2)$$

are the corresponding (dehomogenized) Euclidean points. H has nine degrees of freedom, but since it is invariant up to a scaling factor, one can assume $h_{33} \neq 0$ and fix $h_{33} = 1$ (this holds unless $[0,0,1]^T$ is mapped to infinity). This reduces the degrees of freedom to a total of eight. With $h_{33} = 1$,

$$x'_i = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + 1} \qquad (2.3)$$

and

$$y'_i = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + 1} \tag{2.4}$$

holds. Transforming equations 2.3 and 2.4 into

$$x'_i = h_{11}x_i + h_{12}y_i + h_{13} - x'_i(h_{31}x_i + h_{32}y_i) \tag{2.5}$$

and

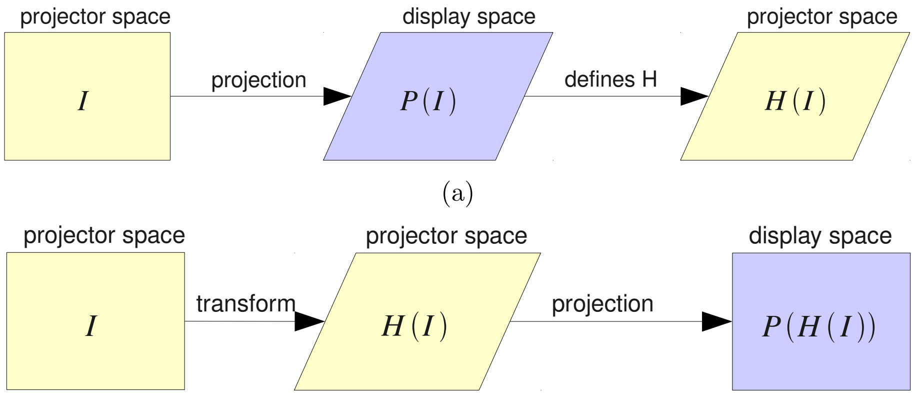
$$y'_i = h_{21}x_i + h_{22}y_i + h_{23} - y'_i(h_{31}x_i + h_{32}y_i) \tag{2.6}$$

yields

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}}_{h} = \underbrace{\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}}_{b} . \tag{2.7}$$

Using a method of choice, e.g., Gauss-Jordan elimination, to solve the linear system, $A \times h = b$ yields the eight coefficients $\{h_{ij}\}_{i,j}$ of the homography matrix H.

In order to compute H it is necessary to define the four point pair correspondences $\{L_i \leftrightarrow R_i\}_i$. When calibrating the Tiled++ system semi-automatically as explained in section 2.3.3.2, this is achieved with the help of a small configuration program which generates a calibration cursor that is displayed by the projector that needs to be calibrated. All calculations take place in two-dimensional projector screen space. The projector image has to cover the entire area of the display wall. The configuration program displays a mouse cursor that is used to select the four corners of the display wall. For instance, moving the mouse cursor into the upper left corner of the display wall and pressing the mouse button defines a correspondency between that corner and the corresponding point in projector screen space. This point-and-click selection is repeated for each of the three remaining corners of the display wall.



(a)

(b)

Figure 2.11: Homography-based keystone compensation: (a) selecting corners defines correspondences between the undistorted image and points in projector screen space yielding H, (b) applying H distorts the original image I so that its projection $P(H)$ is free of keystone effects.

When camera-based calibration as described in section 2.3.3.3 is employed, the projector displays a gradually refining binary pattern in order to determine which corner of the display wall corresponds to which point in projector space.
Regardless which method is employed, the calibration procedure finally yields four point pair correspondences that are needed to calculate the coefficients of H. H maps the points of the original undistorted quadrilateral scene to a distorted quadrilateral image in projector screen space. Projecting the distorted scene resulting from applying H to the projector image compensates the keystone effects. Therefore, H is computed and multiplied with OpenGL's projection matrix (see figure 2.11). The scene is then rendered with the new projection matrix.

2.3.3.2 Semi-automated Calibration

A Tiled++ display can be calibrated either semi-automatically or with a camera-enabled fully automated calibration tool. Currently, the semi-automated calibration method yields better results in terms of calibration quality. The first step is to determine the display wall parameters that go into the *AnyScreen* configuration file, i.e., its geometry. These can easily be measured with a measuring tape using an arbitrary unit of length. For each tile, the dimensions of the actual LC area, the size of bezels, and the size of the gaps between adjacent monitors are determined. Measuring rotation parameters is a bit more complicated and requires the use of a perpendicular.

In the second calibration stage, the homography coefficients for the keystone correction are determined. This is done with the calibration tool described previously. It requires users to select corners of the display wall in order to establish point-pair correspondences between the original image and its projection.

Display wall and projector parameters have to be determined only once for a fixed installation since they are stored as entries in an *AnyScreen* configuration file.

2.3.3.3 Camera-enabled Calibration

Both calibration steps are also supported by a small camera-enabled calibration tool, whose implementation is based on the OpenCV [Ope] framework, an open source computer vision library providing many built-in functions that are commonly used in image processing. Since the implementation is still in the early stages of development, only the basic idea is outlined.

The tool uses a Logitech QuickCam Sphere AF camera that supports a resolution of $1,280 \times 960$ pixels and is placed somewhere in the room so that it can observe the display wall. In order to determine the display wall's geometry parameters, a checkerboard pattern of alternating red and green colors is displayed on the LCDs. The colors are swapped, i.e., green LCDs turn red and vice versa. From the difference of both observed images the tool is able to compute the screen contours in camera coordinates. Assuming all monitors run at their native resolution, one can conclude that a contour's pixel ratio corresponds to the ratio of a monitor. Next, homographies are computed that map each contour into the plane of the display wall. Applying these homographies to all tiles enables the software to determine the necessary configuration parameters.

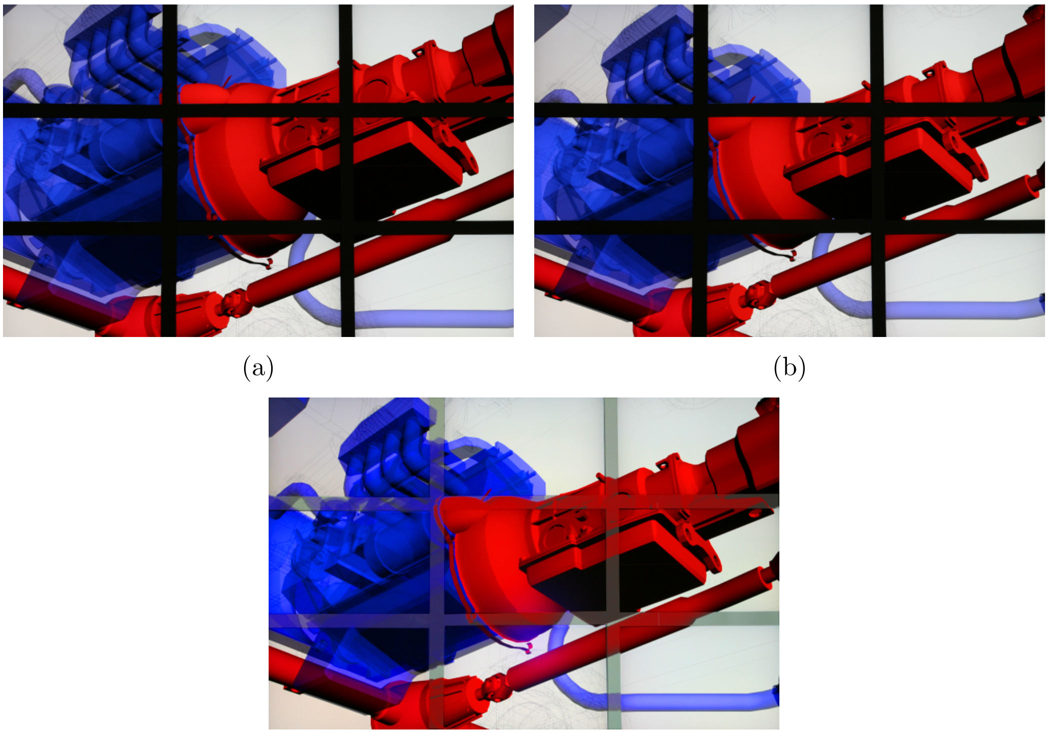
In order to compute a homography for keystone correction, a mapping between camera space and projector space has to be found. The method applied for this task is based on structured light and resembles the technique described by Lee et al. [LDMA$^+$04], except that no photo sensors are used for Tiled++. The projector generates a gradually refining binary pattern and the camera observes whether a corner of the wall is illuminated by that pattern or not (corners are bonded with white cardboard for that purpose). In the end, the calibration tool has information about the tiled display corner coordinates in projector space, which yields the set of corresponding pairs needed to calculate the keystone homography.

2.3.4 Protoypes

Tiled++ has been set up on two different-sized tiled displays. The first setup is the 3×3 display wall described in section 2.1.2.1. As shown in figure 2.2(a), its black frames have been covered with diffuse reflecting white cardboard in order to provide a good projection surface. Bezels are projected on by a $1,920 \times 1,080$ pixel NEC HD projector, whose driving software thread represents the master process of the application (see figure 2.4).
To prove Tiled++ scales to systems of arbitrary size, the second prototype has been set up on the 200 megapixel HIPerWall display cluster, consisting of fifty LCD tiles arranged in a 10×5 grid. In order to project onto the bezel area of the entire display, multiple projectors have to be locally aligned to subareas of the display cluster, using the calibration methods described previously. The brushed aluminum monitor frames of HIPerWall already provide a better projection area than the black bezels of the 3×3 display, but could also be covered with white cardboard to improve its reflective qualities.

2.3.5 Examples

The benefits of Tiled++ were investigated in different scenarios that are presented in this section:
When working on construction plans, it is important to display machine parts and other components without distortions. Figure 2.12 illustrates the outcome obtained when visualizing an engine block using the different bezel strategies. When the offset technique is applied (see figure 2.12(a)), the drive shaft of the motor appears discontinuous and broken to the middle. This is especially obvious when looking at the tile on the bottom right. Further artifacts can easily be made out at the other



(a) (b)

(c)

Figure 2.12: A CAD example: (a) strong discontinuity effects with the offset strategy disturb perception, (b) with overlay strategy, image information is hidden under the bezels, (c) Tiled++ provides missing information by bezel projection.

monitor borders. Apparently the offset technique is not a good choice for visualizing proportion-sensitive data, such as construction plans.

Figure 2.12(b) shows that some of the artifacts can be eliminated by the overlay method. Offsets are not visible anymore, but black bars traverse the scene and disturb perception.

Tiled++ outperforms both previous methods in terms of image quality, as depicted in figure 2.12(c). Although bezels can still be made out by color and resolution, the information provided on the lattice and the absence of severe optical artifacts make the projection-based method superior to the other two.

Another scenario is illustrated in figure 2.13. It shows a situation in which the effects of the overlay technique cause a problem that is more serious than distortions of an objects' actual geometry and its proportions. While users can sometimes extrapolate

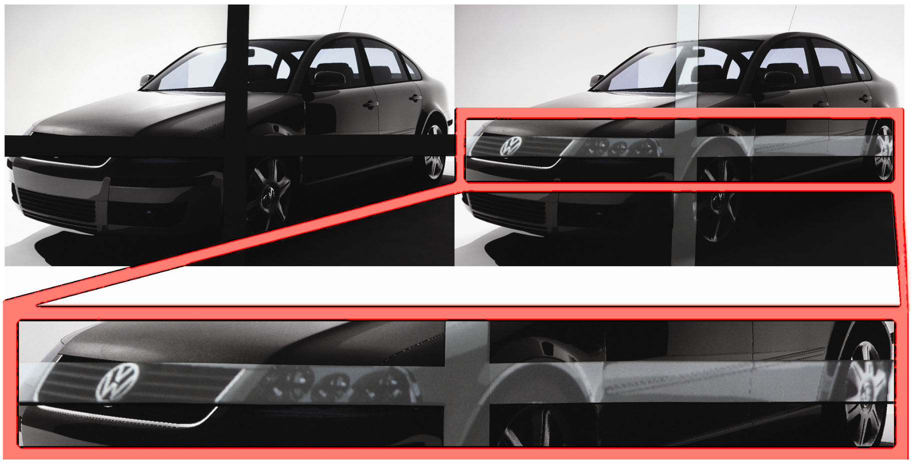


Figure 2.13: Tiled++ provides "higher order" image information, such as, in this case, a company logo.

missing information from incomplete or distorted images, based on experience and assumptions about the geometry, the hiding of "higher order" information is often more critical. In the picture, the front area of a car coincides with a bezel bar of the display wall. Radiator as well as headlights are hidden. Also, the manufacturer's logo identifying the car is missing. As a consequence, users that are unfamiliar with differences between car models find it hard to determine the manufacturer. This is especially true, since the shapes of many of todays cars resemble each other. Again, Tiled++ provides a way to get around this shortcoming. The information provided on the lattice, even when displayed at a lower resolution, clearly identifies the brand of the car.

Both scenarios reveal that without corrections, images can be ambiguous or impossible to interpret.

2.3.6 Interacting with Tiled++

Using computer projectors to enhance the bezel areas of monitor-based tiled walls seems to imply constraints regarding the possibilities of users to interact with the system at a first glance. This section explains how users navigate in front of a Tiled++ setup and why interaction is not affected.

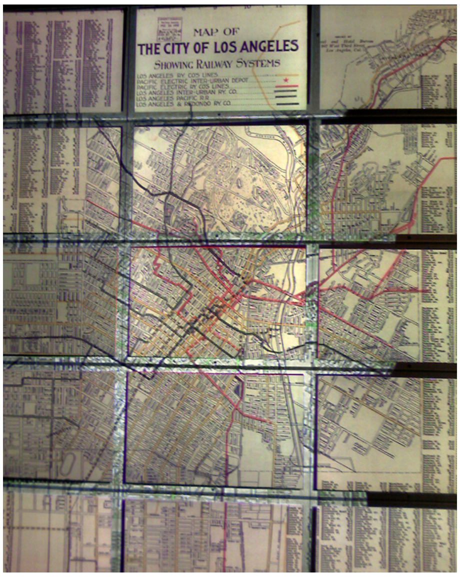


Figure 2.14: A HIPerWall user enters the projection area and casts a shadow on the Tiled++ projection, because the projector is mounted too low.

Multi-projector systems often use back projection as a way to prevent shadowing when users move in front of the display. Being able to interact directly in front of the screen is important, since a primary application field of large displays is to support collaborative work between multiple users. For obvious reasons, back projection is not an option for Tiled++, and the problem arises that users entering the projection area cast shadows on the grid, as shown in figure 2.14. Note that only the bezel area is affected.

The projector calibration described previously is robust and allows inclined positions of up to 10° between projectors and displays, where 90° refers to a straight-on projection and 0° to a projector position parallel to the wall. Placing projectors collateral to the screen allows users to move in front of the displays at a reasonable distance without shadowing the bezel grid. A collateral setup facilitates a natural habit observed when users interact in front of a large display. Users, who want to get an overview, step back in order to gain a larger field of view, whereas a detailed view requires them to step forward and come closer to the screen. This behavior is a human focus+context technique refered to as *human zooming*. Human zooming integrates into Tiled++, as low-resolution regions on the grid are essential to perceive

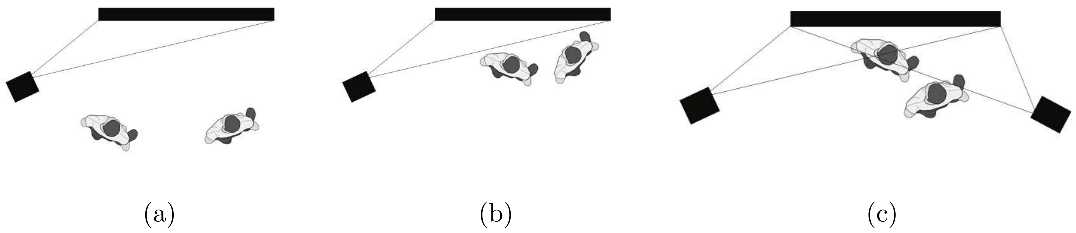


Figure 2.15: Human zooming and the effects of physical movement on Tiled++: (a) context information is available by stepping back, (b) focus information is available by stepping forward, (c) shadow removal using multiple projectors.

a seamless image from a "context distance" (see figure 2.15(a)). When stepping forward, the high-resolution LCD areas allow users to investigate details of a scene from a close distance (see figure 2.15(b)). At the same time, while stepping forward, the information on the grid becomes less important, since its resolution is lower. Shadowing therefore has a less severe effect when stepping closer to the screen. The general effect of physical navigation in the context of large scale visualization was investigated in a user study conducted by Ball et al. [BN08].

Another way to minimize shadow effects is to overlay the projector image with a second projection from an opposite direction as shown in figure 2.15(c). Because the lattice is illuminated twice, users can touch the bezel areas and still perceive information. However, ghosting from dual projector images may occur in this setting, if the calibration is not done carefully.

2.3.7 Discussion of Results

Semi-automatic calibration adjusts projector images with an accuracy of approximately 5mm. For the 3 × 3 tiled display this means that on an area of 1.94m × 1.21m covered by the display wall, projector images deviate about 5mm in each dimension from the pixels of the LCDs. This difference can be noticed but is still tolerable. Since the projector resolution is significantly lower than the resolution of the LC panels, the error is usually less than one to three projector pixels.

The deviation for results achieved with the camera-enabled calibration tool is between five and eight projector pixels. In the future, quality could be increased by using cameras with higher resolution, i.e., higher than $1,280 \times 960$ pixels. The camera used for the calibration has a resolution which is far from sampling rate required by Nyquist's theorem. Another reason that explains the misalignment is the fact that monitor bezels are not exactly in the same plane as the LC panels. The offset between the bezel plane and the LC plane affects the alignment of projector images and tile images. Ball et al. [BVC$^+$05] removed all monitor frames for their study on map navigation on tiled displays, which is not an option for Tiled++. Using non-homography-based registration techniques might further add accuracy to the alignment process in the future. However, for the sake of simplicity, Tiled++ prototypes do not take these alternatives into account yet. The errors encountered are tolerable and precision is acceptable.

There is a significant difference between the maximum resolution of the LCDs and the projectors. This circumstance is inherently connected to the hardware technology. However, the user study revealed the benefits of having low resolution information in regions that would normally be black outweighs a slight discontinuity within the image. As a matter of fact, neither of the 20 participants complained about sharp transitions between monitors and projectors.

2.3.8 Evaluation

The examples given previously show Tiled++ prevents loss of semantics and improves the visualization quality on multi-monitor systems. This section seeks to find out how users perform in certain tasks when using the enhanced display. A user study was conducted in order to quantify the effects of Tiled++. The *main hypothesis* was that the additional information provided by the enhanced display and the lack of distortions have a positive effect on the performance of users. Participants were expected to perform faster and more precisely.

The study was designed to investigate how users perform when being faced with different bezel handling strategies, i.e., overlay, offset, and Tiled++. A major challenge in the design of the evaluation was to pick experiments that are suitable for a wide group of potential users and do not focus on domain specific tasks only suited for a small group of experts. The experiments were kept simple and incorporate only basic geometric objects. This allowed participants to focus on the actual tasks and accomplish them without extensive initial training. Moreover, simplicity ensured that all measured effects were due to manipulations of the *independent variable* and not to factors that are hard to control and can easily be overlooked. A 3D shooter game, for instance, would be a bad choice for an experiment, since the setup is very complex and user performance is influenced by many additional factors, e.g., familiarity with the input devices or previous experience with computer games. The effects showing up in the investigated basic experimental setup equally affect more complex tasks, as they require the same set of skills these basic tests are based on, e.g., testing path tracing- and reaction capabilities. In more complex tasks, users also have to deal with discontinuities due to the offset technique, gaps caused by the overlay strategy, and various characteristics specific to Tiled++. Altogether, the experiments of the evaluation provide an insight on what to expect when users deal with tasks of higher complexity.

Similar strategies of designing experiments have been pursued earlier. Ball et al. [BN05c], for instance, focused on simplest geometric objects as targets for a comparison task in their study on the effects of large displays on basic visualization and navigation tasks.

Navigation tasks in the evaluation measure the performance of users in *dynamic* and *static* settings. The performance measured in constantly changing scenes, such as animations, can potentially differ from the performance observed in static scenes, such as maps, where objects remain fixed during the entire task.

For the static navigation experiment, a variant of the commonly known game *HotWire* was chosen. HotWire is a game of skill in which users have to move a mouse cursor along predefined tracks without leaving an area of tolerance. The game was considered to be suitable for the study because it requires participants to navigate across monitor frames and explicitly deal with the advantages and disadvantages of different bezel handling strategies in these regions.

A single player implementation of *Pong* was chosen for the dynamic navigation task. In this game, users navigate a racket on the screen in order to return a ball that is bouncing of a wall. The game was perceived as easy to handle. It was chosen because it includes diagonally moving objects, which potentially can lead to misinterpretations and misjudgments as discussed in section 2.3.1.

While navigation tasks provide objective and measurable data, *perception experiments* capture more subjective aspects when dealing with different bezel strategies. Consequently, two perception experiments were conducted. First, users were asked to rate the difficulty of assigning collinear lines in a *Poggendorff* setup. Then they were asked to summarize their impressions after the presentation of a short *animated scene* with each of the three strategies.

In the *questionnaire* presented at the end of the evaluation, participants had to rate their own performance and satisfaction with each of the techniques. The intention was to find out whether there exists a discrepancy between the impartially measured performance and the subject's subjective rating. Participants had to take a particular test once for each bezel handling strategy, resulting in a total of three tests per experiment. In order to prevent learning effects and symptoms of fatigue, the order of strategies in the experiments was permanently changed. For similar reasons, the sequence of level tracks in HotWire was constantly altered.

At the beginning of the evaluation, a mock-up explaining the general idea of each experiment was presented to participants. Before starting a test, users received a verbal introduction and practical training for the application on a single monitor system. The evaluation was conducted on the 3 × 3 tiled display system using a single projector to enhance the monitor frames and took between 30 and 40 minutes per participant.

2.3.8.1 HotWire

This game follows the identically named game of skill in which a metal loop has to be directed along a curved wire without touching it. If the loop touches the wire an electric circuit closes and a ringing bell indicates an error. The tiled wall implementation of the game works as follows: Players have to direct a mouse cursor along a line strip without leaving a predefined area of tolerance around the track. The goal is to complete the track as fast as possible with as few errors as possible. Line strips contain intermediate checkpoints that are unlocked as soon as a mouse cursor crosses them, i.e., they turn green. If a cursor leaves the track it is reset to the last unlocked checkpoint, and the player is imposed with a time penalty of two seconds, during which it is not possible to continue.

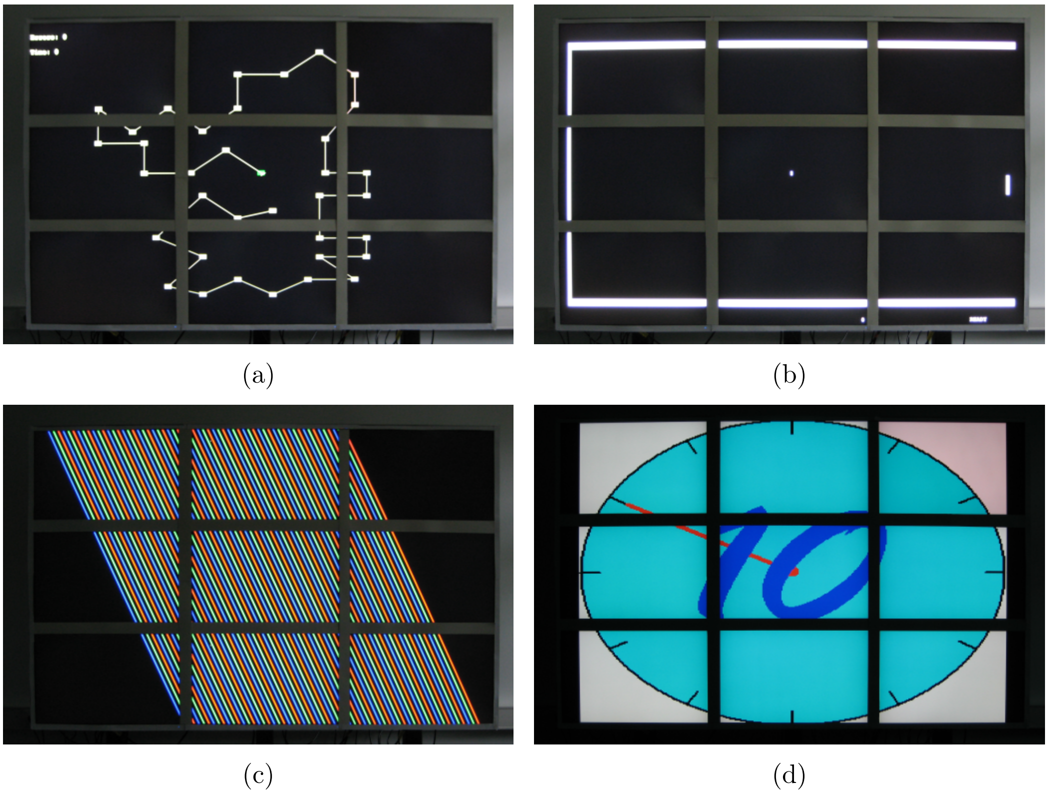

(a) (b) (c) (d)

Figure 2.16: The navigation and perception tasks of the evaluation: a) HotWire, b) Pong single-player mode: Squash, c) Poggendorff test, d) animation test.

In the experiment, the independent variable corresponds to the current bezel handling strategy and may take on three values, i.e., offset, overlay, and Tiled++. The dependent variable is the time spent to complete the track. When running a game, the target time, the total amount of errors, and the error locations are recorded. Ideally, a significant amount of errors occurs in the bezel region uncovering issues when trying to navigate across boundaries.

All level tracks have been carefully designed based on the following considerations: Each level involves a different track in order to prevent learning effects. Furthermore, the track order is constantly altered between participants. All tracks have equal length and a similar degree of difficulty to ensure results are comparable. To achieve this, tracks are derived from a *template* by applying length-preserving geometric transformations, such as mirror and rotation operations. The template design follows a set of rules: In order to avoid favoring a particular handedness of participants, its layout is almost circular. Thereby, a balance between the number of vertical and horizontal components is achieved. This is important since horizontal movements are often considered easier than vertical ones. Directional changes of line segments are restricted to predefined angles in order to generate tracks that are regular. Additionally, a maximal segment length was defined in order to prevent quick walk-throughs by uncontrolled fast mouse drags. Figure 2.16(a) illustrates one of the levels tracks.

2.3.8.2 Pong

Pong is based on the classic arcade game and similar to table tennis. A bouncing ball must be kept in the game by returning it with a small paddle (see figure 2.16(b)). A single player variant called Squash has been implemented. The game was chosen because it is very intuitive and can also be played by participants that normally do not play computer games. Furthermore, it is well suited for investigating how users deal with different bezel strategies under dynamic conditions. In contrast to HotWire, Pong is permanently changing. The ball moves rather fast and users must try to estimate its movement in order to return it. This turns out to be quite challenging, especially when the ball rapidly skips at the borders of two monitors or vanishes under a bezel.
In the user study, each participant played ten games per bezel handling strategy. The goal was to return the ball as often as possible. The independent variable was the current bezel strategy. The total number of successful returns was measured as dependent variable.

2.3.8.3 Poggendorff Experiment

The Poggendorff experiment is one of the perception tests in which participants did not perform under temporal or other constraints. They were asked to freely investigate a given scene and decide what the benefits and shortcomings of the selected bezel handling strategies were. The test shows a scene containing multiple diagonal oblique white lines. At the bezels borders, users were asked to identify corresponding lines. This proved to be rather difficult due to the effects of the Poggendorff illusion.

Participants could verify their choice by adding color to the scene, thus identifying corresponding lines by identical color as illustrated in figure 2.16(c). Naturally, the effect does not occur when choosing the offset method. Here, distortions are directly due to the chosen technique. Tiled++ mitigates the effect as all lines appear continuous. Participants were asked to compare the strengths and weaknesses of each method and write them down in the questionnaire.

2.3.8.4 Animation Test

The animation test consists of a video sequence of a clock that was shown to users (see figure 2.16(d)). Afterwards, participants had to rate their satisfaction with the bezel strategies in the questionnaire. They were asked to pay attention to how distracting image distortions with the offset technique were compared to black bars that are inherent to the overlay method.

2.3.8.5 Questionnaire

After the experimental task, participants were required to fill out a questionnaire with 16 questions. Seven of these were directly related to the experiments and followed the pattern ”How would you rate the perceptual quality during test X when applying method Y?”, or ”Do you think strategy X improved your performance in test Y?”. Additional questions asked about general impressions of the utility of Tiled++, and ideas for improvements. Statistical information about the participants was obtained for classification purposes.

2.3.8.6 Population

Twenty volunteers participated in the study, four of them being female. Most participants were recruited from the computer science lab, and therefore the majority stated to have considerable knowledge with computers or at least be familiar with them. Six were undergraduate students, the rest were graduate students. A majority claimed to have used multi-monitor systems never before or only in a few rare cases. Three volunteers stated they regularly use such systems at work. The average

Technique	Mean time (s)	Standard deviation of time (s)
Offset	142,5	56.95
Overlay	263.15	142.40
Tiled++	109.05	33.25
	Mean error rate	**Standard deviation of errors**
Offset	15.95	16.02
Overlay	40.70	23.47
Tiled++	9.45	5.26

Table 2.2: Averages and standard deviations of performance time and error rates for HotWire

age of participants was 29.3 years. The age bracket of the participants was between 21 and 49 years.

2.3.9 Results

The results of the user study are presented in this section. The data obtained from the navigation tasks was statistically analyzed by conducting an *Analysis of Variance* (ANOVA) in order to identify significant correlations between the means of data series. The idea behind ANOVA is described in appendix A.

2.3.9.1 Results - Navigation Tasks

A one-way ANOVA was conducted with the data of the static and dynamic navigation task. Table 2.2 summarizes the average amount of time and the mean error rates for HotWire (static), including standard deviations.

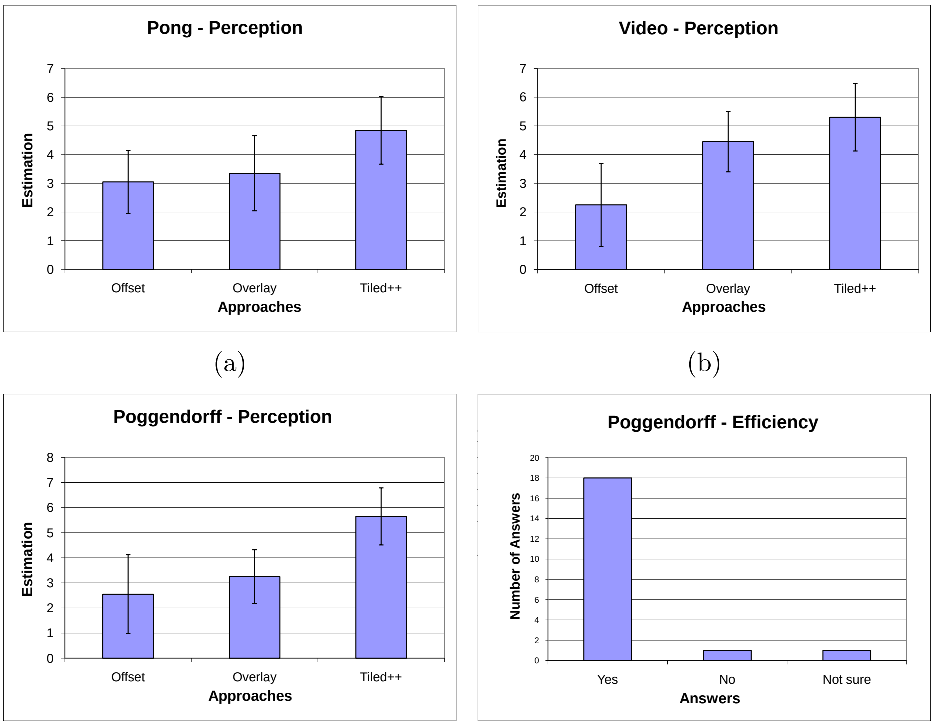
The results reveal a significant effect on how fast participants performed in HotWire for different bezel strategies, i.e., $F_t(2, 57) = 16.01$, $p_t < 0.001$. Tukey's *Honestly Significant Different* (HSD) post-hoc test results further reveal significant pairwise differences among the means of the offset and overlay method and among the overlay method and Tiled++. Differences between the means of the offset technique and Tiled++ were not significant. The ANOVA results of error rates confirm these discoveries, i.e., $F_e(2, 57) = 19.52$, $p_e < 0.001$.

Method	**Mean # of returns**	**Standard deviation**
Offset	10.1	6.99
Overlay	13.55	9.48
Tiled++	12.05	12.24

Table 2.3: Averages and standard deviations of returns for Pong.

These results are not unexpected. It was assumed the overlay strategy would cause the biggest problems when navigating across monitor bezels and that Tiled++ would perform best. Trying to reconstruct the missing information under the bezels appears to be harder than trying to compensate for the discontinuities of the offset technique. Further, it seems natural that with the additional information provided on the bezel grid, Tiled++ outperforms the overlay technique. The recordings of error positions show that with the overlay technique, 89% of errors occurred in the bezel regions of the tiled display. However, it is interesting to find that no significant differences exist between the means of the offset method and Tiled++, since discontinuities were expected to have more severe effects on the navigation. There is strong evidence that the availability of the complete image information is the most important aspect for completing the given task in a reasonable time.

The same statistical tests were applied to the Pong data set. The results reveal no difference among any of the means, i.e., neither of the three methods performed significantly better than the other two (see table 2.3). This is surprising, since it was assumed the discontinuous trajectories caused by the offset technique would yield the worst results. Furthermore, it was anticipated the additional information on the bezel grid would increase the ability of users to estimate the virtual ball trajectory. A possible explanation for this observation is: In dynamic and rapidly changing environments, the intensity of the bezel effect seems to decrease. A reason might be that users perceive the image more as a whole and do not concentrate so much on particular tiles and the transitions between them. Interestingly enough, a majority of participants preferred Tiled++ in terms of overall perception. Figure 2.17(a) depicts an extract from the questionnaire findings. Participants were asked to rate their experience with the different techniques for each task (1 = poor, 6 = excellent). Although the analysis of variance revealed no significant differences among means in terms of efficiency, a majority preferred Tiled++.



(a) (b)

(c) (d)

Figure 2.17: Results of the user study: (a) perception ratings for Pong, (b) perception ratings for the video test, (c) perception ratings for the Poggendorff test, (c) efficiency ratings for the Poggendorff test.

2.3.9.2 Results - Perception Tasks

The Poggendorff illusion had a clear effect on the results of the perception experiment. Participants stated to have moderate to high difficulties in matching collinear lines at the vertical borders of screens. This indicates the effect should be considered when designing applications for LCD-based tiled display walls. Offsets made the task generally harder. Some users figured out rules to systematically identify collinear lines with the offset technique, e.g., "In order to determine the corresponding line, go two steps down and then one step to the right." Thereby, they were able to perform rather quickly, but discovering these rules took some time. Naturally, the task was greatly simplified with the additional information displayed on the bezel grid.

When being asked to rate the perceptual experience and their efficiency, participants answered as illustrated in figure 2.17(c) and figure 2.17(d). Perception with Tiled++ was rated highest with an average of 5.65 (1 = poor, 6 = excellent). The difference between overlay (3.25) and offset technique (2.55) was not significant. The efficiency rating shows that a majority of participants, about 90%, considers Tiled++ to increase performance.

The results obtained from the rating of the video perception are depicted in figure 2.17(b). The techniques that do not distort the image were rated significantly higher than the offset method. Tiled++ and overlay method were on average rated 5.3 and 4.45, respectively, whereas the offset technique was only rated 2.25. A reason for Tiled++ not being significantly better than the other techniques might be the missing color calibration. As stated in section 2.3.3, calibration was restricted to geometric alignment. Therefore, a transition between the bezel grid and LCDs was clearly visible. This interpretation was mostly confirmed by the questionnaire comments.

2.3.9.3 Results - Questionnaire

Participants were asked to provide feedback about what they consider to be the major benefits and drawbacks of Tiled++. The feedback was collected in order to obtain useful information for future improvements. The comments may be summarized as follows: The additional information on the lattice was rated positively across all participants, and it was consistently stated that the monitors were perceived as an almost seamless tiled display. Surprisingly, neither of the participants complained about different levels of resolution for the projector and the LCDs. A majority of 70% stated they would use Tiled++ again, while 30% were not sure. Furthermore, 90% considered Tiled++ to be a benefit for displaying information on tiled walls. 10% were sceptical about the method. Altogether, feedback was generally consistently positive.

A major point of criticism was the difference in brightness and color between the projector and the LC panels. This was especially obvious during the video sequence. In the future, techniques that have already been applied to calibrate the color temperature of multi-projector systems [MHTW00, MS05] could help to minimize this effect.

2.3.10 Discussion

In multi-monitor systems, display frames distort scenes or cover important image parts which leads to potential misinterpretation of information. User performance in static navigation tasks is seriously affected when content is missing due to using the overlay method. Performance in dynamic navigation tasks is less affected. The results of the evaluation foster a discussion about the use of tiled displays in safety-relevant applications, such as flight control or nuclear power plants. In these scenarios misinterpretation of image information can have lethal consequences. However, high resolution and affordable prices make these systems attractive for many application fields.

As demonstrated, it is possible to overcome the drawbacks with minimal effort. Tiled++ is a hybrid technique that combines different display technologies and is able to produce almost seamless renderings on monitor-based tiled displays. A 3×3 and 10×5 setup proved scalability and technical feasibility of the method using standard display components. Thus, Tiled++ contributes to a very small set of methods that explicitly deal with the bezel effect.

The examples show that Tiled++ is not limited to certain application fields and that collaboration among users as well as interaction with the display content is not affected by the projector setup. The feedback obtained from the user study was mainly positive and helped identifying opportunities for future improvement. Especially the difference in color and luminance between the projectors and LCDs was criticized in the perception ratings. In the future, techniques that have been applied for the calibration of multi-projector systems can solve this problem.

2.4 Conclusions

This chapter provided an introduction to large high-resolution displays from a software- and hardware point of view. Various large display configurations have been presented that mainly differ in the type of display devices used for the setup. One option for creating a high-resolution display is to combine the images of multiple computer screens, resulting in monitor-based systems. Another possibility is to use computer projectors to build projector-based displays. Some systems are hybrid displays that combine both display technologies.

When implementing large display applications, distributed rendering software is required to make the individual tiles act as one display. Since applications are usually executed on a cluster of compute nodes, libraries are needed to serve as an abstract layer between the software and the underlying architecture.

Different execution modes for distributed rendering libraries as well as data organization strategies have been discussed. Despite the increasing importance of large high-resolution displays, as of now standard software solutions suited for all tasks and systems are not yet available.

AnyScreen addresses the universal need for a flexible middleware that handles most display configurations. The newly developed library is lightweight and focuses on supporting most common large display types such as arbitrary display-in-display setups. In contrast to many other rendering libraries, it supports a variety of stereo modes used in immersive scenarios and is mostly platform-independent. It has been successfully used for the implementation of powerwall applications combining 2D+3D focus+context views and the implementation of the Tiled++ prototype.

One of the strongest arguments against building monitor-based tiled display walls are screen bezels that distort visualizations and cause undesired, distracting effects, similar to the divisions of a French window. However, monitor-based solutions are an affordable way to quickly set up large displays comprising several hundred megapixels without requiring an extensive calibration. Tiled++ explicitly addresses this shortcoming of conventional tiled displays and minimizes the bezel effect through projector-generated overlay images that are solely displayed on the bezel area of the tiled display.

In this chapter, the extent of the bezel effect on visualizations has been investigated and the setup and calibration of a Tiled++ system has been described. The evaluation revealed that Tiled++, with a high degree of user acceptance, has a positive effect on user performance particularly in static navigation tasks and can indeed help to improve user performance in perception-driven tasks on monitor-based tiled displays.

Chapter 3

Interaction with Large High-Resolution Displays

High-resolution displays are attractive because (1) they provide enough screen space to visualize large data sets at their natural resolution, i.e., without having to apply focus+context techniques, (2) they can visualize multiple data sources in parallel, and (3) they enable users to physically navigate the space in front of the screen and collaborate with each other and the virtual environment. However, with the extended screen space new challenges arise. A major question is how large displays affect the visualization of information, the interaction with the application, and the collaboration of users. Most visualization and interaction techniques have their origins in traditional desktop systems and thus have limitations when used in large display systems. For instance, mouse interaction is effective in single monitor environments, but tracing a mouse cursor on a 200 megapixel display, such as HIPerWall, can be tedious. This chapter is aimed at finding solutions to these problems and focuses on the *interaction-specific* aspects of large high-resolution displays.

Section 3.1 gives a state-of-the-art overview of related work in the field of large display interaction. General challenges of interaction metaphors on large displays are investigated. Where and why do traditional techniques reach their limit, what must be considered when applying new techniques, what exactly is it that lets collaboration benefit from an extended screen area, and what devices are currently used to interact with large high-resolution displays?

Section 3.2 describes an interaction approach that is based on two-dimensional visual tags and incorporates smart phones as universal interaction devices for large displays. Smart phones are becoming increasingly popular and support a variety of different input and output modalities through keyboards, touch screens, acceleration sensors, microphones, Global Positioning System (GPS) receivers, cameras, Bluetooth, infrared and wireless units. Their versatility makes them an interesting alternative to traditional input devices. Tags serve as the interface linking the information on the display to the camera-enabled smart phones of the users. The tag approach helps to address many interaction problems that are hard to tackle with other techniques, such as how to customize visualizations based on the status of a user or how to deal with multiple user groups simultaneously. Two sample applications are presented that implement the tag-based interaction approach on a monitor-based tiled wall.

3.1 State-of-the-Art and Related Work

This section summarizes state-of-the-art results obtained from research in the field of large high-resolution display interaction. The results provide useful insights and guidelines for the design of future systems. In particular, they help to design new effective interaction techniques for large high-resolution displays (see top 7, section 2.1.4), find empirical evidence for the benefits of large high-resolution displays (see top 9, section 2.1.4), design and evaluate groupware for large high-resolution displays (see top 6, section 2.1.4), and identify perceptually valid ways for presenting information on large high-resolution displays (see top 8, section 2.1.4).

3.1.1 Human Performance

There is strong evidence that large displays increase the performance of users in certain tasks and affect their interaction with applications. However, it is not quite obvious what causes the improvement. Neither is it self-evident how the quality of work is affected by an extended screen area, nor what the quantitative effects are and what exactly causes them. Several observational studies and controlled experiments have been conducted in order to find answers to these questions.

3.1.1.1 Qualitative Aspects

Ball et al. [BN05a] present the results of an observational study in which they observed the behavior and actions of five people using a 3 × 3 monitor-based tiled display over a period of six months as a substitute for desktop PCs. They further observed 65 people who used the same display in three controlled experiments. They come to the conclusion that large high-resolution displays, and especially monitor-based tiled walls, afford a number of benefits and drawbacks. They summarize the advantages as follows:

1. Improved user performance for task switching or viewing large documents,
2. Increased ability to spatially position applications and shortcuts for quick access and recall,
3. Bezel adaptations for easy separation of tasks,
4. Increased ability to work collaboratively,
5. Increased screen space for awareness for secondary tasks, e.g., email, instant messengers, news, etc., and
6. Enjoyment - interviewed users almost unanimously prefer multiple monitors to one.

The disadvantages are:

1. Adjustment periods (both from one monitor to multiple monitors, and vice versa),
2. More screen space wasted,
3. Unpredicted behaviors with software applications,
4. Navigational issues with losing track of the mouse cursor, input focus, or other highlights, and the fact that
5. Physical size and layout may require more physical strain.

Though these results were obtained purely observational without a statistical evaluation, they give useful hints about how the quality of work is affected by large displays. Further observations revealed that large displays are preferably used in group meetings and similar collaborative work environments. Section 3.1.3 discusses the influence of large high-resolution displays on collaborative settings.

3.1.1.2 Quantitative Aspects

When building large displays, the size and resolution of the screen change the way the display is perceived. Two aspects need to be considered:

1. The acuity of the human eye is limited by the number of receptor cells comprised in the retina. Using screen devices whose dots per inch (dpi) exceed the receptor density of the human eye is unlikely to be beneficial. Ware [War00] states the optimal display should comprise $4,000 \times 4,000$ pixels to best match the resolving power of the human eye. This assumes that the display fills most of the field of view of the observer.
2. If the visual capacity of the human eye limits the dpi of display devices, the only way increase the resolution of a large display is to actually increase its size by adding more tiles. Eventually, one gets to the point where the size of the display requires head movement or physical navigation in order to see all pixels of the screen.

The higher the total of number of pixels the more information can be displayed. More information implies increased mental processing workload. A major question is if the increased mental workload has a negative effect when using large high-resolution displays or if the availability of more information outweighs this circumstance. Various experiments have been conducted in order to evaluate the performance of users when using large high-resolution displays. Czerwinski et al. [CSR+03] carried out a user study to provide evidence about the benefits of large versus small display surfaces for complex, multi-application office work. They report significant benefits in the use of a prototype, larger display, in addition to significant positive user preference and satisfaction with its use over a small display.

Ni et al. [NBC06] investigated the individual and combined effects of display size and resolution on task performance in an Information-Rich Virtual Environment (IRVE). They state that users were most effective at performing IRVE search and comparison tasks when using large high-resolution displays. These observations are supported by the results of several other studies [YHN07, BN05b, SBY$^+$06].
Shupp et al. [SBY$^+$06] further evaluated the effects of viewport curvature on the human performance. They compared user performance time, accuracy, and mental workload on multi-scale geospatial search, route tracing, and comparison tasks. The results show curving large displays improves user performance and leads to less strenuous physical navigation.
Ball et al. [BN08] tried to find the reason why large high-resolution displays improve user performance over standard displays on many large-scale visualization tasks and investigated the effects of peripheral vision and physical navigation. They found the opportunity for increased physical navigation is more critical to improving performance than an increased field of view.

3.1.2 Interfaces and Interaction Techniques

Although large high-resolution displays are becoming increasingly prevalent, most user interfaces and interaction techniques employed in applications have been developed for single monitor desktop systems and do not scale to large displays. The following section summarizes design issues and challenges for interaction concepts on large high-resolution displays and gives examples of current state-of-the-art interaction techniques.

3.1.2.1 Usability Issues and Interaction Challenges

HCI separates interaction metaphors into three categories [KEM07]:

Navigation widgets enable users to browse a data set.

Direct manipulation widgets enable users to change the properties of data objects, usually after having navigated to them.

Human interaction widgets enable users to communicate with each other through the interface.

On desktop systems, the windows, icons, menus, and pointing devices (*wimp*) paradigm provides interaction metaphors covering all three categories. Robertson et al. [RCB$^+$05] conducted a study to identify *wimp*-specific design issues on large displays. On the basis of their research, they identified six groups of usability issues:

1. *Losing the cursor.* On large display surfaces it is hard to keep track of the cursor position, e.g., when acceleration and speed are high, in order to cover the increased distances on the screen, or when the cursor is stationary and needs to be located.
2. *Bezel problems.* The bezel effect has been discussed in section 2.3.1. It affects applications in *wimp* environments in the same way as described before.
3. *Distal information access problems.* Due to the display size, objects may be "out of reach". The *wimp* paradigm does not consider this fact, e.g., when trying to move a file between folders by dragging an icon across multiple monitors.
4. *Window management problems.* With increasing display size one runs into window management problems. For example, should the maximize- or full screen operation of a window refer to the entire display area or just to the area of a tile?
5. *Task management problems.* Large screen surfaces encourage users to execute multiple tasks in parallel. Efficient task management strategies are required to support multitasking on large high-resolution displays.
6. *Configuration problems.* There is hardly any support of applications for large display configurations, e.g., for tiled displays with varying pixel densities.

Applying metaphors designed for desktop systems in large display environments potentially leads to the usability issues described above. Basically, metaphors do not *scale to the size* of the display. However, *scalability with the number of users* is another challenge. Since desktop systems provide rather limited workspaces, most interaction metaphors are designed to support single users. Large high-resolution displays, in contrast, support collaboration and it is often desired that multiple users interact with applications in parallel. This is where many traditional metaphors reach their limit. In the *wimp* paradigm, for instance, mouse navigation is adequate to support single users. However, using multiple mouse cursors, even when displayed in different shapes or colors, simultaneously in collaborative environments quickly leads to confusion.

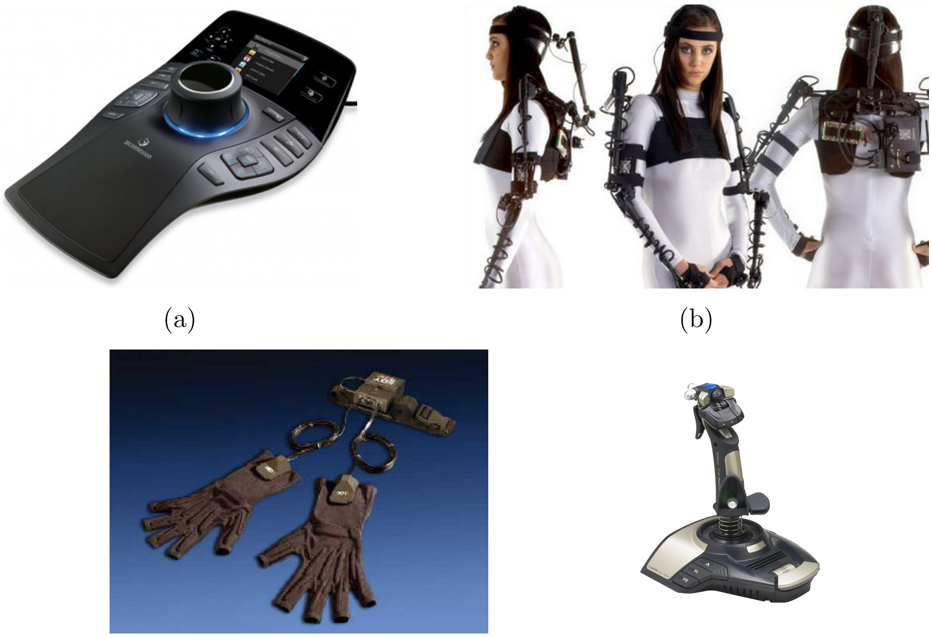
The discussion shows that the design of intuitive and reliable interaction techniques is a challenging task and deserves to be listed among the top 10 research challenges for large high-resolution displays.

3.1.2.2 Interface Examples

Robertson et al. [RCB$^+$05] presented various approaches to overcome the above mentioned interaction challenges in *wimp* environments. For instance, in order to overcome the cursor tracking problem, they propose "Mouse Ether" which eliminates warping effects in multi-monitor systems by applying appropriate transformations to all mouse move events. For distal access problems, a technique called "Drag & Pop" is proposed which is an extension of traditional Drag & Drop. When dragging an icon towards some target icon, Drag & Pop responds by moving potential target icons towards the current cursor location, allowing users to select targets with relatively small hand movements. An alternative metaphor is "Tablecloth". Tablecloth provides access to distant objects by applying the window scrolling technique to the entire desktop. Similar to pulling real tablecloth, users move distant objects towards them by dragging the desktop.

Kerren et al. [KEM07] took a top down approach and identified different classes of input devices to interact with visualization on large displays.

Basic interaction devices encompass input devices that are commonly used for interaction with desktop PCs, i.e., keyboards, computer mice, or joysticks. Extensions to six degrees of freedom have been proposed that provide suitable 3D interaction capabilities, e.g., Spacemouse or Spacepilot [3dc] (see figure 3.1(a)).



(a) (b)

(c) (d)

Figure 3.1: Examples of interaction devices for large high-resolution displays: (a) 3Dconnexion SpacePilot$^{\text{TM}}$ Pro [3dc] *Image courtesy of 3Dconnexion Inc., Tobias Keuthen*, (b) Animazoo Gypsy$^{\text{TM}}$ motion capture suit [ani] *Image courtesy of Animazoo, Paul Collimore*, (c) 5DT DataGlove Ultra Wireless$^{\text{TM}}$ [5DT], (d) Saitek Cyborg evo Force Feedback$^{\text{TM}}$ joystick [sai] *Image courtesy of Saitek Inc.*

While these devices are usually placed on desktops when being used, handheld interfaces, such as the Wanda device [wan], also enable interaction in immersive environments, e.g., CAVEs$^{\text{TM}}$.

Tracking devices are used in combination with immersive display types, such as powerwalls or CAVEs$^{\text{TM}}$. They determine the position and orientation of the entire user or certain body parts in the virtual environment and use this information to facilitate interaction (see figure 3.1(b)). Different tracking technologies have been developed based on mechanical, magnetic, ultrasound, or optical methods.

Gesture interfaces capture and interpret the hand gestures of users to enable interaction. Most devices comprise a set of embedded sensors in a pair of gloves (see figure 3.1(c)). Interaction is very intuitive and in general more accurate than with tracking devices.

Haptic devices provide haptic feedback when interacting with the virtual environment. This is achieved by applying forces, vibrations, and motions to the user through the interface. Lately, haptic devices have become popular in the gaming industry, where force feedback controllers increase the degree of realism in flight simulations or racing games (see figure 3.1(d)).

Audio interfaces work in both directions, i.e., as output devices that allow users to locate objects in virtual environments or as input devices for speech recognition.

Olfactory interfaces and taste interfaces are two groups of rather exotic interaction devices. Olfactory interfaces [YNTT03] synthesize scents from a combination of different aroma that are released into the air. Taste interfaces [IYUM04] work in a similar way and generate flavors by mixing a set of chemical ingredients.

Another study was conducted by Moritz et al. [MWM05]. They compared different input devices and display types for protein visualization. They come to the conclusion that in order to explore three-dimensional protein structures the virtual environment should provide a large field of view at a high resolution in order to show as many molecules as possible. Simple and intuitive pinch gestures and ergonomically designed cordless gaming devices ensure intuitive and unencumbered interaction with protein models.

3.1.3 Collaborative Environments

Collaborative working environments are an important application field for large high-resolution displays. Large displays enhance group interaction because they provide enough space to hold several work areas and visualize multiple data sources (see figure 3.2(a)). At the same time, they encourage users to work in parallel on given problems (see figure 3.2(b)). However, effective interaction techniques are required to facilitate these tasks. Some previously mentioned interaction devices and

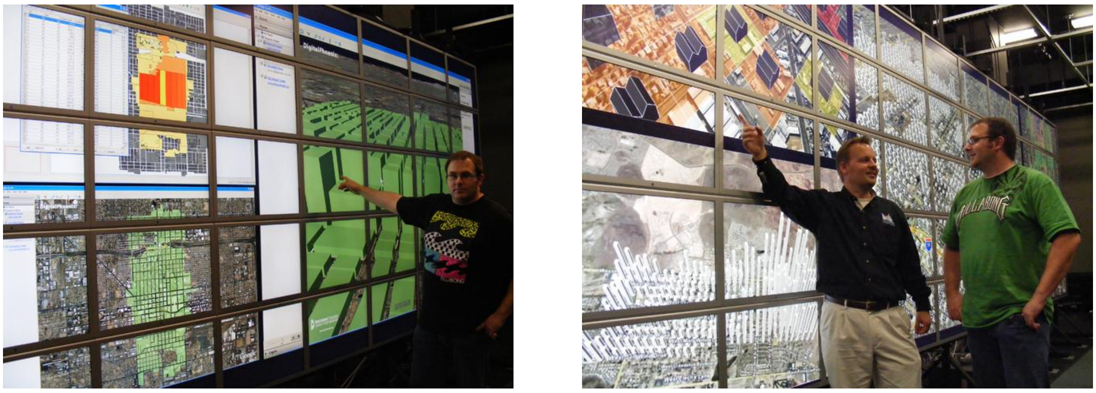

(a) (b)

Figure 3.2: Collaborative working environment: (a) a tiled display holding multiple data sources, (b) interaction of multiple users.

metaphors work fine when a single user is interacting with the application but they do not scale to larger user groups and do not meet the particular requirements of collaborative environments. Systems that have been invented to support collaboration on large high-resolution displays are presented in this section.

Guimbretière et al. [GSW01] present a digital brainstorming tool supporting interaction with high-resolution wall-size displays. Interaction is implemented through pen-based interaction techniques on a $6' \times 3.5'$ projector-based display. The authors adapt techniques that have been developed for electronic whiteboards and support free-hand sketching and working with images and 3D models. A similar approach also featuring pen-based interaction on a projector screen is "Liveboard" [EBG+92]. "i-LAND" [SGH+99] is based on several components occupying an entire room. The system consists of an interactive electronic wall, two computer-enhanced mobile and networked chairs with integrated interactive devices, an interactive table, and two "bridges" for linking virtual information with physical objects.

Huang et al. [HMRS06] give a state-of-the-art survey on large display groupware and formulate design rules for future systems by comparing the benefits and flaws of competing systems. "Notification Collage" [GR01] and "MessyBoard" [FFP02], for example, are two approaches that let users post notes and share media on large displays in shared places and on PC monitors. In both cases, desktop clients are used for interaction and uploading new items to the board. "BlueBoard" and "MER-

Board" [RTD04] both employ touch-sensitive plasma displays to facilitate informal synchronous collaboration. MERBoard is an extension of BlueBoard and has been developed to support science tasks in NASA's Mars Exploration Rover (MER) missions. Both systems provide a whiteboard to support free-hand sketching, a web browser application, and individual user data repositories which can be accessed by swiping ID cards through a Radio Frequency Identifier (RFID) badge reader.

An interesting exploratory study by Birnholtz et al. [BGMB07] investigates the effects of input device configuration on group behavior and performance in collaborative tasks involving large displays. The authors compare the results obtained when using a single, shared mouse with those obtained when using one mouse per user. The study reveals a higher degree of parallel work when using multiple computer mice. Yet, the quality of results is higher in the shared mouse scenario. Moreover, it seems as if users behave more self-centered when using their own computer mouse.

3.1.4 Mobile Interaction

Though originally not designed as interaction devices, smart phones and similar mobile devices have been used as input and output devices for large high-resolution displays. Part of this development is due to the fact that smart phones are becoming more versatile and support interaction through various input and output channels, such as touch displays, GPS sensors, microphones, and built-in cameras. This section gives an overview of latest research results that are relevant to the remainder of this chapter.

Boring et al. [BJB09] investigate how the acceleration sensor can be used to control mouse pointers in two-dimensional navigation tasks and compare three smart phone-based techniques to each other. Vajk et al. [VCBR08] present a similar study also involving the acceleration sensor. They evaluate the usefulness of smart phones as Wii-like controllers for playing video games on public displays. Just like the previous one their study only considers two-dimensional gaming scenarios.

Usually, interaction between mobile devices and large displays is realized via a wireless or Bluetooth connection. Often, visual markers serve as an interface between the display and the input device. Ballagas et al. [BRS05] use visual codes to establish absolute coordinate systems on large display surfaces which are then used to select objects on high-resolution screens via camera-enabled cell phones. Other marker-based interaction methods are presented, for instance, by Jeon et al. [JHKB10], Madhavapeddy et al. [MSSU04], and Rohs [Roh05].

Section 3.2 introduces a tag-based interaction approach with certain similarity to the previously mentioned approaches. However, the idea behind this approach is not to offer smart phone-based solutions for typical *wimp* interaction operations like selection, scrolling, and zooming. Instead, the main focus is on employing location-dependent marker codes that are used to overcome certain collaboration issues arising when multiple user groups work with a large display.

An alternative technique that does not rely on a data connection between cell phones and large display is presented by Shirazi et al. [SWS09]. Their approach uses a camera on the display side to track the flashlight of the smart phone in order to control the mouse pointer position, make selections, and perform scrolling/zooming operations. A comprehensive survey on how smart phones have been used as ubiquitous input devices is given in the paper by Ballagas et al. [BBRS06].

3.2 Tag-based Interaction with Large High-Resolution Displays

An interaction method is presented that addresses the scalability issue by giving each user a separate interaction device, which serves both as an input device by means of a camera, a small keyboard and touch input, as well as an output device capable of displaying rich text or small graphics in addition to the data that is displayed on the large screen [TMM+09].

3.2.1 Interaction using Visual Tags

The approach provides solutions for group-based collaboration and is based on camera-enabled cell phones and two-dimensional visual tags/barcodes that act as links between the large display and the mobile devices. Unlike other marker-based approaches (see section 3.1.4), the focus of the method is not on enabling navigation in *wimp* environments, i.e., selection, scrolling, and zooming.

As discussed previously, collaborative interaction with displayed data is often limited by the number and type of input devices used in most installations. A single input device, such as a mouse or keyboard, is generally shared by multiple users, and often one person takes the lead and controls the visualization for the other users. This means other users are less flexible to explore data sets and potentially less creative due to their dependence on the controlling user. Unconstrained collaboration means every user has their own input device that they can use to interact with the data displayed on the screen, and both the interaction device and the display wall potentially respond to each user's input. Smart phones offer a promising solution due to their versatility.

The idea of the tag approach is to use today's most common electronic device, the cell phone and its built-in camera, in combination with two-dimensional barcodes as a way to interact with large displays. By scanning barcodes, web-services are launched that, in their simplest form, display computed data on the cell phone screen. As it turns out, more complex interaction metaphors are possible with this approach. By associating a profile with each user, the interaction experience can be customized, and different users may use different input methods and may get a different response based on their level of expertise, their role in a decision process, or their personal preference.

Section 3.2.2 and section 3.2.3 deal with the technical aspects of the marker scanning approach while section 3.2.4 explicitly summarizes its utility for large display interaction. So far, the technique has been implemented in two large display applications which are described in section 3.2.5 and section 3.2.6, followed by a discussion in section 3.2.7.

3.2.2 High Capacity Color Barcodes

A variety of two-dimensional barcodes exist (e.g., EZcode, QRCode, Maxicode, and ShotCode) that mainly differ in the way information is encoded and the number of bytes that can be represented. The applications presented in this section use Microsoft's *High Capacity Color Barcodes* (HCCBs) [mst]. HCCBs consist of grids of colored triangles either encoded by a 4-color or 8-color system. Microsoft's implementation for mobile devices uses 4 colors in a 5 × 10 grid to represent 13 bytes

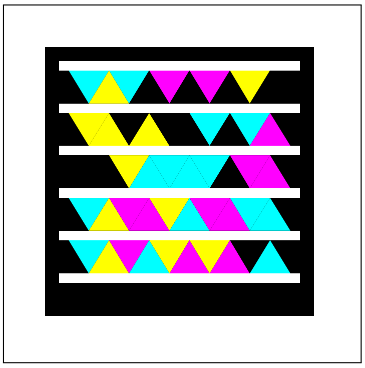


Figure 3.3: High Capacity Color Barcode: triangles of four different colors encode 13 bytes of raw data including error correction in a 5 × 10 grid.

of raw data including Reed–Solomon error correction (see figure 3.3). Each HCCB encodes one of the following items:

1. *Uniform Resource Locators (URLs).* URLs directly link to websites that are loaded in the cell phone's web browser.
2. *Free text passages.* Free text passages contain text segments of practically arbitrary length[1].
3. *vCards.* vCards resemble electronic business cards that can be associated with HCCBs to share business information.
4. *Dialers.* Dialers are used to quickly dial the company or person encoded in the tag.

In any case, the actual data is not stored in the tag itself. Instead, each barcode refers to an entry stored on Microsoft's Tag Server which provides the information sought by clients. The software to identify and decode HCCBs is available for a number of cell phones and free of charge at the time of writing. By using Microsoft's software, many cell phone specific implementation details are avoided. Since no programming work is required on the cell phone side at all, developers are freed from creating multiple software versions for specific mobile phone operating systems or programming environments, e.g., Java, C, or, Objective C.

---

[1] At the time of writing, no maximum length is specified.

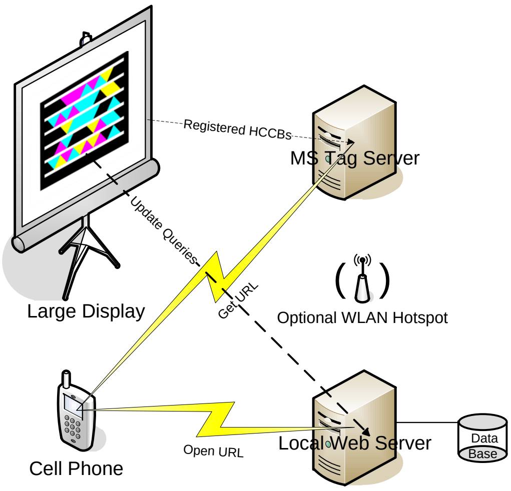


Figure 3.4: Client-server architecture used for applications based on the HCCB approach.

3.2.3 System Architecture

The general system architecture for applications using the HCCB technique is depicted in figure 3.4. Items on the screen are labeled with HCCBs which have been created on the MS Tag Server. The data and type of information they encode have also been specified there. Currently, applications use HCCBs to solely encode URLs. When users scan a barcode, the MS Server transfers the URL that has been registered with the tag to the cell phone. The URL links to a local web server that is able to process client requests and generate HTTP output. The web server runs a set of Common Gateway Interface (CGI) scripts written in Python. The name of the script to execute is specified in the URL including optional input parameters passed in the query string[2]. Calling `http://theserver/add.py?a=2&b=3`, for instance, executes the server script `add.py` and processes two key-value input pairs, i.e., `a=2` and `b=3`. The script computes the sum of both numbers and generates HTTP code that displays the result on the requesting cell phone. Since data processing takes place on the web server side, even complicated operations can be implemented. For instance, it is possible to attach a MySQL data base and query data from that data

---

[2] Query string refers to the part of the URL after the "?". Query strings are used to pass lists of variable and value pairs with "&" as the pair delimiter, and "=" as the variable and value separator.

base for further processing. In essence, the technique incorporates cell phones as thin clients in a typical client-server setup. All mobile devices require access to a stable Internet connection, which can either be provided by the phone company's Global System for Mobile Communications (GSM) or Code Division Multiple Access (CDMA) data network or by a wireless local area network (WLAN). Though no direct link between mobile devices and large displays is inherent in the design, computers driving the displays can query update information from the web server. That way, a bidirectional communication between cell phones and large displays can be established, enabling users to interact with data sets and modify them.

3.2.4 Contribution

The contribution of the technique can be summarized as follows:

Scalability. Using cell phones as universal input devices allows interaction metaphors to scale both in terms of user group and display size. The system depicted in figure 3.4 has no theoretical technical limitations and can easily handle user groups between one and several dozens of members. In fact, the size of a display is of no importance to the approach at all.

User group management and access control. Data visualized on a large screen is visible to everyone. Often however, it is necessary to structure content and restrict access to certain bits of information through user privileges. Many large display approaches ignore user group hierarchies and present the whole data set visible to the entire user group. Thus, it is hard to deal with confidential data (e.g., answers to a quiz) or present information depending on the status of a user (e.g., student or teacher). HCCBs provide a simple solution. When barcodes are scanned, access privileges are verified based on the IP address of the cell phone or by querying a user name and password. That way, information can be organized and visualizations can be customized according to the privileges of a user.

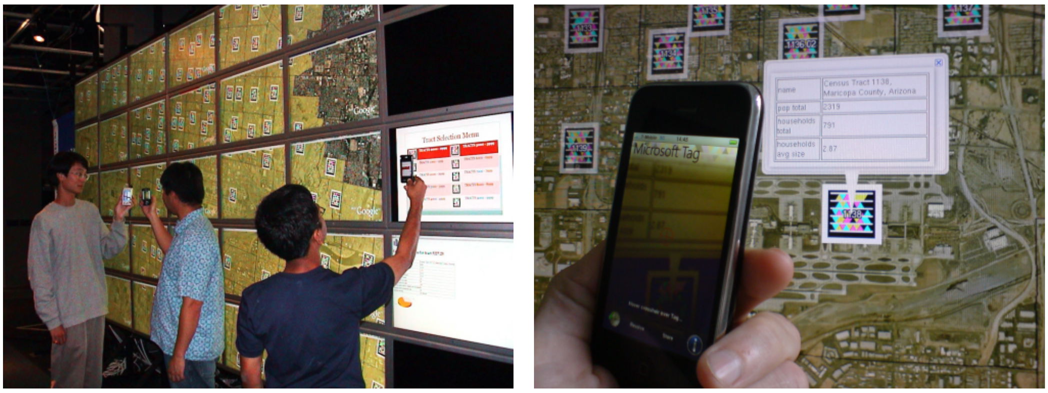
Information overload. A common problem in computer graphics when visualizing large multi-variate data sets is information overload, i.e., the presence of too much information. Information visualization came up with methods to meaningfully reduce the dimensions of data sets (e.g., principal component analysis or self-organizing maps) or to display them trying to avoid visual overload (e.g., parallel coordinates or scatter plot matrices). The tag approach implicitly offers a way to deal with high-dimensional data sets. By tagging data items with barcodes, the entire information of a data object can be retrieved and displayed on the cell phone without overloading the scene on the large display.

Integration. The approach is developed for usage on large displays. The resolution of these screens allows the rendering of HCCBs on a small spatial area in a decent quality. Barcodes can be integrated into applications without dominating scenes and still be recognizable for cell phone cameras.

Versatility. The technique is versatile and widely applicable. Besides a web server to process user requests and generate output, the technological requirements are rather low. HCCBs are created on the Microsoft Tag Server and associated with specific screen locations on top of the displayed data set. As the examples show, this is straightforward for most applications.

3.2.5 Google Earth

One of the large display applications whose interaction component implements the visual tag approach is a distributed version of Google Earth [goo] used for the visualization of Geographic Information System (GIS) data. The application has been implemented on HIPerWall (see section 2.1.2.2), so instead of having a single program instance scaled to the entire size of the wall, twenty-five instances (each instance drives two tiles) that communicate via a gigabit network are executed in parallel. That way, each tile of HIPerWall displays Google Earth at the full resolution of the LC panel, i.e., $2,560 \times 1,600$ pixels. Google Earth is an ideal scenario for evaluating the interaction technique, as GIS data usually contains many data objects that are associated with high-dimensional feature vectors, and the risk of information overload is generally high. The interaction module has been integrated exploiting Google Earth's support for Keyhole Markup Language (KML) files. KML is an XML-based language schema for geographic annotation and visualization which is part of the Google Earth API [kml]. All interaction methods are based on the client-server architecture described in section 3.2.3 and listed in the following:



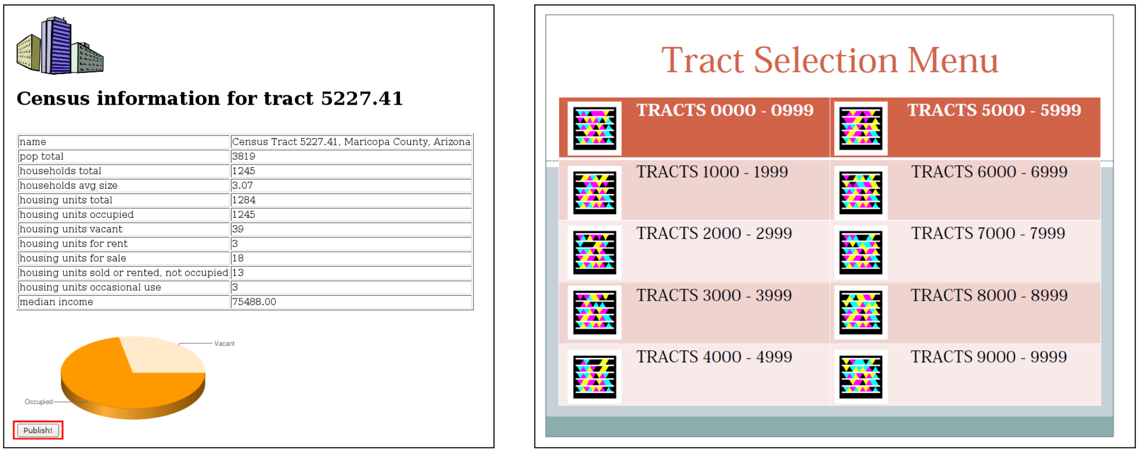
(a) (b)

Figure 3.5: Tag-based interaction with Google Earth on HIPerWall: (a) a group of users is working in front of the census data set. The picture shows the data set as well as the public screen and the ”toggle menu” located on the right (three monitor columns are not captured in this image), (b) a description balloon pops up after scanning a placemark barcode and displays four out of sixty data values.

Visibility. Google Earth supports the visualization of information layers which contain groups of data objects/placemarks. With the help of barcodes it is possible to toggle their visibility by scanning the according tag. The ”toggle tags” are displayed in a separate menu on a designated monitor of the tiled wall (see figure 3.6(b)). On the web server the visibility information is retrieved from a MySQL data base and updated after each HCCB scan. On the display side, the visibility of a layer is updated with the XML `<refreshMode>` tag defined in the KML standard, which constantly refreshes objects that have been loaded via `NetworkLinks`.

Public screen. A public screen is a designated monitor of the tiled wall which displays information that would normally only be visible on a cell phone display. The public screen shows the very same information in a full screen web browser. The idea is to facilitate group discussions by making the object of focus visible to the entire group and is directly aimed at a use on monitor-based tiled displays.

Description balloon. Another Google Earth feature is the so-called description balloon which turns out to be useful in combination with HCCBs. Description



Census information for tract 5227.41

name	Census Tract 5227.41, Maricopa County, Arizona
pop total	3819
households total	1245
households avg size	3.07
housing units total	1284
housing units occupied	1245
housing units vacant	39
housing units for rent	3
housing units for sale	18
housing units sold or rented, not occupied	13
housing units occasional use	3
median income	75488.00

(a) (b)

Figure 3.6: Presentation and interaction with data: (a) detailed census tract information shown on cell phone displays. The red marked publish button at the bottom of the site loads the content into a web browser on the public screen, (b) tags of the tract selection menu are used to toggle the visibility of different census tract groups based on their ID.

balloons pop up like speech bubbles when users select placemarks. They are convenient, since they directly link information about an object with the object itself. By using Google's extension of the KML standard, supported in Google Earth 5.0 and later, it is possible to toggle the visibility of description balloons (`<gx:balloonVisibility>`). Instead of using the mouse as interaction device, placemarks are labeled with barcodes. When scanning the tag associated with a placemark, its description balloon pops up for a certain amount of time and disappears again (see figure 3.5(b)). In contrast to public screens, the information provided by description balloons is always in spatial proximity to the object of interest and usually not as comprehensive. Public screens use the full screen area of an LC panel to show information about data objects. Balloons, however, occupy a smaller fraction of the display area, having a direct influence on how people use both techniques for their work.

User group support. The system is able to handle multiple user groups in parallel and offers ways to customize the presentation of data through information classes. Each cell phone IP is associated with a information class. Depending on the class, the information presented is more or less comprehensive.

In the example, Google Earth is used to visualize census data of Maricopa County, Arizona. Census data sets contain systematically acquired information about the population in a certain area, i.e., age distribution, ethnical background, housing situation etc. Data sets are structured into census tracts, whereas each census tract is associated with a high-dimensional feature vector. The data set used in the scenario consists of 663 tracts and stores 60 values per entry. With that size, it is too large to be displayed on a single monitor system for proper investigation, and the number of dimensions is likely to cause information overload.

Census tracts are visualized as brown colored polygons that are labeled with tract IDs and barcodes in their centers. When scanning a barcode, users decide whether they want a description balloon to pop up or whether they want to display information about a tract on their cell phone. Description balloons consist of tables containing general information about census tracts, i.e., name, total population, total number of households and average household size (see figure 3.5(b)). The data shown on the cell phone is more comprehensive. A detail level for the number of values can be set by indicating an information class number. The higher the number, the more details are loaded from the data base. Figure 3.6(a) illustrates a typical example for content on a cell phone screen that appears after scanning an HCCB. Twelve out of sixty feature values are shown in the table. Additionally, a diagram gives a better overview of the census tract's housing situation. When pressing the "publish" button at the bottom of the site (marked red), its content is transferred to the web browser on the public screen, which is located at the rightmost column of the display wall (see figure 3.5(a)).

The "tract selection menu" depicted in figure 3.6(b) is located right above the public screen. Ten barcodes allow users to toggle the visibility of census tracts according to their ID number group, i.e., 0000-0999,. . . ,9000-9999.

As illustrated in figure 3.5(a), the method enables groups of users to work individually and collaboratively in front the tiled display. While one part of the group is exploring the data set, another user is scanning the "tract selection menu" in order to add further census tracts in the upper right corner of the scene. Because of the display size, the data set is always accessible to the entire group. The possibility for physical navigation as propagated by Ball et al. [BN08] is always given and not limited by technical constraints of the input device, i.e., tracking radius, cables, or number of multiple users.

3.2.6 A 3D Human Brain Atlas

The second application that integrates the visual tag approach is a 3D human brain atlas [TMEH09] implemented on HIPerWall. The main idea of the atlas is to use a large, volumetric model of the human brain, which is computed from high-resolution cross-sectional images, so-called cryosections, and to employ HCCBs which allow various user groups to interact with the large display using their smart phone. 3D representations of the human physiology provide interesting options. Understanding the human brain seems to be easier when the anatomical structure is shown in the three-dimensional domain rather than in a 2D or flat projection. Seeing how the brain is "wired" and how the different regions are connected to form circuits and complex networks requires a spatial understanding of the structure. Conclusions about how this structure evolved can be drawn more easily from a 3D model than from a 2D depiction of the brain.

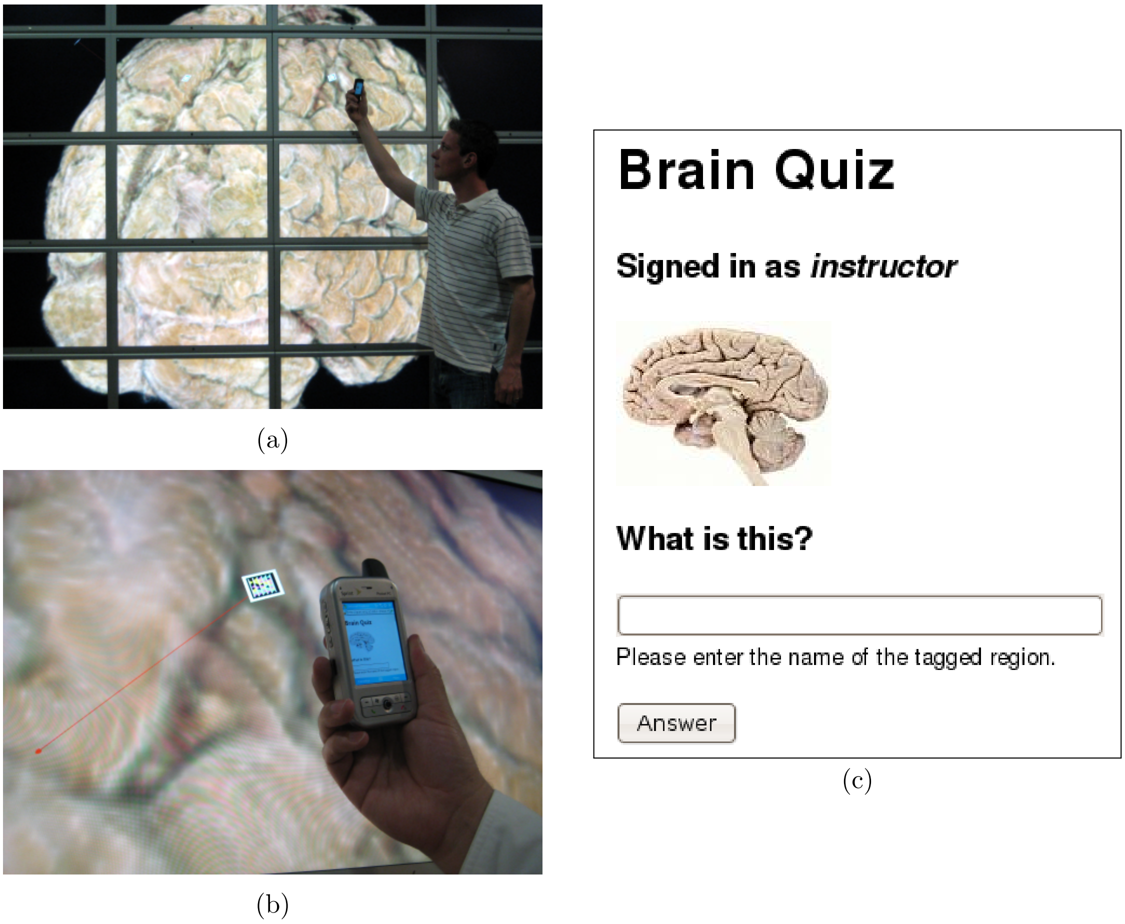
Education is the main purpose of the atlas. The intention is to implement an application that can be used for training of medical students as well as for education of fifth-graders. The atlas is supposed to give an insight in the physiology of the human brain and can be used as a tutor system or as an interactive quiz. With a target audience spanning over multiple age groups and levels of expertise, requirements for the interactive software application are quite diverse. User groups differ in their background knowledge and their preferences, which must be taken into account.

In the past, there have been various successful attempts to create digital brain atlases on single monitor desktop systems, both 2D and 3D. Mikula et al. [MTSJ07] describe the implementation of an internet-enabled virtual brain atlas that provides high-resolution microscopy data of different species. Clients are able to query multi-resolution images of various brain parts from a server storing more than 31 terabytes of data. Besides, various 3D models can be downloaded as VRML files. "The Whole Brain Atlas" [Who] is another internet-based application focusing on the human brain and provides a multitude of 2D views to investigate the effects of brain strokes and degenerative diseases like Alzheimer's or Huntington's disease.

Though not focusing on the human brain, the "Allen Brain Atlas" [All] seeks to combine genomics with neuroanatomy by creating gene expression maps for the mouse brain. The desktop application allows users to explore the expression of approximately 20,000 genes in a fully interactive 3D environment.

3.2.6.1 The Visualization Module

The brain model generated by the atlas is calculated from a 3.57 GB histology data set of a human cadaver brain that has been cut into 753 slices and scanned at a resolution of $1,472 \times 1,152$ pixels per slice. The resolution of the data set is high enough to make even smallest details, such as blood vessels, visible. 3D texture mapping is used as a rendering method for the brain model. The data set size demands for special processing techniques, since the main limitation when it comes to 3D texture-based volume rendering is the available amount of texture memory, which in case of HIPerWall is less than 1/10th of the data set size per render node, i.e., the graphics card of each render node contains 256 MB of video memory. Instead of downsampling the entire brain model to fit into the video memory, the computation is split up among the nodes of the cluster in order to let each node display its part of the scene at the highest possible quality.
In order to overcome the constraints of limited texture memory, a dynamic subdivision scheme incorporating a multi-resolution representation of data is used. In an image-based out-of-core frustum clipping step, an octree helps determining those parts of a scene that have to be visualized by a particular render node. Once the subvolume of each render node has been computed, a wavelet level is calculated, so that the subvolume fits into the video RAM of the render node with highest possible quality. Instead of scaling the entire scene to fit the size of the wall, each of the fifty tiles of HIPerWall displays its part of the data set in an optimum resolution. A detailed description of the visualization module and the employed data structures is given in section 4.2.



(a)

(b)

(c)

Brain Quiz

Signed in as *instructor*

What is this?

Please enter the name of the tagged region.

Figure 3.7: Tag-based interaction with the 3D atlas: (a) user in front of the display, (b) close-up: user prompted to enter data after scanning a tag, (c) input form: instructor can query answers directly by pressing the answer button.

3.2.6.2 Execution Modes

There are two execution modes for the brain atlas that differ in the way people use the application for their work: quiz mode and tutor mode. In quiz mode, candidates test their knowledge of the anatomy of the human brain and receive immediate feedback from the application. In tutor mode, users deepen their knowledge by exploring the data set individually, using their cell phones as interaction devices. Both modes offer different opportunities for tag-based interaction, which are described in the following:

Quiz mode. In quiz mode, users test their knowledge of the physiology of the human brain. Therefore, 23 brain regions that have to be recognized and named by test candidates have been labeled with HCCBs. Red pins indicate the exact region to be named. A tag is attached to the end of each pin and must be scanned by the test candidate as illustrated in figure 3.7(b). After scanning by the user and retrieving the registered URL from the Microsoft Tag Server, an input form appears on the cell phone display asking to name the region (see figure 3.7(c)). The test person's answer is encoded in the URL's query string and transmitted to the web server. A CGI script analyzes the string and keeps track of all current answers. After having named all regions, statistics are presented on the cell phone screen indicating which answers were correct and where errors occurred. Before taking a quiz, users have to register and specify their status by scanning one of two barcodes that are constantly displayed in the lower left corner of the display wall. Users registered as "students" take part in the quiz with standard privileges. Users registered as "instructors" are able to query answers directly without filling out the input form by pressing the "answer button" at the bottom of the form. While everyone is allowed to take the quiz with student privileges, only selected users can register as instructors. The script retrieves the IP address of the HTTP request and searches for it in the MySQL data base on the web server. If the IP is found in the table of registered instructor IPs, the correct answer is transmitted upon request.

Tutor mode. In tutor mode, users may study human brain physiology and learn about the functionality of different brain regions. The mode is versatile and supports a variety of submodes, each aimed at the requirements of a particular user group. Just like in quiz mode, regions are labeled with barcodes that are then scanned using camera-equipped cell phones. Here however, instead of asking for the name of a region, users are provided with relevant information about a selected brain region. The information provided to the user depends on their background knowledge and their specific profile. The following user groups are supported:

1. *K-5.* This elementary grade mode provides children with an opportunity to discover the human brain at a basic level. The information provided is mostly free from medical jargon. This version of the atlas teaches, for instance, about the parts of the brain which are in charge of processing aural, haptic and

visual input, and which regions are mainly responsible for long- and short-term information memorization.

2. *Undergraduate.* For the study of the human brain anatomy and various taxonomies, the respective regions in the data set have been labeled and barcodes link to encyclopedic websites which contain detailed information and references to medical handbooks, research sites, and medical studies. The information is displayed in a summary format on the handheld device and can be transferred to a public screen for more detail.

3. *Medical.* Pathological information is provided to members of this user group. Medical students need to learn which parts of the brain can be affected by diseases and how to diagnose them correctly. This is accomplished by using the same label set as before and linking the barcodes to medical reports, case studies, diagnostic, and treatment methods. The information is displayed in a summary format on the handheld device and can be transferred to a public screen for more detail.

4. *Research.* Scientists often need an overview of the current literature about a particular topic. Using the same label set as before, the server script performs a literature search in a public data base and displays the results on the cell phone screen. If desired, the results can be transferred to a large, public screen for further inspection and research.

The numbers and locations of tags may vary between user groups. For K-5, for instance, a simplified set of 11 barcodes is rendered indicating the most important functional regions of the human brain. The number of regions that may be relevant to a medical student, practitioner or researcher, however, is much higher and would most likely overwhelm a younger audience. Therefore, the 3D atlas offers a feature which allows the user to select a particular group membership. A "user group selection menu", similar to the "tract selection menu" of Google Earth (see section 3.2.5), is displayed on a designated screen on the right side of the tiled wall, containing HCCBs that trigger a reload of the set of barcodes for the brain model. Besides a designated screen for the "user group selection menu", the 3D atlas also features a public screen, which enables discussions and is capable of displaying documents with large amounts of text, images, and graphics. Since cell phone displays are too small for larger documents and difficult for more than one user to see, the public screen shows the very same information in a full screen web browser. Users decide whether they want to make the information on their cell phone public. In order to transfer content to the public screen, they have to press a "publish button" that is always displayed at the bottom of the cell phone screen. The idea is to spur group discussions by making the object of focus visible to the entire group and to let users display additional information.

3.2.7 Discussion

The interaction method provides useful solutions to the issues addressed in previous sections. For instance, users are freed from tethered input or tracking devices by providing them with a wireless, universal input and output device using two-way communication, allowing for physical navigation and interaction with a large display and collaboration with other users holding a similar device (see figure 3.7(a)). In fact, the system could support many more users, as the screen space extends beyond the dimensions of a single room. Currently, user-to-user communication, besides the human aspect of working in the same space in front of the display, is established by means of a public screen, which serves as a common data display which can be viewed by multiple users. It would also be possible to send data directly to another user's handheld device, which can be implemented in future versions.

Scanning barcodes on highly reflective LCD surfaces was not an issue. The cell phones that were used easily identified tags at a distance of 1-3 feet, depending on the barcode size. However, some differences between various cell phone types were encountered which were due to the quality of the camera lens and the image capture resolution supported by the device.

A minor problem is the fact that barcodes cannot be scanned when they are interrupted by the bezels of the LC panels. The current strategy is to adjust the view to make sure the barcodes are displayed continuously. However, Tiled++ (see section 2.3) could help to overcome this issue.

3.3 Conclusions

The focus of this chapter was on interaction with large high-resolution displays. The size and resolution of screens have an influence on how users perceive the display, interact with each other, and the content that is displayed. The state-of-the-art section provided a comprehensive overview of research results in that field. Studies have shown that large high-resolution displays have a positive effect on user performance in complex tasks that deal with large data sets. Part of the positive effect is due to the opportunity for physical navigation in front of the screen which seems to go along with an improved user recognition memory. Cooperation in group meetings and other collaborative scenarios is improved and benefits, since multiple users have access to the same data. However, appropriate interaction methods are required to support users working with these systems.

A strong focus of this chapter was on mobile interaction using smart phones as ubiquitous input devices for large high-resolution displays. Smart phones free users from tethered input or tracking devices and can be used for multi-modal interaction by means of touch displays, acceleration sensors, built-in cameras, etc. The tag-based interaction approach aimed at facilitating collaboration of user groups in front of large high-resolution displays. Its main contribution is to provide ways to customize the large display experience based on user profiles, so that different users may use different input methods and get different responses.

Summarizing, it has been observed that the design of new interaction techniques for large high-resolution displays is a challenging task and is therefore still an active field of research. Requirements for large display interaction are very different from the ones that apply when working with traditional single monitor systems. Most interaction techniques and paradigms that have been applied to desktop systems for decades turn out to be less useful or not useful at all when applied to large displays and immersive scenarios. Scalability is a major issue as most methods have been designed for smaller screen surfaces and smaller groups of users. Researchers have identified and published certain groups of usability issues for large display interaction and formulated guidelines for the design of new interaction techniques. The methods proposed in this chapter were designed to provide solutions to many of these issues, including multi-user interaction, access control, and scalability.

Chapter 4

Volume Visualization on Large High-Resolution Displays

This chapter deals with the visualization of scalar data on large high-resolution displays using direct volume rendering methods. Large displays are predestined for visualizing high-resolution data that requires an extended screen area to be perceived in its entirety. Therefore, the two major questions that arise and are subject to this chapter are:

1. How can data sets be effectively handled considering the resource limitations of the rendering cluster?
2. How can data sets be visualized at interactive frame rates?

Before proposing possible solutions to these questions, section 4.1 gives an overview of state-of-the-art techniques for direct volume rendering and provides a summary of related work that is relevant to this chapter.
Section 4.2 describes and out-of-core technique making efficient use of the resources available on modern graphics cards which mainly limit the amount of data that can be visualized with GPU methods. The system employs a dynamic subdivision scheme incorporating multi-resolution wavelet representation to visualize data sets with several gigabytes of voxel data on distributed rendering clusters using GPU texture slicing.

Section 4.3 investigates methods for direct volume rendering via ray casting that operate at interactive frame rates by exploiting acceleration structures based on hierarchies of bounding volumes.

4.1 State-of-the-Art and Related Work

Volume rendering is an important discipline in computer graphics employed in scientific and medical visualization of scalar field data. Lately, is has become increasingly popular for enhancing animated scenes in video games and generating special effects in movies. Direct volume rendering is based on a physical model of the propagation of light. Due to its high computational costs, volume rendering has always sought at exploiting the features of latest generation hardware for implementing fast visualization systems. This section introduces the basics of direct volume rendering, discusses the CPU and GPU as implementation platforms for state-of-the-art visualization algorithms, and gives a summary of multi-resolution techniques and data structures used in subsequent sections for accelerating the render process and for dealing with large-scale data sets on distributed display clusters.
Besides the original papers, the reader is referred to Weiskopf et al. [Wei06] and Hadwiger et al. [HKRs$^+$06] for an excellent introduction to the topic.

4.1.1 Volume Data

A scalar volume can be interpreted as a continuous three-dimensional signal [RS04]

$$f(\vec{x}) \in \mathbb{R} \text{ with } \vec{x} \in \mathbb{R}^3. \tag{4.1}$$

Discrete scalar volume data sets represent $f(\vec{x})$ by samples taken at various locations in space that are given by an underlying sampling grid. Medical data sets, as they arise from computed tomography (CT), magnetic resonance imaging (MRI) or ultrasound (US), for instance, are acquired on a uniform rectilinear grid, whereas finite element simulations (FEM) generate unstructured grids based on tetrahedra or prisms. Since values of $f(\vec{x})$ are only given at discrete locations in space, an interpolation scheme is required to obtain the intermediate values. Modern graphics hardware, for instance, supports *trilinear interpolation* for uniform grids. Unless mentioned otherwise, all data sets used in this thesis are given on a uniform rectilinear grid.

4.1.2 Physically Based Direct Volume Rendering

The basic idea behind direct volume rendering is to simulate the propagation of light in terms of a participating medium. The medium, whose properties are defined by the data set, is treated as a gaseous material which interacts with light passing through it. Typically, models consider up to three different types of interaction:

Emission. Light can be emitted by the gaseous medium, increasing the radiative energy of a light ray traveling through the volume.

Absorption. Light can be absorbed by the gaseous medium, decreasing the radiative energy.

Scattering. The direction of a light ray can be changed, as it travels through the medium.

The *emission-absorption* model described by Max et al. [Max95] neglects the scattering of light and assumes that the radiative energy of a volume can be described in terms of an energy emitting source component and an attenuating extinction component. In this model, the *volume rendering equation* is formulated as

$$\frac{dI(t)}{dt} = g(t) - \tau(t)I(t), \tag{4.2}$$

where $I(t)$ describes the amount of radiative energy along the direction of light flow, $g(t)$ the emission of light from the medium, and the extinction coefficient $\tau(t)$ the attenuation by the medium. *Transfer functions* assign emission and extinction coefficients to scalar values of the data set. Solving for $I(t)$ by integration yields the *volume rendering integral*

$$I(D) = I_0 e^{-\int\limits_{t_0}^{D} \tau(t')\,dt'} + \int\limits_{t_0}^{D} g(t) e^{-\int\limits_{t}^{D} \tau(t')\,dt'}\, dt \tag{4.3}$$

that describes the radiance leaving the volume between an end point $t = D$ and an initial point $t = t_0$,with I_0 denoting the background light at $t = t_0$. With *transparency* defined as

$$T(t) = e^{-\int\limits_{t_0}^{D} \tau(t')\,dt'}, \tag{4.4}$$

one obtains

$$I(D) = I_0 T(t_0) + \int\limits_{t_0}^{D} g(t) T(t)\, dt. \tag{4.5}$$

The integral can be approximated by a Riemann sum over n equidistant segments of length $\Delta x = (D - t_0)/n$, resulting in

$$I(D) \approx I_0 \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_i, \tag{4.6}$$

where

$$t_i = e^{-\tau(i\Delta x + t_0)\Delta x} \tag{4.7}$$

is the transparency of the ith segment and

$$g_i = g(i\Delta x + t_0)\Delta x \tag{4.8}$$

is the source term for the ith segment. Instead of using transparencies t_i, it is common to use *opacity* values defined as $\alpha_i = 1 - t_i$.

Transfer functions map scalar values to colors C_i and opacities α_i. The discrete volume rendering integral given in equation 4.6 can be computed iteratively by either *front-to-back* or *back-to-front* composition. In front-to-back compositing, the

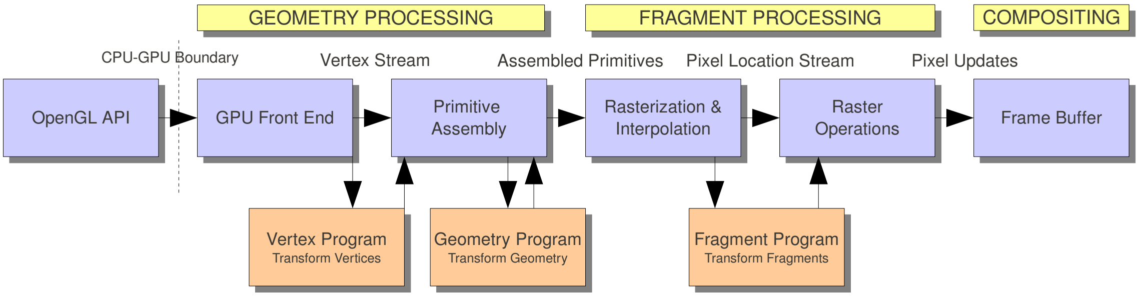


Figure 4.1: The programmable GPU graphics pipeline.

viewing ray is traversed from the eye point into the volume. The color and opacity equations are

$$C_{dst} \leftarrow C_{dst} + (1 - \alpha_{dst}) C_{src},$$
$$\alpha_{dst} \leftarrow C_{dst} + (1 - \alpha_{dst}) \alpha_{src}.$$

The accumulated color C_{dst} and opacity α_{dst} are updated by values C_{src} and α_{src} which denote the optical properties assigned by the transfer function at the current location along the ray. Back-to-front compositing traverses the viewing ray in opposite direction, accumulating colors values via

$$C_{dst} \leftarrow (1 - \alpha_{src}) C_{dst} + C_{src}.$$

4.1.3 The Graphics Pipeline

Graphics Processing Units (GPUs) convert streams of vertices defining objects in a three-dimensional scene into a raster image that is displayed on the video screen. The conversion takes place in several consecutive stages of the *graphics pipeline*, which is sketched in figure 4.1. Until a couple of years ago, the stages of the graphics pipeline were fix and could not be altered. Modern programmable GPUs, however, allow programmers to replace certain parts of the fixed-function pipeline by self-written microprograms, so-called *shader programs*.

Detailed information about the graphics pipeline is given in the OpenGL Programming Guide [SWND07]. The three basic stages of the graphics pipeline are:

Geometry processing. During geometry processing, vertices are transformed from three-dimensional object space into world space, eye space, and normalized screen space. For this, matrices for modelview-, projection-, and viewport transformation are consecutively multiplied with vertex coordinates. Sets of vertices are grouped into geometric primitives during primitive assembly.

Fragment processing. During fragment processing, screen space primitives are rasterized into fragments, i.e., potential screen pixels. Each fragment is processed independently. Filter and texture operations are performed to determine the color of a fragment.

Compositing. Since multiple fragments can correspond to a screen pixel, a series of frame buffer tests are performed to determine if and how a single fragment contributes to the color of a screen pixel.

Vertex programs (vertex shaders) are called once per vertex and replace the traditional fixed-pipeline transformation from object space into screen space. Vertex shaders have access to application-defined vertex attributes, such as position, color, and texture coordinates. They can manipulate these parameters but cannot discard vertices or generate new ones.

Geometry programs (geometry shaders) were first introduced with the implementation of Shader Model 4.0 in the NVIDIA GeForce 8800 series [CG]. Geometry shaders take the output of the primitive assembly stage (vertices including adjacency information) as an input and generate new primitives or manipulate existing ones. They are able to add or remove vertices from a mesh or procedurally generate new geometry.

Fragment programs (fragment shaders) are called once per fragment during fragment processing after primitives have been rasterized and vertex attributes have been interpolated via barycentric interpolation. Fragment shaders are used to calculate the final color of a fragment as an RGBA quadruple that is processed afterwards during compositing in the frame buffer.

The use of shader programs is optional. For instance, if a fragment program has been implemented but the vertex program is omitted, the system falls back to the fixed-function pipeline during geometry processing. The same happens if a vertex program has been implemented but the fragment program is missing. However, implementing a geometry shader without vertex shader is invalid.

Shader programming is supported by various application programming interfaces (APIs). OpenGL provides a C-like shader language called OpenGL Shading Language (GLSL) [RLKG$^+$09] which integrates into the traditional OpenGL API and is mostly similar to NVIDIA's C for Graphics (CG) [FK03].

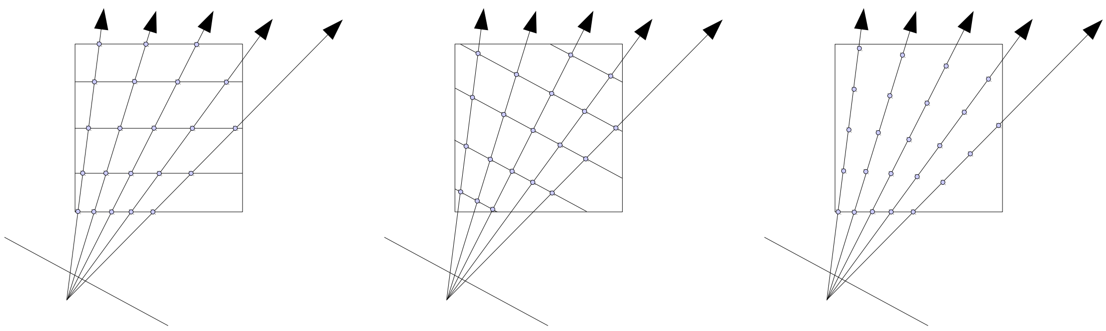
4.1.4 Direct Volume Rendering Techniques

Various algorithms have been invented to generate images of scalar field data. At the top level, methods can be divided into *indirect* and *direct* approaches. Indirect volume rendering techniques compute isosurfaces from the data, i.e., regions of the volume representing a constant scalar value. A prominent example is the Marching Cubes (MC) algorithm by Lorensen et al. [LC87]. Direct methods seek at visualizing the volume as a semi-transparent medium by approximating the volume rendering equation described in section 4.1.2. They can be divided into *image-order* and *object-order* techniques. While image-order algorithms consider each pixel of the final image separately and compute the contribution of the entire volume to the final color of a pixel, object-order algorithms compute the contribution of each voxel on the final image.

4.1.4.1 Texture-Based Methods

Texture-based slicing is an object-order method that exploits built-in texture-mapping capabilities of GPUs to visualize scalar data sets.

2D texture slicing [HKRs$^+$06] stores volume data as a set of 2D textures which are mapped to a stack of proxy geometry. Proxy geometry usually consists of axis-aligned polygon slices that are parallel to one of the faces of the data set (see figure 4.2(a)). Since there is only bilinear interpolation between 2D texture coordinates within the same slice and no interpolation between slices, three stacks of 2D textures must be stored, one for each coordinate axis. In order to reduce artifacts, the system has to switch between stacks so that the angle between viewing ray and slice normal is always minimal.



(a) (b) (c)

Figure 4.2: Sampling schemes for various direct volume rendering approaches: (a) 2D slicing: sampling scheme for object-aligned slices, (b) 3D slicing: sampling scheme for viewport slices, (c) Ray casting: equidistant sampling scheme.

3D texture slicing [CCF94, CN94] stores the volume in a single 3D texture which is mapped to a set of polygon slices. In contrast to 2D slicing, however, slices are view-aligned and texture coordinates are interpolated trilineary by the graphics hardware (see figure 4.2(b)). The advantage of 3D texture slicing over 2D texture slicing is that memory requirements are lower since there is no need for multiple texture stacks. Furthermore, there are no switching artifacts so that the overall perception is smoother when transforming the volume. In the past, 3D texture slicing was considered to be slower than 2d slicing because trilinear interpolation was more expensive. Today, however, 3D texture mapping is a natively built-in function of all GPUs and performance differences are negligible. 3D texture slicing is implemented in the out-of-core system presented in section 4.2.

4.1.4.2 GPU-Based Ray Casting

Ray casting is an image-order method that computes the final image pixel by pixel by casting rays into the volume. Rays are traversed in front-to-back order (from the eye point into the volume) so that samples taken along the rays can be accumulated as explained in section 4.1.2 to yield the final color and opacity of a pixel (see figure 4.2(c)). Volume ray casting on the CPU has been studied since the 1980s [Lev88]. However, the advent of programmable shader units enabled the implementation of ray casting algorithms in a fragment program on the GPU. The first implementations were presented by Röttger et al. [RGW$^+$03] and Krüger et al. [KW03]. The actual ray casting code is straightforward and sketched in listing 4.1 as GLSL pseudo-code.

Listing 4.1: GLSL Ray Casting

```
 1  //3D texture storing the volume
 2  uniform sampler3D volume;
 3  //transfer function
 4  uniform sampler1D tf;
 5
 6  //ray origin, ray direction, step size
 7  vec4 dvr(vec3 org, vec3 dir, float dt)
 8  {
 9    vec4 dp =
10         vec4(ray_dir.x, ray_dir.y, ray_dir.z, 1) * dt;
11    //current sample position
12    vec4 p;
13    //initial sample point
14    p.xyz = ray_org + ray_dir * tenter;
15    p.w = tenter;
16
17    //stores the final color
18    vec4 rgba = vec4(0,0,0,0);
19
20    //dvr loop
21    for(;;)
22    {
23      //get interpolated value
24      float f = texture3D(volume, p.xyz);
25      //apply transfer function (classification)
26      vec4 sample = texture1D(tf, f);
27      float a_1msa = alpha * (1 - sample.w);
28      //blend rgb values
29      rgba.xyz += sample.xyz * a_1msa;
30      //blend alpha values
31      rgba.w += a_1msa;
32      //continue along ray
33      p += dp;
34      //ray leaves volume or maximum opacity is reached
35      if (p.w > texit || rgba.w > term_alpha) break;
36    }
37    return rgba; //return final pixel color
38  }
```

Acceleration methods for ray casting aim at minimizing the number of samples to be taken in the direct volume rendering (dvr) loop (line 21). Common optimization strategies are:

Early termination. The dvr loop is exited either when the ray leaves the volume or a maximum opacity value is reached (line 35).

Adaptive sampling. The sample rate varies along the ray. Instead of sampling at equidistant positions (line 33), the step size adapts dynamically based on an importance metric.

Empty space skipping. Regions of empty space can be skipped because they do not contribute to the final color of a pixel. Acceleration structures help identifying those regions and are discussed section 4.1.5.

Section 4.3 focuses on fast ray casting implementations for CPU and GPU in order to achieve interactive frame rates on large high-resolution displays.

4.1.4.3 CPU-Based Techniques

CPU volume renderers can be accelerated through rigorous usage of *Streaming SIMD Extension* (SSE) operations on x86 architectures [HLS, Int]. SSE was introduced in 1999 with the Pentium III series and provides a way to execute Single-Instruction, Multiple-Data (SIMD) operations on floating point vectors. Therefore, the CPU register set was originally extended by eight 128-bit registers, `XMM0` through `XMM7`. Each register is able to pack vectors of various data types. Depending on the SSE version, different data types are supported:

`_m128`. Four 32-bit floating point values are packed into one `XMM` register.

`_m128d`. Two 64-bit floating point values are packed into onw `XMM` register.

`_m128i`. Sixteen 8-bit, eight 16-bit, four 32-bit, or two 64-bit integer values are packed into one `XMM` register.

Compilers, such as the Visual C++ compiler or GCC, support SSE through compiler intrinsics. Adding two 4-vectors (32-bit floating point), for example, is simply done via `_m128 _mm_add_ss(_m128 a, _m128 b);`. Thus, vector-wise processing yields a speed-up of four compared to the naive iterative computation.

CPU volume graphics strongly benefits from SSE, as many operations work on three-dimensional homogeneous vectors, represented as 4-tuples. The work of Wald et al. [WSBW01, WFKH07a] strongly exploits SSE for coherent packet ray tracing, in which packets of rays are traced through the volume instead of single rays. Details of the coherent packet algorithm are discussed in section 4.3.2 and section 4.3.3.

4.1.5 Acceleration Structures for Large-Scale Data Sets

Acceleration structures enable direct volume rendering systems to deal with large data sets and achieve high frame rates. Many data structures subdivide volume

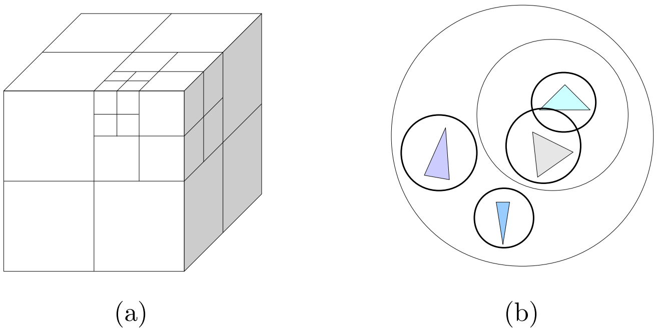

(a) (b)

Figure 4.3: Acceleration structures: (a) recursive octree space subdivision, (b) Bounding Volume Hierarchy (BVH).

space hierarchically based on a subdivision criterion, which can be exploited, e.g., for frustum culling, empty space skipping, and adaptive sampling metrics. The following section introduces acceleration structures that are used in subsequent parts of this chapter.

4.1.5.1 Octrees

Octrees are tree-like, three-dimensional acceleration structures that recursively subdivide space into regions of non-overlapping, axis-aligned boxes. The root of the octree represents the entire space that is further divided. New nodes are generated by progressively splitting parent nodes into eight child nodes of equal size. Subdivision stops when a termination criterion is met, e.g., a maximal tree depth, a minimal number of geometric primitives (for polygonal data sets), an empty region, or a minimal volume/axis dimension. Figure 4.3(a) illustrates how space is subdivided recursively by an octree. In computer graphics, octrees have been successfully applied to compression, simplification, isosurface extraction, and rendering itself. A comprehensive survey of octree volume rendering methods is given by Knoll et al. [Kno08].

Section 4.2 introduces an out-of-core system employing an acceleration structure that combines octree space subdivision characteristics with a multi-resolution representation of volume data based on wavelets.

4.1.5.2 Bounding Volume Hierarchies

Bounding Volume Hierarchies (BVHs) are tree-like acceleration structures whose nodes hierarchically enclose the objects of a scene (see figure 4.3(b). In contrast to octrees, BVHs are trees of arbitrary degree, i.e., their nodes have arbitrary numbers of child nodes. In practice, the degree is often limited to two, so that the BVH corresponds to a binary tree. BVHs do not partition space per definition, since nodes can overlap and do not have to correspond to Axis-Aligned Bounding Boxes (AABBs). However, when used for volume data, the build can be adapted so that nodes represent mutually exclusive AABBs and partition space. The structure of a BVH tree strongly depends on the build metric that determines whether to treat a bounding volume as an inner node or a leaf and how to split inner nodes. Ideally, the hierarchy of bounding volumes is "tight" and approximates the scene with as few bounding volumes as possible. For geometric data sets, the Surface Area Heuristic (SAH) [GS87] is an often used metric trying to minimize the traversal costs when splitting a node.

Wald et al. [WBS07] introduced the *implicit* BVH for interactive ray tracing of deformable triangle mesh scenes. Implicit BVHs resemble min-max trees which store additional information about the minimum and maximum scalar value of a subvolume in each node of the hierarchy. Using the min-max values, it is easy to cull entire regions of the data set by checking the range of values.

Section 4.3 introduces direct volume rendering implementations that exploit implicit BVH trees as acceleration structures on the GPU, the CPU, and on a hybrid CPU/GPU system.

4.1.6 Multi-resolution Techniques

Large-scale data sets comprise gigabytes to terabytes of volume data and easily exceed the computational resources of a computer system (i.e., HDD space, RAM, and texture memory). A simple approach to tackle this problem is to downsample data sets and generate multiple low-resolution representations of a high-resolution volume. However, downsampling implies loss of information and increases the amount of resources occupied by multiple copies of the same original volume.

Multi-resolution techniques based on wavelets are an easy, yet smart way to generate scaled-down versions of data. A wavelet transformation rearranges the information so that the new representation requires the same amount of memory as the original data set, i.e., the transformation is lossless. However, the representation comprises multiple levels of detail which can be progressively refined. The following description solely focuses on *Haar wavelets*, the simplest of all wavelet forms. It follows the excellent introduction by Stollnitz et al. [SDS95a, SDS95b].

4.1.6.1 Haar Wavelets

Wavelets are a mathematical tool for hierarchically decomposing functions. With wavelets, a function can be described in terms of a coarse overall shape, plus details that range from broad to narrow. Volumetric data sets as introduced in section 4.1.1 can be interpreted as three-dimensional discrete functions and thus be represented using wavelets. Before discussing the mathematics of *wavelet transformations* (also known as *wavelet decompositions*), an example using a one-dimensional image is given to illustrate the general idea of the method. The example and the rest of this chapter focus on a special group of wavelets called *Haar wavelets*, which correspond to a set of basis functions called *box functions.*
Let

$$[9\ 7\ 3\ 5]$$

be a one-dimensional image consisting of four pixels. A lower-resolution image containing half the number of pixels is obtained by pairwise averaging of neighboring pixel values, i.e.,

$$[8\ 4].$$

By storing detail coefficients, the original image can be reconstructed. In the example, 1 and -1 are stored as detail coefficients. The first average is 1 less than 9 and 1 more than 7, i.e., $8 + 1 = 9$ and $8 - 1 = 7$. The second average is 1 more than 3 and 1 less than 5, i.e., $4 + (-1) = 3$ and $4 - (-1) = 5$. The full decomposition can be obtained by recursively repeating this process of pairwise averaging and differencing:

Resolution	**Averages**	**Detail Coefficients**
4	[9 7 3 5]	
2	[8 4]	[1 -1]
1	[6]	[2]

Each decomposition step reduces the resolution by one half and generates a set of detail coefficients. The final wavelet transformation can be written as a single coefficient representing the overall image plus a series of detail coefficients in order of increasing resolution for the reconstruction, i.e.,

$$[6\ 2\ 1\ \ -1].$$

Before describing the wavelet transformation of volume data sets, the concept is first formalized for one-dimensional images. A generalization for three-dimensional data sets is straight forward and introduced afterwards.

Let V^j denote the vector space of all piecewise-constant functions defined on the interval $[0,1)$ with constant pieces over each of 2^j equal-sized subintervals. A one-dimensional image $\mathcal{J}(x)$ containing 2^j pixels is an element of V^j. Because piecewise constant functions with two intervals can be described as a piecewise constant function with four intervals, when each interval of the first function corresponds to a pair of intervals in the second, the vector spaces V^j are nested, i.e., $V^j \subset V^{j+1}, j \in \mathbb{N}_0$. Basis functions for V^j are given by a set of scaled and translated *box functions*

$$\phi_i^j(x) := \phi(2^j x - i),\ i = 0, \ldots, 2^j - 1, \tag{4.9}$$

where

$$\phi(x) := \begin{cases} 1 & \text{for } 0 \leq x < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

As an example, figure 4.4 illustrates the box basis functions for V^2.

Another vector space W^j can be defined that comprises all functions of V^{j+1} and is orthogonal to V^j. The basis functions spanning W^j are the so-called *Haar wavelets* given by

$$\psi_i^j(x) := \psi(2^j x - i),\ i = 0, \ldots, 2^j - 1, \tag{4.11}$$

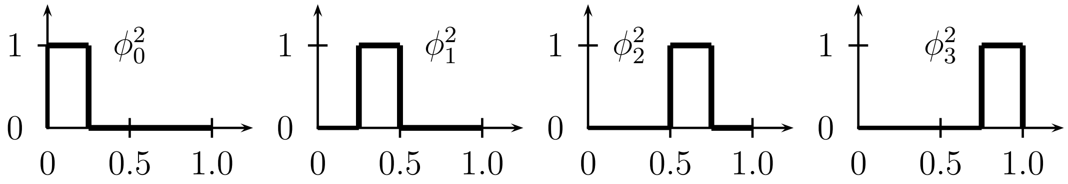


Figure 4.4: The box basis functions for V^2.

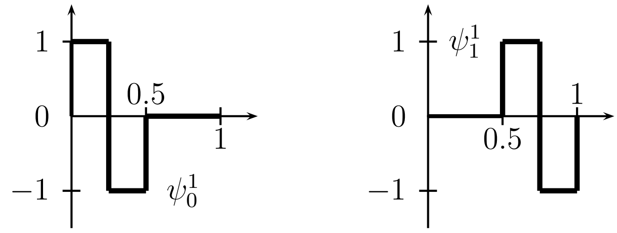


Figure 4.5: The Haar wavelets for W^1

where

$$\psi(x) := \begin{cases} 1 & \text{for } 0 \leq x < \frac{1}{2}, \\ -1 & \text{for } \frac{1}{2} \leq x < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.12}$$

The Haar wavelets for W^1 are illustrated in figure 4.5.
An image $\mathcal{J}(x)$ containing $n = 2^j$ pixels with pixel data c_i^j $(i \in \mathbb{N}_0)$ can be described as

$$\mathcal{J}(x) = \sum_{i=0}^{n-1} c_i^j \phi_i^j(x). \tag{4.13}$$

After the first decomposition step, $\mathcal{J}(x)$ can be written as

$$\mathcal{J}(x) = \sum_{i=0}^{\frac{n}{2}-1} c_i^{j-1} \phi_i^{j-1}(x) + \sum_{i=0}^{\frac{n}{2}-1} d_i^{j-1} \psi_i^{j-1}(x). \tag{4.14}$$

The first sum of the equation represents the reduced image resulting from pairwise averaging and contains the low-pass filtered components of the signal. The second sum comprises the high-pass filtered detail coefficients. By continuing recursively, the full decomposition of the image is obtained.

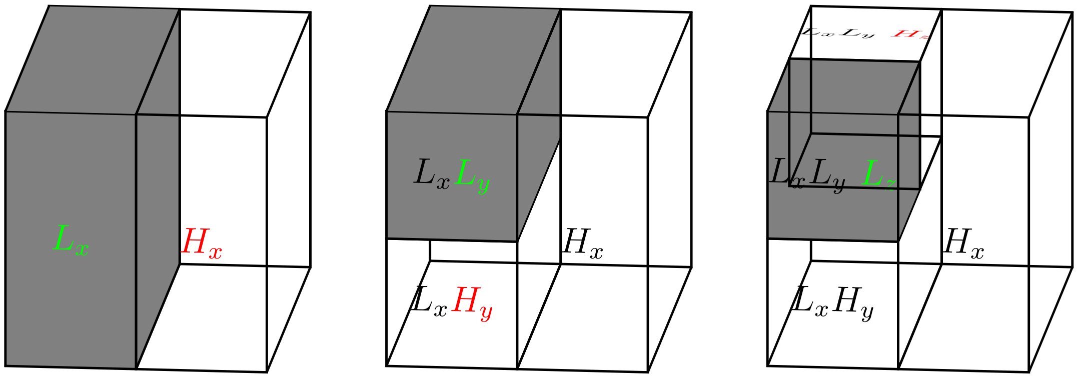


Figure 4.6: Non-standard decomposition in three dimensions.

The method described for one-dimensional discrete functions is expanded into a *non-standard decomposition* for volume data by sequentially performing wavelet transformations on alternating axis, as depicted in figure 4.6. First, all rows of the original volume are decomposed into a low-pass filtered component L_x and a high-pass filtered component H_x. For the next transformation, the algorithm is applied to the columns of L_x, resulting in L_xL_y and L_xH_y. Applying the algorithm to L_xL_y in z-direction completes the first cycle and results in $L_xL_yL_z$ and $L_xL_yH_z$. A reduced representation of the data set is now located in the left upper front corner of the volume. Subsequently, the algorithm is only applied to this quadrant and terminates when the quadrant size is one voxel in each dimension or when a maximum wavelet level n is reached.

Nguyen [Ngu08] describes an efficient file storing scheme supporting incremental refinement. The low-pass filtered block $(L_xL_yL_z)_n$ at level n is stored at the beginning of a file, followed by the detail coefficients at each level for each dimension. The order of blocks in a file is as follows:

$$(L_xL_yL_z)_n, (L_xL_yH_z)_n, (L_xH_y)_n, (H_x)_n, (L_xL_yH_z)_{n-1}, (L_xH_y)_{n-1}, (H_x)_{n-1} \dots$$

An object can thus be refined by sequentially loading detail coefficients and using them for the reconstruction.

Contrary to a non-standard decomposition, a *standard decomposition* performs a full transformation on each dimension of the volume before switching to the next. The two transformation schemes yield to different basis functions for the higher-dimensional case. As a result, computing the non-standard decomposition is slightly more efficient than the standard decomposition. For further details the reader is referred to Stollnitz et al. [SDS95a, SDS95b].

4.1.7 Large Display Volume Data Systems

Medical applications, e.g., CT or MRT, often yield data sets consisting of image slices stacked on top of each other. Instead of rendering the entire volume, an approach often favored by physicians out of habit is to display each slice of the stack separately. There are several systems that support the visualization of high-resolution imagery on large displays. JuxtaView [hip], for example, is a cluster-based application for scalable tiled displays. The software implements a distributed approach for out-of-core montage visualization using LambaRAM, a software-based network-level cache system. TileViewer [KVV$^+$04], a similar system for displaying large-scale imagery, exploits the tiled pyramidal TIFF format to display images with a resolution of 200-600 megapixels on HIPerWall.

However, there are a couple of volume rendering systems that have been particularly designed for scalable large displays: Schwarz et al. [SVR$^+$04] present Vol-a-Tile, an interactive tool for exploring large volumetric data on scalable tiled displays. Vol-a-Tile is able to visualize seismic data sets on the 5×3 GeoWall2 display grid at the Electronic Visualization Laboratory (EVL). The authors implement a master-slave prototype using an MPI-based rendering library and an OptiStore data streaming server. Unfortunately, implementation specific details are mostly spared in the original paper. The TeraVoxel project [LMS$^+$01] founded at the California Institute for Technology is able to interactively render a $256 \times 256 \times 1,024$ data set on a $3,840 \times 2,400$ display using four VolumePro 1000 graphics cards by employing pipelined associative blending operators. In contrast to the approach presented in section 4.2, TeraVoxel employs a sort-last strategy and graphics hardware particularly optimized for volume rendering.

QSplat [RL00] is a multi-resolution rendering system for displaying isosurfaces out of point data sets. It uses a progressive refinement technique and generates images using a splatting approach.

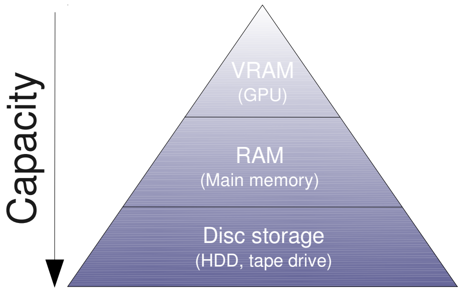


Figure 4.7: The HDD-CPU-GPU memory hierarchy.

4.2 Out-of-Core Techniques for Large-Scale Volume Rendering on Distributed Display Clusters

Computer systems often cannot handle the amount of information that is comprised in large-scale data sets without advanced visualization techniques. The Visible Human Project [oM], for instance, provides 65GB of data for a real-color RGB scan of a male cadaver, and 40GB of data for the female cadaver, asking for sophisticated methods to process the enormous amount of information. The number of pixels of regular computer monitors is too low to display such complex data sets at their full level of resolution. For three-dimensional visualizations, the gap between the amount of information contained in a scene and the available screen space is even more critical than in the 2D case. Large high-resolution displays offer enough screen real estate to render these volumes, but require methods to deal with the size of data sets on a per node basis. The memory hierarchy illustrated in figure 4.7 shows the size of data sets that can be visualized with GPU methods is ultimately limited by the amount of video RAM (VRAM) on the graphics card.

The main contribution of this section is the description of an out-of-core strategy to overcome the RAM-VRAM memory gap. The strategy enables the use of GPU methods to render volume data on large displays whose size exceeds the available amount of video memory in a render node of the cluster. The CPUs on the cluster driving the GPUs are used to split up the computation of a scene into different areas, where each area is computed by a different rendering node. Octree-based out-of-core frustum clipping is combined with wavelet-based multi-resolution data representation to increase the visual quality of scenes. 3D texture slicing is used in the prototype system to generate renderings for each tile. However, the out-of-core technique is mostly independent of the GPU algorithm and can be replaced by other visualization methods, such as GPU ray casting (see section 4.1.4.2). The system was implemented and tested on the 10 × 5 HIPerWall display grid.

4.2.1 Data Structures

This section presents details of the multi-resolution data structure which was first described by Meyer et al. [MBH+02] in a different context. It has been used to implement a network-based rendering application that is able to visualize data sets between 20MB and 76GB size by first transferring low-resolution versions of the data via network and then gradually refine them.

4.2.1.1 Adaptive Octree

The volumetric data sets considered in this section are too complex to be rendered in real time on hardware for cluster nodes and with current 3D texture-based rendering techniques. They must be reduced to a reasonable size, so that each node can quickly access the appropriate section of the 3D volume and render this portion of the data instantaneously. The slice format in which volume data often comes is not suitable for this purpose, because too many files need to be opened in order to display a 3D volume from cross-sections. Opening large numbers of files has proven to be prohibitively slow. Therefore, the octree space subdivision algorithm described in section 4.1.5.1 is employed to rearrange the data. Instead of using a database, the Unix file system and an efficient indexing scheme are used to access the right block of data instantly. Three simple binary decisions (left or right, front or back, and top or bottom) are necessary at each level to determine the position of a voxel in the octree. Each decision corresponds to a two bit index, which can take on $2^3 = 8$ states (0-7), i.e., the eight child nodes of the octree. The number represents the file name, and several numbers in a file name indicate an octree traversal path. With this fast indexing scheme, it is possible to traverse the tree almost instantaneously in order to get to the leaf of the tree which carries the data.

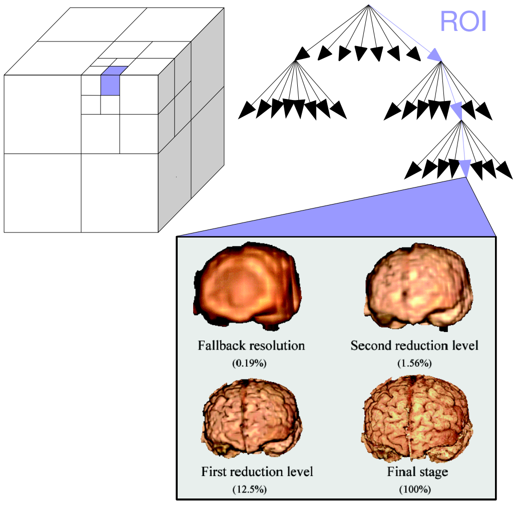


Figure 4.8: Octree space subdivision in combination with multi-resolution wavelet representation.

A major advantage of the octree data structure is the early termination of the traversal in branches that are empty. It works best for data sets where the object is located near the center of the data cube and has an empty or nearly empty background (e.g., for CT or MRI scans), but it also works for sparse data sets. The information is determined in a preprocessing step when creating the octree data structure. Empty areas are not stored, saving a significant amount of disk space and RAM.

At this point, depending on the depth of the octree, the data block to be read may still be too large to fit into the texture buffer of a rendering node's graphics card. For this reason, leaves are stored in a multi-resolution wavelet format.

4.2.1.2 3D Non-standard Haar Wavelet Decomposition

A three-dimensional non-standard Haar wavelet decomposition method is used to store the leaf information. As described in section 4.1.6.1, the benefit of using this method is the fact that low-resolution information can be stored at the beginning of a file, and if more detail is desired, more data in the form of detail coefficients can be loaded in order to refine the image. Depending on the number of wavelet levels, a tremendous reduction of data can be accomplished with acceptable visual artifacts in the resulting image. On the first level, only 1/8th of the data needs to be rendered, on the second level 1/64th (less than 2% of the total data set), and so forth. This exponential reduction leads to a significant acceleration and memory savings. An important advantage of the wavelet approach is the fact that the same amount of memory is used to store the hierarchy as to store the original volume. Although the decomposition would typically be implemented using floating point arithmetic to calculate the detail coefficients, it is possible to replace this data type by integers with no loss of precision as described by Calderbank et al. [CDSY98], and make use of the fact that integer arithmetic is much faster than floating point arithmetic on CPUs.

4.2.2 System Details

The prototype was implemented on HIPerWall using CGLX, a rendering library specifically developed for the fifty tile display cluster and other, similar configurations.

4.2.2.1 CGLX - A Cross-Platform Cluster Graphic Library

As discussed in section 2.1.3, there exists a variety of distributed rendering libraries. The system described in this section is based on CGLX - a Cross-Platform Cluster Graphic Library [DK] that was developed at the California Institute for Telecommunications and Information Technology at UC San Diego. CGLX is similar to *AnyScreen* and is a flexible and transparent OpenGL-based graphics framework for distributed visualization systems in a master-slave setup. Communication between nodes is realized through a light-weight, thread-based network communication protocol. Attempting to be as transparent as possible to the developer, CGLX offers a GLUT-like (OpenGL Utility Toolkit) interface, which allows the library to intercept OpenGL calls and provide a distributed large scale OpenGL context on a tiled display.

Theoretically, applications can be ported to CGLX by only changing the traditional `GL/glut.h` include to `clgX.h` and calling `cglXOrtho`, CGLX's equivalent to `glOrtho`. In practice, however, things like synchronized access to global variables and shared resources require additional modifications. Other than in standard GLUT environments where usually only one context/window is opened, CGLX handles multiple OpenGL windows which together form one large OpenGL canvas. Most registered callback functions are executed on a per context basis. As a result, in default mode, CGLX processes one context after the other. The reader is referred to the CGLX documentation [DK] for further details about different execution modes.

4.2.3 System Design

When doing volume rendering of large-scale data sets, the limited resources of the system pose restrictions to the amount of information that can effectively be visualized. Above all, the available amount of video RAM determines the size of data sets that can be visualized with GPU methods on a given system. The same restrictions apply when designing cluster applications, where each node by default computes the entire scene. The waste of system resources is enormous, since nodes allocate memory for the entire scene, but only display a small part of it. The approach presented in the following seeks to make better use of the given resources by employing the octree-wavelet data structure described previously. The idea is to first determine those parts of a scene that need to be rendered by a particular node and cull the rest of the volume. Then, a maximum level-of-detail representation is calculated for the remaining subvolume so that it still fits into the texture memory of the GPU. In order to reduce sharp visual transitions between different levels of detail, the representation is refined progressively by incrementally loading detail coefficients from the wavelet files. First, an overview of the rendering algorithm is given, followed by a description of the progressive refinement procedure.

4.2.3.1 Rendering Algorithm

The rendering algorithm employed in the prototype maps 3D textures to stacks of view-aligned proxy geometry as described in section 4.1.4.1. 3D texture mapping is able to produce good-quality results with decent performance and is easy to implement. However, the combined octree-wavelet data structure does not depend on a particular visualization method and solely aims at providing chunks of volume data with highest possible resolution for subsequent rendering. The frame rates achieved with a particular rendering algorithm surely have influence on the overall performance of the system. If all nodes are able to render at fast speeds, the overall performance increases. Section 4.3 deals with various GPU and CPU optimization methods to increase performance.

4.2.3.2 Progressive Refinement

Each time the volume is transformed by translation, rotation, or scaling operations, frustum clipping must be executed since parts of the data might cover a different set of LCDs. Frustum clipping slows the system down because it implies time consuming file operations to load new data chunks from hard disk into main memory. Therefore, it is impossible to clip without perceivable transitions at interactive rates. The system's solution for this bottleneck is a *fallback* texture. This texture permanently resides in VRAM, is loaded once at start-up time and contains the entire, scaled-down volume. Whenever the volume is transformed, i.e., when the user interacts with the volume, the system switches from the individually computed subvolumes of a node to the fallback texture. The size of the fallback texture is chosen in a way that each rendering node maintains a minimum frame rate. That way, the system is kept responsive and the user is provided with permanent visual feedback. Because the fallback texture contains the entire data set and has to be stored once on each node, its detail level is usually lower than the ones that result from the clipping process.

When the mouse is released, i.e., when user interaction is paused, the system increments a counter, which determines the current detail level and triggers the loading of detail coefficients from the wavelet files for the reconstruction. A node starts at an initial level i, displays the data set at that resolution and then, triggered by the counter, switches to level $i-1$, $i-2$ and so forth. The image gradually refines up to the maximum level each graphics card can display. For small movements through the volume, where only parts of a scene change, the user experience can in the future be improved by loading additional high-resolution data around a hull of the frustum.

Node synchronization is achieved through CGLX's built-in synchronization mechanism. Before OpenGL render buffers are swapped, a barrier synchronization lets the system stall until all threads reach the barrier. While this means having to wait for the slowest node in the cluster, it provides a way to switch synchronously between wavelet levels during progressive refinement and reduce artifacts.

4.2.4 Implementation

Frustum clipping determines the subvolumes that each node of the cluster has to render. Before rendering into the frame buffer, nodes must assemble the volume in order to yield the correct visual results.

4.2.4.1 Out-of-Core Frustum Clipping

During out-of-core clipping, all rendering nodes determine for each of their monitors, i.e., two in the case of HIPerWall, which parts of the data set contribute to the scene on their screens. This yields a subvolume for each screen for which the respective files are loaded at the node. The subvolume is loaded progressively until a maximum detail level is reached that still lets the data fit into a predefined portion of the graphics card's texture memory. As a result, frustum clipping yields a partial copy of the entire data structure from hard disk in a node's main memory, ready to be sent to the GPU for visualization.
To determine whether an octree brick is displayed on a particular monitor, its eight corners are projected into 2D space and their window coordinates are calculated. Determining the two-dimensional window coordinates (wx, wy) of a three-dimensional point $v = (ox, oy, oz, 1.0)$ given in homogeneous coordinates in object space under a modelview matrix M, a projection matrix P, and a viewport V can be accomplished via OpenGL's `gluProject()` method [SWND07]. `gluProject()` first computes $v' = P \cdot M \cdot v$. The actual window coordinates are obtained through

$$\begin{aligned} wx &= V[0] + V[2] \cdot (v'[0] + 1)/2 \\ wy &= V[1] + V[3] \cdot (v'[1] + 1)/2. \end{aligned}$$

The viewport V is defined by the total resolution of the display, i.e., $25,600 \times 8,000$ pixels for HIPerWall. The projection matrix P projects the viewing frustum to the viewport V. The implementation transforms volumes by manipulating OpenGL's texture matrix T. Therefore, instead of using the modelview matrix M, T^{-1} serves as input parameter for `gluProject()`. The inverse is calculated because T manipulates texture coordinates and not the actual texture image.

Once the window coordinates of the brick corners are known, their bounding box is tested for intersection with the area of each LCD in the grid. The intersections with the screen of a node define a rectangular subvolume whose content is loaded from hard disk. The wavelet representation of the brick files enables the system to load the subvolume at a level of detail that does not exceed a predefined number of voxels n in texture memory. The wavelet level for a subvolume of dimensions dx, dy, dz is calculated as follows:

Listing 4.2: Level-of-Detail Calculation

```
//level of detail, 0 = full resolution
int lod = 0;
int cMod = 1;
//does data fit into VRAM buffer of size n?
while ( (dx / cMod) * (dy / cMod) * (dz / cMod) > n )
{
  //decrease level of detail
        lod++;
        cMod = 2 * cMod;
}
//return maximum detail level
return lod;
```

Textures are limited to $n = 16.7 \times 10^6$ texels, and each texel corresponds to a voxel of the data set. A three-dimensional RGBA texture containing that many voxels comprises 64MB of volume data. Notice, each node has to store three textures: one clipping texture for each of its two monitors and one low-resolution fallback texture containing the entire data set.

4.2.4.2 Assembling the Volume

Texture buffers contain information from different regions of the data set at different levels of resolution and are rendered by mapping them to a stack of proxy polygons.

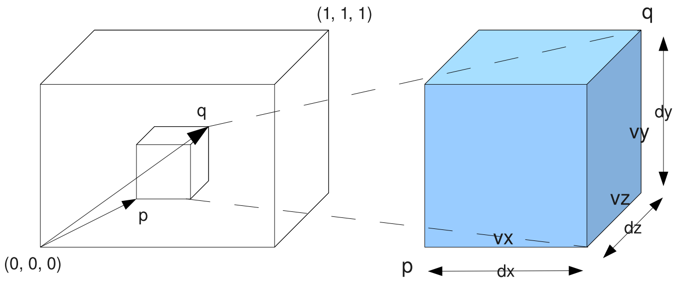


Figure 4.9: Mapping information.

In order to yield correct mapping results, texture coordinates must be calculated for each subvolume. Generally, there are two ways to calculate these coordinates: The first approach maps each subvolume to its own stack of proxy geometry, whose size and position in object space depend on the particular subvolume. The second approach maps each subvolume to the same full-screen stack of proxy geometry but scales and translates each texture so that it appears at the right position when being mapped. The second approach was implemented, as it can easily be realized with the meta information that comes with each buffer and is provided by the data structure that implements the wavelet-octree (see figure 4.9).

The following facts are known about each subvolume: vx, vy and vz denote the buffer's axis dimensions. As already mentioned, the buffer size is restricted to $vx \cdot vy \cdot vz \leq 16.7 \times 10^6$ voxels. Furthermore, the rectangle's two diagonal points $p = (px, py, pz)$ and $q = (qx, qy, qz)$ are known. Their coordinates are normalized with respect to the full size of the data set, i.e., $(0, 0, 0)$ and $(1, 1, 1)$ span the entire volume. The diagonal points p and q can be used to compute the actual portion of the data set that is represented by all $vx \cdot vy \cdot vz$ voxels of the buffer through $dx = qx - px$, $dy = qy - py$ and $dz = qz - pz$. In addition, p provides the normalized offset within the entire volume with respect to the origin $(0, 0, 0)$. With this information the calculation appropriate scaling factors and translation vectors is straightforward.

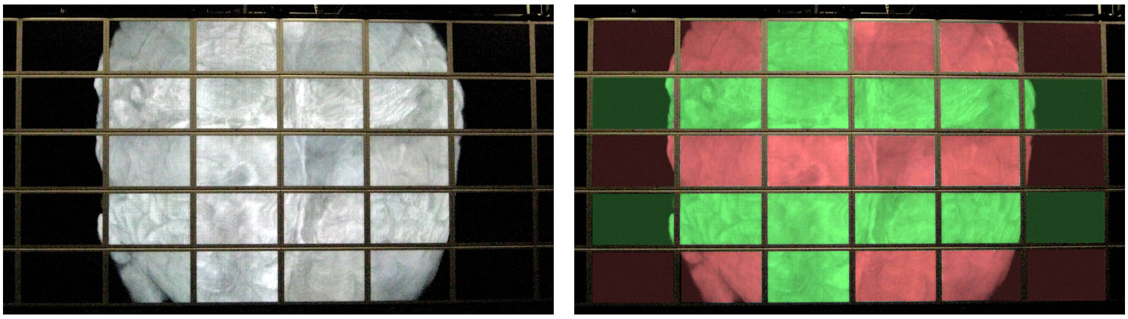
4.2.5 Results

Limiting the texture size to 16.7×10^6 voxels, i.e., 64MB for an RGBA texture, does not seem logical considering each of HIPerWall's graphics cards is equipped with 256MB VRAM. However, each node maintains three textures, all with a maximum size of 64MB, resulting in a total of 192MB. Another 31.25MB are used for the frame buffer of each display supporting a resolution of $2,560 \times 1,600$ pixels. Thus, only 32.75MB remain for additional operations. The calculation shows that a limit of 64MB per texture results in an optimal use of the available video resources. In fact, even the main memory workload of a node is kept low. Theoretically, the application only needs to allocate main memory for all three texture buffers. In practice, about twice the amount of memory is required due to the internal management of data. If more physical memory was added to the graphics cards, these observations would scale accordingly, and the visual quality of the displayed image would further improve. A variety of factors influence the quality of visualizations:

- Data sets themselves influence the quality of images as their dimensions partly determine the level of detail. Due to the limited depth of the texture buffer in the z-dimension, data sets consisting of many slices and therefore stretching deeply into the z-direction will not be displayed at the same detail level as data sets with fewer slices. This can be partially compensated by adding more view-plane parallel projection planes and by making the volume buffer asymmetrical, but this approach is limited due to memory limitations and increased rendering time.

- The depth of the octree has a major influence on the final result. An octree of depth one partitions the entire volume into eight bricks, resulting in a rather coarse subdivision for frustum clipping. An octree of depth two, however, produces 64 bricks and increases the quality since frustum clipping yields smaller subvolumes that better approximate the actual portion of data to be rendered.

- Last but not least, the current view point and zoom level have a strong effect on the detail level displayed on the wall. Zooming-in maps fewer octree bricks to a tile and can therefore increase the level of detail. Rotations and translations have a similar influence.



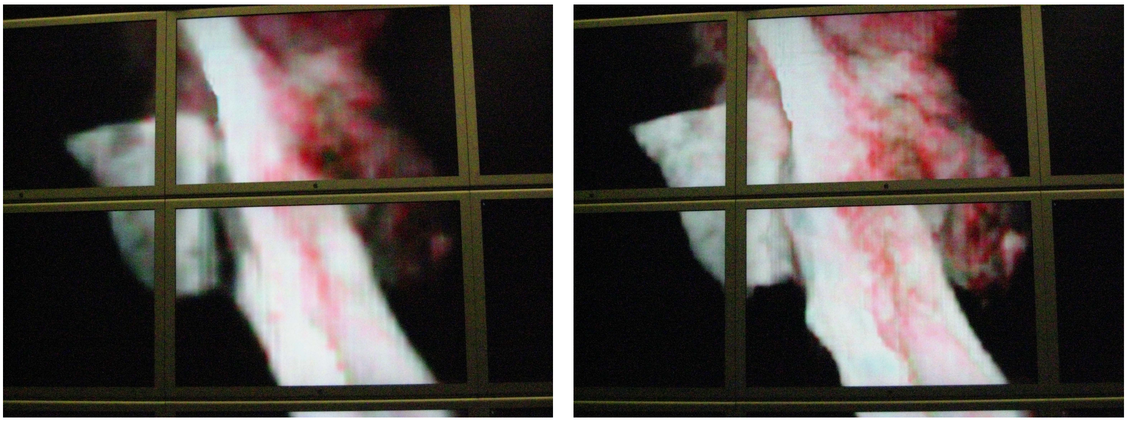
(a) (b)

Figure 4.10: Multi-resolution scene on HIPerWall: (a) Toga data set. (b) distribution of wavelet levels among monitors: red corresponds to tiles of level 2 (1/64 = 1.5% of the original quality), green to level 1 (1/8 = 12.5% of the original quality).

Figure 4.10(a) depicts a typical multi-resolution scene, in which a human brain data set is rendered at different wavelet levels on the fifty tile HIPerWall display cluster (see table 4.1 for information about data sets). In order to illustrate the distribution of wavelet levels all among tiles, screens have been color-coded in figure 4.10(b). Tiles marked red correspond to regions of wavelet level 2, meaning they display data at 1/64 = 1.5% of the resolution of the original data set. Green tiles correspond to areas of wavelet level 1, i.e., they display 1/8 = 12.5%. The close-up in figure 4.11 illustrates the difference between these two levels for a human mandible data set. In the left picture, each LCD displays at wavelet level 2, which corresponds to the detail level of the fallback texture. After refinement, the upper LCDs switch to wavelet level 1, thereby increasing the amount of detail by a factor of 8 and revealing more anatomical structures.

4.2.6 Analysis

The multi-resolution data structure is computed in a two-stage preprocessing step. In the first stage, a tool called *VolCon* computes the adaptive octree structure described in section 4.2.1.1 and converts an input data set, consisting of a set of files storing slice information, into a set of files storing octree bricks. During conversion, empty regions are culled based on a threshold criterion, i.e., their data is not stored.



(a) (b)

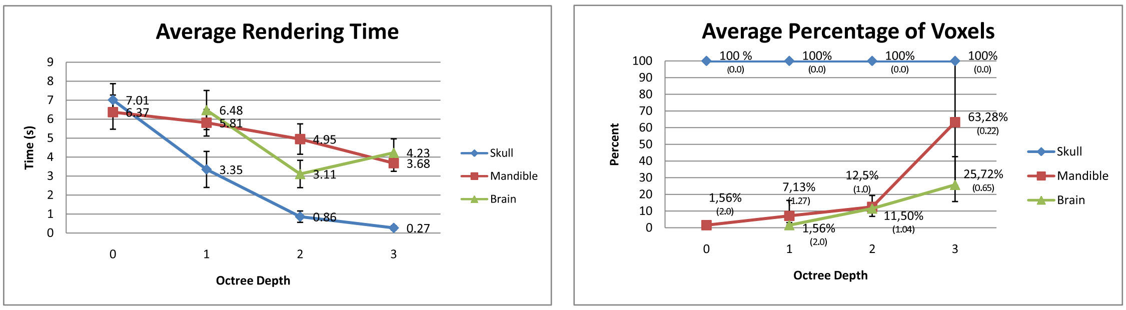
Figure 4.11: Difference between levels of resolution: (a) level 2 before progressive refinement, (b) level 1 after progressive refinement.

Data Set	**Dimensions** $(x \times y \times z)$	**Size**(MB)	**VolCon**	**Compress**
Skull	$256 \times 256 \times 113$	28.25	0m 2.5s	0m 2.7s
Mandible	$1,024 \times 1,024 \times 374$	1496	2m 17.2s	4m 2.5s
Toga Brain	$1,024 \times 1,024 \times 753$	3012	5m 9.9s	9m 45.8s

Table 4.1: Data set characteristics and preprocessing times for octree depth 2 (64 bricks) and two levels of wavelet compression.

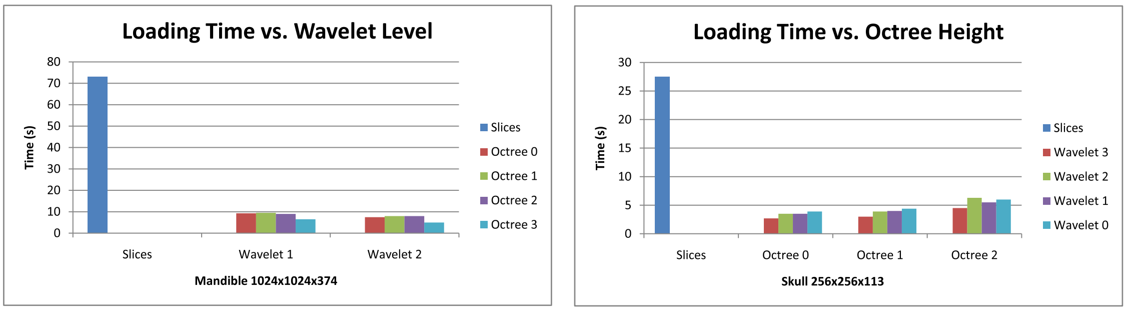
In the second stage, a tool called *Compress* computes the wavelet representation described in section 4.2.1.2 using the output of the previous stage. The depth of the octree and the maximum level of wavelet compression are passed as user-defined parameters. Table 4.1 illustrates the preprocessing times for various data sets with an octree depth of two and two levels of wavelet compression on a Core2Duo 2GHz laptop machine with 2GB RAM.

The time it takes each node to perform frustum clipping and refine the scene after interaction was measured for different octree depths and multiple data sets. Also, the wavelet level at which each tile displays a scene was monitored. A fixed perspective was used during measurements, i.e., the orientation and zoom level remained unchanged. Figure 4.12(a) shows that the average time it takes to clip and render a scene decreases with increasing octree depth and thereby increasing number of bricks (except one outlier for the brain data set). The overhead of loading unnecessary parts of the data set decreases because the clipped volumes better approximate



(a) (b)

Figure 4.12: Influence of octree depth on average rendering time and average level of detail: (a) average time measured to perform frustum clipping and volume assembling, (b) average percentage of original voxels displayed (including average wavelet level and variance).



(a) (b)

Figure 4.13: (a) Loading time vs. wavelet level, (b) Loading time vs. octree height *Image courtesy of Nguyen [Ngu08].*

the regions of the data set that tiles have to display. Figure 4.12(b) shows that at the same time, with increasing octree depth, the average level of detail over all tiles increases, i.e., the average wavelet level over all fifty tiles gets smaller, because the available texture memory is used more efficiently to store structural details of the volume. The detail level of the skull data set remains constant with increasing octree depth because the volume is small enough to fit entirely into the graphics card's texture memory. Note that the real-valued wavelet levels in figure 4.12(b) result from averaging the discrete values of each tile. Though these compression levels cannot be achieved in practice, they give a good theoretical value describing the overall level of detail comprised in all fifty LCD panels. Further note that the partly asymmetric variances depicted in figure 4.12(b) are due to the fact that the percentage of voxels scales non-linearly with the wavelet level, i.e., the level of detail comprised at level i is 1/8th of level $i - 1$. In general, both the average rendering time and the level of detail are positively affected by a deeper octree structure. It was empirically determined that an octree depth of four is sufficient when working with most data sets.

In earlier experiments, Nguyen [Ngu08] investigated the data structure's performance when accessing volume data in a non-distributed volume rendering application. Figure 4.13 illustrates the results and compares the performance of a desktop system using the octree-wavelet structure to a desktop system that neither uses spatial subdivision nor multi-resolution representation. The unoptimized system loads the entire data by directly accessing a set of files storing slice information, i.e., slices are loaded sequentially into main memory. Figure 4.13(a) compares the time it takes to load the entire mandible data set slice-wise into RAM to the time it takes to load the same volume at different wavelet levels for a fixed octree depth of three. Though the time difference observed for the system using the octree-wavelet structure is marginal between wavelet levels, loading is less than 7% of the average time measured for the cross-sectional data. Figure 4.13(b) illustrates the influence of octree depth on the loading time for the skull data set. It can be observed that loading takes longer with increasing octree depth, which is due to the increased number of I/O operations that have to be performed and are generally considered to be slow. This pattern also holds for various wavelet levels.

The results of figure 4.13(b) and figure 4.12(a) seem contradictory. On the one hand, a deeper octree leads to a better approximation of subvolumes and reduces the amount of data that needs to be loaded. On the other hand, the increased number of file operations works against this effect. For the data sets investigated, the benefit of having a smaller clipping volumes outweighs the negative effect of additional I/O operations. In the future, additional benchmarks are required to further quantify the opposing effects.

4.2.7 Discussion

The system presented in this section implements a static sort-first algorithm which reduces communication overhead and network traffic between nodes of the cluster to a minimum. This is important for the visualization of large data sets, as it prevents the network from being a performance bottleneck.

The out-of-core technique makes efficient use of the resources available on modern graphics cards. For GPU methods, the gap between RAM and VRAM mainly limits the size of data that can be visualized. By harnessing the combined rendering power of a large number of GPUs it is possible overcome this constraint and bridge the gap. The implementation was successfully tested on a tiled display comprised of 25 compute nodes driving 50 LCD panels. The system was able to produce renderings of volumetric data sets larger than the texture buffer size of a single graphics card at significantly higher levels of detail than on a single desktop display.

To some extent the combination of octree data structure and multi-resolution technique resembles scenegraph systems. Scenegraphs also perform frustum culling to reduce the complexity of scenes and they also implement level-of-detail methods to estimate an object's level of detail based on its distance to the viewing plane. However, scenegraphs are generally oriented towards geometric data and not suited for managing volumetric data sets.

Future work will focus on the implementation of prefetching strategies to compensate for small transformations of the data set and reduce the need of having to rerun the entire clipping and assembling process.

4.3 Fast Direct Volume Rendering Techniques for Distributed Display Clusters

Section 4.2 described a technique to deal with large-scale data sets when their size exceeds the available amount of texture memory of a render node's GPU. The solution presented was a combination of out-of-core frustum clipping and wavelet-based multi-resolution techniques that enabled the rendering of data sets using 3D texture slicing. Texture slicing is a built-in function of modern graphics cards and trilinear texel interpolation is natively implemented in hardware. However, the underlying sampling scheme is rather simple (see figure 4.2(b)) and potentially causes visual artifacts. Previously it was emphasized that texture slicing is not inherent to the out-of-core approach and can be replaced by more sophisticated rendering algorithms.

Today, ray casting is considered to be the state-of-the-art algorithm for direct volume rendering. With the advent of programmable shader units featuring branch and loop support, ray casting methods experienced resurgence on the GPU. When NVIDIA introduced the G80 architecture, ray casting began to outperform slicing and other rasterization-based methods for an equivalent number of samples.

This section concentrates on fast visualization techniques for large-scale data sets on high-resolution displays via ray casting as an alternative to 3D texture slicing. The research focus is on how to keep the application interactive and achieve the highest possible frame rates on each render node of the cluster. The methods are not solely restricted to display clusters and can also improve the performance of direct volume rendering systems in single display environments.

Three implementations are discussed, aimed at achieving interactive frame rates by exploiting hierarchies of bounding volumes as acceleration structures (see section 4.1.5.2). BVHs have been successfully used to render mesh data and isosurfaces, yielding impressive results in terms of performance. Though their usage looks promising, BVHs have hardly been employed in the context of direct volume rendering.

First, a GPU BVH implementation is described, followed by a hybrid CPU/GPU system. Then, a purely CPU-based implementation is presented. Benchmarks reveal that implementations strongly differ in the frame rates that can be achieved and that the size of data sets strongly influences performance.

4.3.1 BVH-Based Direct Volume Rendering on the GPU

Octrees and kd-trees are popular spatial acceleration structures that have been extensively used for ray tracing polygonal mesh data and isosurfaces. While first implemented on the CPU, systems that generate and traverse these data structures on the GPU have been presented recently [HL09, ZHWG08, GMGA08].

Wald et al. [WBS07] showed that BVHs are competitive with these data structures. They demonstrated how hierarchies of bounding volumes can be used for fast CPU ray tracing of static polygonal models when applying a careful tree construction, SSE programming, and ray packets. GPU ray tracers based on BVHs were presented, for example, by Gunther et al. [GPSS07] and Carr et al. [CHCH06] for non-volumetric data sets.

This section describes an experimental direct volume rendering system featuring BVH traversal on the GPU. First, the build of the data structure, which takes place on the CPU side, and the underlying storage scheme are described. Afterwards, the particularities of a stack-based traversal on the graphics hardware are discussed.

4.3.1.1 BVH Build & BVH Representation

The BVH is built upon the *implicit* BVH by Wald et al. [WFKH07b]. Instead of building a hierarchy of bounding volumes for a fixed scalar value, each node in the hierarchy stores the minimum and maximum of all values in the subtree. This allows for easy culling of subtrees during traversal. Also, the min-max values can serve as input parameters for optimization heuristics trying to minimize the traversal costs for the BVH (see section 4.3.3).

The BVH is built on the CPU and traversed on the GPU. Instead of storing the recursive structure as a tree using pointers to other tree nodes as one would do in a high-level programming language, the hierarchy is ”serialized” and stored as a one-dimensional array. An indexing scheme enumerates nodes during the build and assigns them to their respective position in the array. That way, the BVH can easily be stored as a 1D texture that is sent to the GPU. The build procedure constructs a binary tree, i.e., each node (except leaf nodes) has two child nodes. Furthermore, the bounding volumes consist of non-overlapping AABBs. Pseudo code illustrating how the BVH is built is given in listing 4.3:

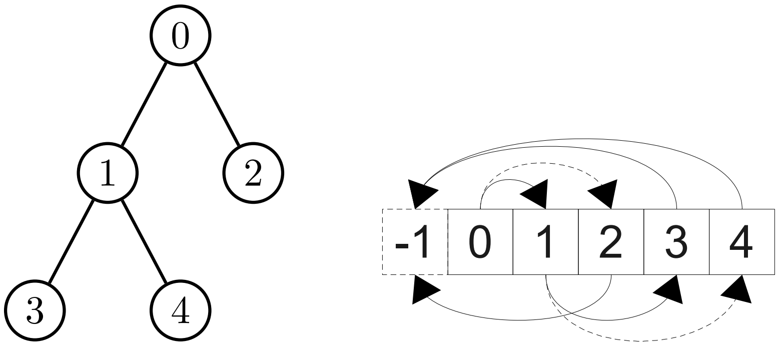


Figure 4.14: Storage scheme for BVHs.

Listing 4.3: BVH Build

```
void BVH::build()
{
    //allocate 1D array for Node elements
    nodes = new Node[dim[0]*dim[1]*dim[2]];

    //index of next free element in the array
    nextFree = 1;

    //top level AABB
    AABB root;
    root.setmin(0.0, 0.0, 0.0);
    root.setmax((float)dim[0]-1, (float)dim[1]-1,
                                 (float)dim[2]-1);

    //start recursive build
    build(0, root);
    //update node information, bottom-up
    update();
}

void BVH::build(int nodeID, const AABB &aabb)
{
    //store corner coordinates
    nodes[nodeID].setmin(aabb.getmin());
    nodes[nodeID].setmax(aabb.getmax());

    //create leaf node
    if (isLeaf(aabb))
    {
        nodes[nodeID].child = -1.0;
        return;
    }

    //index of left child
    nodes[nodeID].child = nextFree;
    nextFree += 2;

    //determine split axis (x/y/z)
    unsigned int splitDim = getSplitDim();
    //determine split position
    float splitPos = getSplitPos();

    //left child & right child AABB
    AABB left_child = aabb, right_child = aabb;
    left_child.max[splitDim] = splitVal;
    right_child.min[splitDim] = splitVal;

    //recurse
    build(child + 0, l_voxel);
    build(child + 1, r_voxel);
}
```

A one dimensional array `nodes` is allocated storing the elements of the hierarchy. Throughout the build, the variable `nextFree` indicates the position of the first free array element in `nodes`. Once an AABB for the root has been initialized, the recursive subdivision starts. `build(int, const AABB&)` checks if the current AABB should be further subdivided or stored as a leaf node. Leaf nodes are indicated by negative child indices, i.e., `nodes[nodeID].child = -1.0`, whereas inner nodes store the array index of their *left* child node, i.e., `nodes[nodeID].child = nextFree`. The index of the right child for any element in `nodes` is implicitly given by `nodes[nodeID].child + 1`. In case of an inner node, `splitDim` defines the axis along which the current AABB is split. `splitPos` indicates the exact splitting position along `splitDim`. After having determined how to split the current BVH node, `AABB` elements for the left and right child node are initialized and recursively subdivided.

Figure 4.14 illustrates how a hierarchy of bounding volumes is stored as a one-dimensional array of `Node` elements in order to be sent to the GPU. Note, regular arrows indicate actual left child indices. Dashed arrows refer to the right child indices, implicitly given by adding 1 to the index of a node's left child. Leaf nodes are indicated by setting a left child index of -1.

The performance gain that can be achieved with an BVH depends on the "quality" of the tree and the metric that is used during the build. The build metric determines how to split nodes and when to stop the recursive subdivision process. In listing 4.3, `getSplitDim()` and `getSplitPos()` are the two functions that control the split of a node. For polygonal data sets the construction is often based on the Surface Area Heuristic (SAH) [GS87], a greedy algorithm that tries to minimize the overall traversal costs for the BVH. However, since direct volume rendering does not deal with with geometric objects, it is difficult to apply SAH to volumetric data sets. Two build strategies have been tested for the BVH build:

1. *Median-split.* `getSplitDim()` returns and integer representing the axis with longest spatial dimension ($x = 1, y = 2, z = 3$). If multiple axis have the same dimension, the one with the smallest axis index is chosen. Afterwards, `getSplitPos()` calculates the splitting position along the splitting axis, which in case of median-split corresponds to the axis' center point. Besides the axis dimension and trimming operations to only enclose the smallest non-empty subvolume in a node, median-split does not take any characteristics of the data set into account when building the BVH hierarchy. The subdivision is

stopped and a leaf is generated when the node volume falls below a predefined threshold, for instance, 8^3 voxels.

2. *Variance-based.* The idea is to split nodes based on the variance they comprise. First, a bottom-up preprocessing step builds the *summed variance table* of the volume. The table is similar to the ”summed volume table” described by Vidal et al. [VMD08], but instead of accounting for non-empty voxels, the summed variance table records the maximum difference between two scalar values in a region of the data set, i.e., the maximum variance. The method tries to minimize the difference between these variances when determining the split position along the longest node axis. That way, the variance serves as an estimate of the expected workload for a particular split.

Empirical observations reveal that variance-based builds yield ”tighter” BVHs than median-split builds as the total number of nodes is generally lower.

The BVH is stored as a one-dimensional array of objects of type `Node`. The `Node` data type is depicted in listing 4.4:

Listing 4.4: BVH Node

```
struct Node
{
  //lower left corner
  unsigned int min_x;
  unsigned int min_y;
  unsigned int min_z;

  //index of left child
  int child;

  //upper right corner
  unsigned int max_x;
  unsigned int max_y;
  unsigned int max_z;

  //min-max values of node
  unsigned char min_w;
  unsigned char max_w;

  //padding bytes
  unsigned char pad0
  unsigned char pad1
};
```

Each `Node` element of the implicit BVH stores the coordinates of the lower left and upper right bounding box corners, a child index, and two entries representing the minimum and maximum scalar value in the subtree. Two padding bytes are added to make a total of 32 bytes (= 256 bits) per `Node` object. That way, instances of

`Node` can easily be stored in two entries of an `GL_RGBA32F_ARB` OpenGL texture buffer object, supporting 128 bits per color, i.e., 32 bits per color channel.

Listing 4.5: BVH Traversal

```

int stack[10];

void traverse_bvh()
{
  if (!ray_intersect(root))
      return;

  //next node to fetch
  int id = 1;
  //top of stack pointer
  int d=1;

  while(d > 0)
  {
      //get left and right child
      leftChild = getNode(id);
      rightCild = getNode(id + 1);

      //ray aabb intersection
      bool tmp1 = ray_intersect(leftChild);
      bool tmp2 = ray_intersect(rightChild);

      bool b1 = tmp1 & !empty(leftChild);
      bool b2 = tmp2 & !empty(rightChild);

      //ensure leftChild is closer
      if(closer(rightChild, leftChild))
      {
          swap(leftChild, rightChild);
          swap(b1, b2);
      }

      //hit, not emtpy and leaf
      if (b1.x & leftChild.child < 0)
      {
          raycast(leftChild);
      }
      //hit, not emtpy and leaf
      if (b2 & rightChild.child < 0)
      {
          raycast(rightChild);
      }

      b1 &= (leftChild.child >= 0);
      b2 &= (rightChild.child >= 0);

      //both are inner nodes, hit, and not empty
      if (b1 & b2)
      {
          id = leftChild.child;
          stack[d++] = rightChild.child;
      }
      else
      {
          //inner node, hit and not empty (leftChild)
          if(b1)
          {
              id = leftChild.child;
          }
          //inner node, hit and not empty (rightCild)
          else if (b2)
          {
              id = rightChild.child;
          }
          //get next node from stack
          else
          {
              id = stack[--d];
          }
      }
  }
}
```

4.3.1.2 BVH Traversal

This section describes the traversal of the BVH on the GPU. The tree is traversed *top-down* and uses a *stack* to record nodes which should be returned to at a later point. Pseudo code portraying the traversal is given in listing 4.5. The actual traversal is implemented in the fragment shader of the GPU and executed once per pixel.

In line 6, the root of the BVH is tested for intersection with the current viewing ray. If the ray already misses the top level bounding box nothing more needs to be done. Otherwise, `id` denotes the array index of the next node to visit. The variable `d` indicates the top element of the stack. The traversal method always fetches the left and right child of a node at the same time and tests them for intersection (line 17-22). Empty regions that do not contribute to a pixel's value are tested and later discarded (lines 24- 25). Line 28-32 ensures the closer node is always handled first and `leftChild` and `rightChild` are swapped if necessary. This is important, since volume samples are blended in front-to-back order, which allows for early ray termination. Afterwards, in line 35-71, the nodes are treated according to the results of the previous tests (empty/non-empty, inner node/leaf node, hit/missed by ray) and either ray casted as described in section 4.1.4.2, pushed onto the stack, or discarded. The traversal exists the main loop when no more nodes are on the stack.

4.3.2 BHV-Based Direct Volume Rendering on a Hybrid CPU/GPU System

The streaming processors of the GPU are extremely fast when performing brute-force SIMD operations on chunks of data. However, a strongly branching and unpredictable program flow can cause pipeline stalls and decrease performance significantly. When traversing a tree-like acceleration structure on the GPU and testing for ray-node intersections, diverging rays can lead to an unsteady program flow, since rays might descend into different branches of the tree. While this is a big problem on the GPU, the CPU can better handle such situations due to a different hardware architecture. To address this issue, a hybrid CPU/GPU system is presented that traverses the BVH on the CPU and ray casts the volume on the GPU.

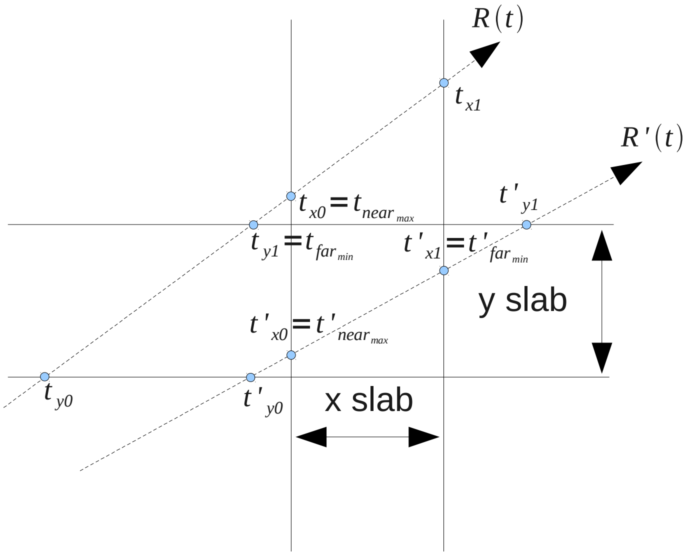


Figure 4.15: Slab intersection test in two dimensions. Since $t_{near_{max}} > t_{far_{min}}$, $R(t)$ misses the boudning box. $R'(t)$ hits the bounding box and $t'_{near_{max}} \leq t'_{far_{min}}$ holds.

The idea is to employ the *coherent ray packet* algorithm by Wald et al. [WBS07] to traverse the BVH on the CPU and compute the intersection parameters of the viewing rays with the leaf nodes of the data structure. Instead of looking at one ray at a time, coherent ray casting works on *packets* of rays that are traced through the volume. When using SSE operations that support a vector width of four on x86 CPU architectures, four rays of a packet, a so-called *packlet*, can be tested and propagated in parallel. Coherent traversal yields a list of intersection parameters for each ray of a packet that describe the intersection points of viewing rays with the leaf nodes. The parameters are sent to the GPU, where they are used to ray cast the respective nodes of the BVH. The following section describes how to obtain the intersection parameters, how traverse the BVH using ray packets, and how the CPU-to-GPU transfer is realized.

4.3.2.1 Obtaining the Intersection Parameters

The intersection parameters for the BVH nodes are obtained by applying a ray-bounding box intersection test as described by Kay et al. [KK86]. The method tests for intersections and returns the parametric t values of the entry and exit point, if existing. The test treats bounding boxes as the intersection of three *slabs*, where a slab is the space between two parallel planes. The method looks at the intersection of each pair of slabs by the ray. It calculates t_{far} and t_{near} for each pair of slabs. If the largest $t_{near_{max}}$ value is greater than the smallest $t_{far_{min}}$ value the ray misses the box, otherwise it hits the box and the entry and exit point are given by those two t values. Figure 4.15 illustrates the basic principle of the slab intersection test for two dimensions.

4.3.2.2 Coherent Ray Casting

The ray-node intersection parameters are obtained by applying a coherent traversal algorithm based on ray packets. Coherent traversal considers groups of rays that are represented and traversed as a packet (typically 8×8 or 16×16 rays per packet). The workload for a group of rays is amortized by performing the common traversal steps for the entire packet. Implicit BVH packet traversal has been introduced by Wald et al. [WBS07, WFKH07a] for isosurface and triangle mesh ray tracing. The following summarizes the core features of the coherent packet algorithm, which are further described and extended in section 4.3.3:

Empty space culling. Empty subtrees that do not contribute to a pixel's final color are discarded and marked as *inactive*.

Speculative first active descent test. If the first *active* ray of a packet intersects the current node's bounding box the algorithm descends into the subtree without checking the other rays of the packet (see figure 4.16(a)).

Frustum test. If the first active descent test returns a negative result the algorithm knows that the packet at least partially misses the bounding box. Intersecting the packet's frustum with the box lets the algorithm determine if the entire packet misses (see figure 4.16(b)).

First active ray tracking. If the two previous tests fail, all remaining rays of the packet are tested until the first active ray is found that intersects the box. By tracking the index of the first active ray, the previous rays that miss the box are marked as inactive because their ray index is smaller than the one of the first active ray (see figure 4.16(c)). If no active ray in the packet intersects the box, the node is rejected and the next one is popped from the stack.

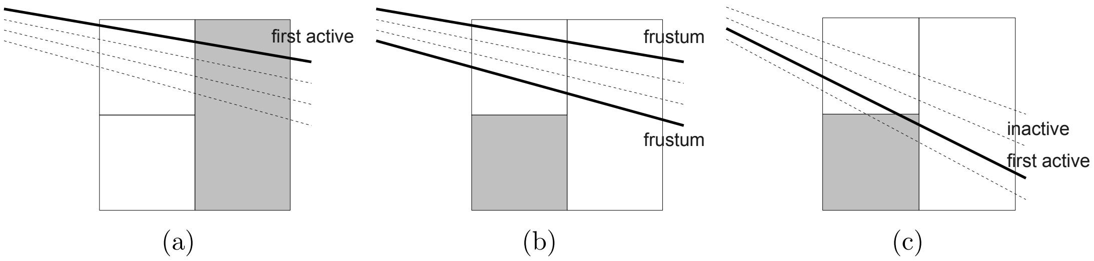


Figure 4.16: Coherent ray packet traversal: (a) speculative first active descent, (b) frustum test, (c) first active ray tracking.

Leaf traversal. When the active rays of a packet hit a leaf of the BVH, the t intervals are stored as entries in a list of intersection parameters that is maintained for each ray of the packet.

Though SIMD is not inherent to the packet approach, a consistent usage of SSE operations can speed up the BVH traversal, as with SSE it is possible to either process four rays at a time or to process all dimensions of a ray simultaneously.

4.3.2.3 CPU to GPU Transfer

After the BVH has been traversed on the CPU, the t values that describe the intersection points of rays with the BVH leaves are sent to the GPU. Pairs of t values define intervals $[t_{near}, t_{far}]$ along a ray, which are ray casted and blended in front-to-back order in the fragment shader as described in listing 4.1. Two one-dimensional arrays are transferred to the GPU using the OpenGL framebuffer object extension:

1. `tintervals`. The array lists *all* t values of *all* rays in successive order. Its layout is $\underbrace{t^0[0], t^0[1], \ldots, t^0[n-1], t^0[n]}_{\text{ray } 0}, \ldots, \underbrace{t^m[0], t^m[1], \ldots, t^m[n'-1], t^m[n']}_{\text{ray m}}$. If for a particular ray an entry point $t^i[j]$ equals the last exit point $t^i[j-1]$, the entry point is discarded and the negated exit point $-t^i[j-1]$ is stored in order to keep the number of entries as small as possible.

2. `tintervals_indices`. For each ray `tintervals_indices` stores a pointer $T[i] = d$ into `tintervals`, indicating the position of the first t value for ray i

is at index d. Thus, the number of entries in `tintervals_indices` equals the total number of rays.

4.3.3 BVH-Based Direct Volume Rendering on the CPU

Depending on the amount of data that needs to be sent at each frame, the CPU-to-GPU transfer can be a bottleneck in hybrid systems. Nevertheless, coherent traversal of BVH structures using ray packets on the CPU is a promising approach for achieving high frame rates. In order to deal with possible bottlenecks caused by slow CPU-to-GPU transfers, this section presents a pure CPU implementation of a direct volume rendering system using BVHs. The implementation is an extension of the coherent BVH traversal algorithm by Wald et al. [WBS07], which has already been sketched in section 4.3.2.2, but is described in more detail subsequently.

4.3.3.1 Optimized Coherent BVH Traversal

Again, the idea of the algorithm is to traverse a group, or packet, of rays simultaneously through the BVH in order to amortize the per-ray cost of the traversal. As before, the algorithm descends the BVH performing a speculative first active hit test. In case of a miss, a conservative frustum test is employed in order to determine if the entire packet misses the node. Eventually, the algorithm yields an interval of rays (if existing) that intersect each BVH node. Intersection tests are performed in groups of four rays at a time (*packlets*) using SIMD/SSE instructions.
The algorithm keeps track of the index of the first active packlet on the traversal stack. It advances this index to the next hit and back-tracks when all packlets (rays) in the packet miss. When a leaf node is reached, all active packlets starting with the first-active are intersected against the leaf node. The ray-leaf bounding box test yields the entry and exit points required for the ray casting algorithm.

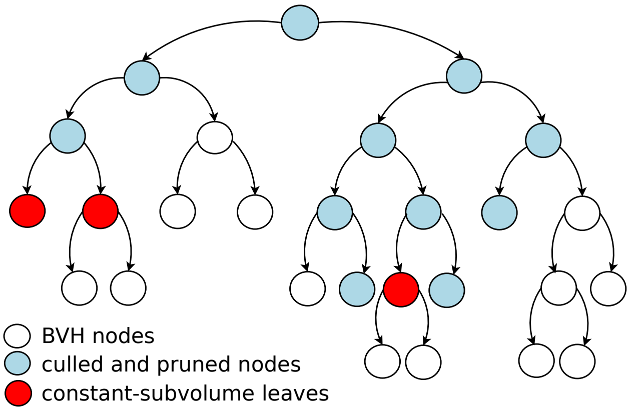


Figure 4.17: Culling and pruning metrics based on the pre-integrated transfer function yield significantly smaller subtrees. A similar metric is used to inexpensively integrate constant subvolumes when possible.

Listing 4.6: Optimized Traversal

```
void traverse(Node∗ nodes, RayPacket& packet){
  //BVH node index
  int id = 0;
  //first active SSE packlet in the packet
  int first_active_packlet = 0;
  //BVH stack
  int stack[32];
  //stack for recalling first−active packlet
  int fa_stack[32];
  int d=0;

  while(true){
    Node& node = nodes[id];
    //speculative min−max tree descent
    while(true){
      //child is empty, i.e., leaf
      if (node.child == 0) break;
      if (node_is_empty(nodes[node.child + 0])
        { id = child + 1; continue; }
      if (node_is_empty(nodes[node.child + 1])
        { id = child + 0; continue; }
      break;
    }
    //speculative first−active traversal
    int first_active_packlet =
        first_that_intersects(packet, nodes[id]);
    if (first_active_packlet <
        RayPacket::MAX_PACKLETS){
      //if any packlet hit
      bool csv = constant_subvolume(node);
      if (node.child && !node_is_leaf(node) && !csv){
        //interior
        int front_child = closest_child(node, packet);
        stack[d] = node.child + 1 − front_child;
        fa_stack[d] = first_active_packlet;
        id = node.child + front_child;
        d++;
        continue;
      }
      else if (node.child){
        //leaf
        if (csv)
          dvr_constant(node, packet);
        else
          dvr(node, packet)
      }
    }
    if (d==0) return;
    d−−;
    id = stack[d];
    first_active_packlet = fa_stack[d];
  }
}
```

Speculative descent into subtrees of the acceleration structure is based on the min-max values stored with the nodes of the implicit BVH. In order to test if a node has a range of scalar values overlapping the transfer function domain, a metric based on the preintegrated transfer function is employed [EKE01]. Likewise, similar metrics serve in determining whether BVH nodes should be pruned or treated as empty. Pruning, i.e., the premature treatment of an inner node as a leaf, can significantly reduce the traversal costs.

Furthermore, if a leaf represents an (almost) constant subvolume, an alternative fast constant subvolume integration routine can be employed in order to avoid the standard per-voxel integration routine.

Pruning and constant subvolume integration are not part of the original coherent ray packet technique [WBS07]. Figure 4.17 illustrates the optimized BVH traversal algorithm, for which C++ pseudo code is given in listing 4.6.

4.3.3.2 Empty Space Skipping

Similar to determining whether an isosurface lies between the min-max values of a node, it is possible to check if a transfer function generates nonzero opacity values for any scalar field value in the min-max range. The required information is encoded in the lookup table of the preintegrated transfer function. Preintegration approximates the volume rendering integral in equation 4.3 by assuming a piecewise linear scalar field along the viewing ray. The method computes a lookup table to approximate the contribution of the ith segment defined by the scalar value f_a at the start (front) of the segment and f_b at the end (back). In other words, for each pair of scalar values (f_a, f_b), the preintegrated lookup table contains an RGBA value describing the expected contribution of that segment. In order to evaluate `node_is_empty()` in listing 4.6, the algorithm checks

$$\rho_\alpha(f_{min}, f_{max}) > \delta_c, \tag{4.15}$$

where ρ_α is the alpha component of the preintegrated transfer function over f_{min}, f_{max}, and δ_c is a threshold below which to cull ($\delta_c <$ 1e-3 works well for most situations).

For the mathematical theory underlying preintegrated transfer functions the reader is referred to the original paper of Engel et al. [EKE01].

4.3.3.3 Pruning Heuristic

Transfer functions can map regions of low frequency in the scalar value domain to regions of high frequency in the color and opacity domain, and vice-versa. In regions of low frequency it is convenient to use larger bounding boxes, as early termination is less likely and intersection tests are potentially redundant. Performing redundant intersection tests increases the overall workload. On the other hand, it is desired to fully traverse the BVH in regions of high frequency and subdivide the acceleration structure as far as possible. Pruning based on characteristics of the transfer function provides a way to address these opposing needs and reduce the number redundant packet-box intersections.
Again, the frequency comprised in a subtree can be estimated using the preintegrated transfer function opacity ρ_α over the min-max interval of a node. The estimation yields an upper bound on the opacity over space contributed by that node. This spatially invariant metric is normalized by the diagonal ratio of the node's bounding box to that of the entire volume, i.e.,

$$\rho_\alpha(f_{min}, f_{max})\frac{|\vec{D}_{volume}|}{|\vec{D}_{box}|} > \delta_p. \tag{4.16}$$

Equation 4.16 is used by `node_is_leaf()` in listing 4.6 to determine whether a given node should be treated as a leaf or interior node. $\delta_p = 1.5$ works well as a default value for most data sets.

4.3.3.4 Constant Subvolume Heuristic

A further metric also based on the preintegrated lookup table can be employed to determine if a region of the data set comprises a sufficiently low variance in order to be treated as a constant block. The benefit of treating nodes of the BVH as constant blocks is the fact that the respective subvolume can be integrated using a far less expensive routine, with neither per voxel lookup nor interpolation. If applicable, constant block integration leads to a significant speed-up in the respective area. Based on the evaluation of the following heuristic in `constant_subvolume()` (see listing 4.6), constant block integration is used if the heuristic succeeds and standard per voxel sampling if it fails:

$$\begin{aligned}
&\rho_{ll} = \rho_{\alpha}(f_{min}, f_{min}) \\
&\rho_{hh} = \rho_{\alpha}(f_{max}, f_{max}) \\
&\rho_{lh} = \rho_{\alpha}(f_{min}, f_{max}) \\
&sup\{|\rho_{lh} - \rho_{ll}|, |\rho_{lh} - \rho_{hh}|\} < \delta_{sv}. \qquad (4.17)
\end{aligned}$$

As before, ρ_{α} denotes the preintegrated opacity values. Experiments show $\delta_{sv} <$ 1e-4 consistently produces good results without noticeably removing visible features.

By consistently applying pruning, culling and constant subvolume optimizations, the traversal of the BVH tree can be accelerated in regions of empty or homogeneous space. Since the underlying metrics are based on the preintegrated transfer function and not on the structure of the implicit BVH, on-the-fly changes in classification can be applied without having to recompute the acceleration structure.

4.3.4 Results

All three BVH systems were benchmarked in order to compare their performance. To draw conclusions about the scalability of each system, the test suite contained a small-sized (heptane), a medium-sized (zebrafish), and a large-sized (Richtmyer-Meshkov intability (RMI)) data set. Figure 4.18 and table 4.2 provide details about the test suite.

The hardware platforms used for the benchmarks were: (1) a 2.67 GHz Core i7 desktop with 8 GB RAM (4 physical, 8 virtual cores) and NVIDIA 285 GTX GPU (1.5 GB VRAM, 240 cores), (2) a 3.0 GHz Core 2 Duo desktop with 2.7 GB RAM and NVIDIA 295 GTX GPU (896 MB VRAM, 480 cores). For consistency, all benchmarks rendered into a 512^2 frame buffer. To better gauge performance, a brute-force GPU ray caster served as a reference system.

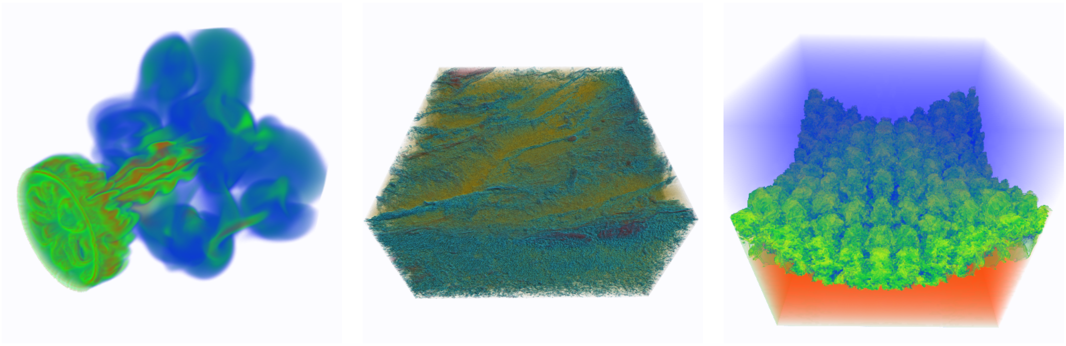

(a) (b) (c)

Figure 4.18: Benchmarking data sets: (a) heptane, (b) zebrafish, (c) Richtmyer–Meshkov instability (RMI).

Data Set	**Dimensions** $(x \times y \times z)$	**Size**(MB)
Heptane	$302 \times 302 \times 302$	28.5
Zebrafish	$900 \times 500 \times 910$	390
Richtmyer Meshkov	$1,024 \times 1,024 \times 1,024$	1,024

Table 4.2: Data sets of the benchmark test suite.

Table 4.3 lists the benchmark results. In all cases, the BVH was built using a median-split strategy generating leaf nodes if the bounding box volume was less than 64^3 voxels.

A general observation is that CPU and GPU volume renderers show different performance behavior for different data sizes. While GPU renderers outperform the CPU for small and medium data sets, the results are reversed for large data sets, such as the Richtmyer Meshkov instability. Despite all optimizations, the GPU BVH and the hybrid BVH renderer perform worse than the brute-force GPU reference system. The issue of the GPU BVH implementation is that the traversal of the acceleration structure does not pay off until it comes to large-sized data sets. The only situation where the system outperforms the brute-force approach is when rendering the Richtmyer Meshkov instability. However, even for this data set the speed-up is marginal. In all other cases the performance is about half the performance of the reference system. One reason for this is the fact that the GPU is extremely good at performing SIMD operations that keep the highly parallel graphics pipeline busy. BVH traversal, however, can cause pipeline stalls, as some rays might still

Data set	**BVH**	**GPU BVH (fps)**		**Hybrid BVH (fps)**	**CPU BVH (fps)**	**Brute-force (fps)**
	#nodes	Core 2 Duo 295GTX		Core 2 Duo 295GTX	Core i7 285GTX	Core i7 285GTX
		stack[16]	stack[20]			
Heptane	255	36.8	18.9	11.2	13.5	**86.2**
Zebrafish	4095	16.4	9.2	10.6	5.3	**29.1**
RMI	8191	1.7	2.5	2.6	**18.8**	1.47

Table 4.3: Benchmark results.

descend into different branches of tree while others have already terminated. Also, there is indication that high register pressure on global GPU memory is decreasing performance. Accessing global memory on the GPU is known to be slow, however, the stack used for traversal resides in global memory. Table 4.3 shows that there is a sweep point between `stack[16]` and `stack[20]`, resulting in a rapid drop of performance. Still, a minimum stack size is required to traverse the acceleration tree. An alternative approach to lower register pressure on global GPU memory would be a stack-less traversal algorithm similar to the one presented by Hughes et al. [HL09] for kd-trees.

The intention of the hybrid system was to shift the complex calculations of the BVH traversal to the CPU in order to lower the overall GPU workload and the pressure on global memory caused the stack-based traversal. The GPU should perform at what it is best at and run aggressive brute-force SIMD operations to ray cast the volume between the intersection points calculated on the CPU. As it turns out, the bottleneck of the hybrid approach is the CPU-to-GPU transfer. Transferring the t values obtained from the intersection tests to GPU memory was implemented using the OpenGL frame buffer object (fbo) extension. Though fbos are specifically designed for this purpose, the latency of per-frame CPU-to-GPU transfers is too high. A possible solution is to generate smaller tree structures that yield smaller lists of t intervals and reduce the CPU-to-GPU bus traffic by applying culling, pruning, and constant subvolume integration concepts as implemented in the CPU system.

Although the CPU BVH approach is outperformed by GPU methods on low-resolution data, it is highly competitive and even faster by over an order of magnitude on large data. With a sufficiently powerful multicore machine, such as the 8 core workstation used for the benchmarks, large data can be rendered interactively without level of detail methods, eliminating the need for progressive rendering and improving interaction. The multilevel cache hierarchy that is implemented in hardware significantly reduces latencies when accessing voxels of data sets that reside in main memory. On the GPU, however, efficient memory management is crucial for large data sets, and the flat cache hierarchy demands for out-of-core techniques like the one described in section 4.2.

4.3.5 Discussion

This section focused on fast visualization methods for direct volume rendering on large high-resolution displays. Three implementations based on BVHs were presented. BVHs have been successfully exploited as acceleration structures for isosurface visualization and polygonal mesh rendering. Investigating their potential for direct volume rendering was one of the goals of this section.

The first system implemented a stack-based BVH traversal on the GPU. Results show the architecture of the graphics hardware poses major limitations when trying to port existing CPU concepts to the GPU: A compact representation of the acceleration structure is required that the GPU can deal with. Due to the lack of higher data structures, this representation is usually different from CPU implementations of the same data structure. Furthermore, resources are precious and a wasteful use decreases performance significantly. Also, the highly parallel streaming architecture of modern GPUs demands for a careful algorithm design.

The second system tried to exploit the benefits of a close CPU/GPU collaboration. Shifting the traversal of the tree structure from the GPU to the CPU enabled the use of coherent ray packets to determine the intersection points with the nodes of an implicit BVH, storing the min-max values of a subtree in each node of the hierarchy. The issue with the implementation was the per-frame CPU-to-GPU transfer mechanism. High bus latencies prevented the system from being interactive. In the future, integrating optimization metrics, such as the ones implemented in the CPU renderer, might yield more compact tree structures and decrease the amount of data that needs to be transfered.

The third volume renderer also traversed the BVH using coherent ray packets. However, the implementation was fully restricted to the CPU and rigorously exploited SSE operations and various optimization metrics to speed-up the traversal of the acceleration structure and the ray casting process. Benchmarks show the system performs particularly well on high-resolution data and outperforms GPU renderers by over an order of magnitude. One reason is the existence of an efficient cache hierarchy built-in hardware. Another reason is that ray packets yield to an amortization of the per-ray costs during traversal.

Generally, when dealing with small and medium data sets, the GPU performs clearly better than the CPU. In combination with out-of-core techniques on large high-resolution displays (section 4.2) GPU methods seem to be the way to go. However, the section showed that implementing efficient optimization techniques on the GPU is challenging. If out-of-core techniques are not desired, BVHs and coherent ray packets are superior on the CPU for large data sets.

4.4 Conclusions

Chapter 4 focused on the development of direct volume rendering techniques for large high-resolution displays. When implementing a distributed volume rendering system, two major challenges arise: (1) the size of data sets often exceeds the computational resources of an individual cluster node and therefore requires special data handling techniques, (2) fast visualization algorithms are required to keep the system interactive. The chapter proposed solutions to both issues.

Section 4.2 introduced an out-of-core technique for dealing with gigabyte-sized data sets on tiled display clusters. When visualizing volume data using GPU-based visualization algorithms, the available amount of texture memory mainly limits the size of data sets that can be rendered interactively. The approach addressed this fact by employing a data structure that combines octree-based space subdivision and wavelet-based multi-resolution data representation. It could be shown that the technique leads to an improvement of render results, as resources are harnessed more effectively. The approach was successfully tested on a fifty tile display cluster and was able to produce renderings of volumetric data sets larger than the texture buffer size of a single graphics card at significantly higher levels of detail than on a single desktop display.

Section 4.3 presented various methods for direct volume rendering via ray casting to achieve interactive frame rates on large displays. All systems used acceleration structures based on hierarchies of bounding volumes. BVHs have been successfully employed for isosurface rendering and polygonal mesh visualization but hardly for direct volume rendering. The section showed that the implementation of efficient acceleration structures on the GPU is challenging. Often, 1 : 1 portings of CPU algorithms do not perform well on GPUs, which is mainly due to the GPU's highly parallel streaming architecture that demands for alternative programming paradigms. Good results for large data sets were achieved with a BVH-based CPU volume renderer. The system implemented an efficient traversal routine exploiting metrics for coherent ray packet traversal and SSE operations. While the system could not compete with state-of-the art GPU implementations for small and medium data sets, it was superior when visualizing data sets that could not entirely reside in the GPU VRAM, which is mostly the case for large high-resolution displays.
The observations give rise to the question how to design future visualization systems for large high-resolution displays: either as GPU-optimized systems using level-of-detail methods and out-of-core techniques, or as CPU-based systems that rely on natively built in cache hierarchies to tackle large-scale data sets. Giving a final answer to this question is outside the scope of this thesis, however, possible solutions have been proposed and may open up new alleys for additional research.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The overall objective of this thesis was the development of new visualization and interaction techniques for large high-resolution displays. Contributions have been made throughout all stages of the large display setup process.

AnyScreen was introduced as a lightweight rendering library enabling the implementation of distributed applications on large display clusters. Being highly flexible and adaptive, *AnyScreen* was able to drive arbitrary display configurations and support a variety of different stereo modes.

Tiled++ was introduced as a hybrid display setup to address to bezel problem in multi-monitor configurations. Though based on a straight forward idea, the user study revealed clear improvements of performance in static navigation tasks and a consistently higher degree of satisfaction with the quality of renderings on the user side.

A tag-based interaction approach was introduced in which camera-enabled cell phones acted as universal input/output devices for high-resolution displays. By giving each user a separate interaction device, the approach scaled both with the size of the display and the size of the user group. Furthermore, it was possible to support various group-specific collaboration concepts, such as handling user profiles, customizing the presentation of information, and dealing with multiple levels of access privileges. Two applications were presented that successfully implemented the interaction approach on a fifty tile display cluster.

On the visualization side this work focused on data management techniques and fast rendering algorithms for direct volume rendering on large high-resolutions displays. An out-of-core strategy was introduced that enabled the rendering of volumetric data sets larger than the texture buffer size of a single graphics card. It could be demonstrated that renderings on a fifty tile display cluster comprised a significantly higher level of detail than similar renderings on a single monitor system. Furthermore, the thesis investigated the potential of BVH acceleration structures for direct volume rendering. Insights on platform-specific implementation challenges were obtained from a GPU, CPU, and hybrid CPU/GPU volume rendering system. For large-scale data sets the CPU implementation using coherent ray packages was highly competitive with state-of-the art GPU systems. On the contrary, the GPU outperformed the CPU for rendering of small and medium data sets.

The implementation of all these techniques has shown that new interaction methods are needed, and the ones that were implemented turned out to improve usability of a large-scale display system significantly. It was also demonstrated that interactive frame rates can be achieved by incorporating state-of-the-art hardware-accelerated, texture-based volume rendering, octree space-subdivision, wavelet decomposition techniques, or SSE-optimized, CPU-based volume ray casting using coherent ray packets. Addressing previous shortcomings, the combination of suitable interaction techniques for large displays with high-performance rendering algorithms has been shown to work well in scalable, parallel computing environments and in multi-user environments.

5.2 Future Work

Tough important advances have been made in the last couple of years, large high-resolution displays still offer many interesting topics for future research. Concerning possible directions for additional studies, the general road map is outlined by the top ten challenges formulated by Ni et al. [NSS⁺06]. In particular, one might consider integrating latest display technologies into the large display setup, such as 3D screens and touch displays. Both technologies are currently gaining popularity not only in the academic field, but also in the home entertainment sector, offering exiting research opportunities especially in human-computer interaction. Likewise, new controller-free user interfaces, similar to Microsoft's project Natal lately presented for XBox 360 [MSN], could help to make the large display experience more intuitive. In summary, new input device and display technologies will open a wide range of new possibilities, making interactive large-scale display systems more ubiquitous in the near future.

Appendix A

Analysis of Variance (ANOVA)

Analysis of Variance (ANOVA) is a statistical method to test the *null hypothesis* that all a means of a dependent variable in a measurement series are equal, i.e.,

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_a.$$

An ANOVA compares two estimates of the variance σ^2 (denoting the variance within each of the a measurement series). The first estimate is based on the variances within samples and called *Mean Square Error* (MSE). The second estimate is based on the variance of the sample means and called *Mean Square Between* (MSB). However, MSB is only an estimate of σ^2 if H_0 is true. If H_0 is false, MSB is something larger than σ^2. The logic of the ANOVA test is as follows:
If H_0 is true, MSB and MSE are about equal, thus $F = \frac{MSB}{MSE} \approx 1$. If H_0 is false, then $F = \frac{MSB}{MSE} > 1$.
In order to estimate MSE, one has to calculate

$$MSE = \frac{\sum s_i^2}{a},$$

with s_i^2 denoting the sample variance of the ith measurement series. MSB is obtained by

$$MSB = Ns_M^2,$$

where N is the number of samples in each group and s_M^2 is the variance of the sample means.

In order to conduct a *significance test*, one must know the sampling distribution of F given that the null hypothesis is true. From the sampling distribution, the probability of obtaining an F as large or larger than the one calculated can be determined. This probability is the *probability value* $\mathcal{F}_{\beta,\gamma}(F)$ and depends on two degree of freedom parameters $\beta = a - 1$ and $\gamma = N - a$. If the probability value is lower than a given *significance level* α, i.e., $\mathcal{F}_{\beta,\gamma}(F) < \alpha$, H_0 can be rejected. Often, $\alpha = 0.05$ is chosen as a significance level for ANOVA.

If H_0 is rejected there are significant differences between the tested means. However, ANOVA does not provide information about which specific means caused the test to fail. In order to determine which measurement series significantly differ from others in respect to the mean, a *post-hoc* test is performed in the second stage of the ANOVA if H_0 is rejected. *Tukey's Honestly Significant Difference* (HSD) test compares the means of each series to the means of every other series and computes

$$t_{i,j} = \frac{\mu_i - \mu_j}{\sqrt{\frac{MSE}{N}}}.$$

A critical value t_c for $t_{i,j}$ can be determined from the *studentized range distribution*. If $t_{i,j} > t_c$ for a pair of means their difference is significant. For a more detailed introduction see Shupe et al. [Shu95].

Bibliography

[3dc] 3Dconnexion (accessed June 28, 2010). `http://www.3dconnexion.com/`.

[5DT] 5DT Fifth Dimension Technologies (accessed June 29, 2010). `http://www.5dt.com/index.html`.

[All] The Allen Brain Atlas (accessed July 7, 2010). `http://www.brain-map.org/`.

[ani] Animazoo - The Future of Motion Capture (accessed June 29, 2010). `http://www.saitek.com/`.

[Arg] Argonne National Laboratory. MPICH2: High-performance and Widely Portable MPI implementation (accessed May 25, 2010). `http://www.mcs.anl.gov/research/projects/mpich2/`.

[BBRS06] Rafael Ballagas, Jan Borchers, Michael Rohs, and Jennifer G. Sheridan. The Smart Phone: A Ubiquitous Input Device. *IEEE Pervasive Computing*, 5(1):70, 2006.

[BCHG04] Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Robert Gruen. Mouse ether: accelerating the acquisition of targets across multi-monitor displays. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1379–1382, New York, NY, USA, 2004. ACM.

[BGMB07] Jeremy P. Birnholtz, Tovi Grossman, Clarissa Mak, and Ravin Balakrishnan. An exploratory study of input configuration and group process in a negotiation task using a large display. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 91–100, New York, NY, USA, 2007. ACM.

[BGS01] Patrick Baudisch, Nathaniel Good, and Paul Stewart. Focus plus context screens: combining display technology with visualization techniques. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 31–40, 2001.

[BJB09] Sebastian Boring, Marko Jurmu, and Andreas Butz. Scroll, tilt or move it: using mobile phones to continuously control pointers on large public displays. In *OZCHI '09: Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group*, pages 161–168, New York, NY, USA, 2009. ACM.

[BJH^{+}01] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, page 89, 2001.

[BN05a] Robert Ball and Chris North. An analysis of user behavior on high-resolution tiled displays. In *In Interact 2005 Tenth IFIP TC13 International Conference on Human-Computer Interaction*, pages 350–363. Springer, 2005.

[BN05b] Robert Ball and Chris North. Effects of tiled high-resolution display on basic visualization and navigation tasks. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1196–1199, New York, NY, USA, 2005. ACM.

[BN05c] Robert Ball and Chris North. Effects of Tiled High-Resolution Displays on Basic Visualization and Navigation Tasks. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1196–1199, New York, NY, USA, 2005. ACM.

[BN08] Robert Ball and Chris North. The Effects of Peripheral Vision and Physical Navigation on Large Scale Visualization. In *GI '08: Proceedings of graphics interface 2008*, pages 9–16, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[BRS05] Rafael Ballagas, Michael Rohs, and Jennifer G. Sheridan. Sweep and point and shoot: phonecam-based interactions for large public displays. In *CHI '05: CHI '05 extended abstracts on Human Factors in Computing Systems*, pages 1200–1203, New York, NY, USA, 2005. ACM.

[BVC^{+}05] Robert Ball, Michael Varghese, Bill Carstensen, E. Dana Cox, Chris Fierer, Matthew Peterson, and Chris North. Evaluating the Benefits of Tiled Displays for Navigating Maps. In *IASTED International Conference on Human-Computer Interaction*, 2005.

[CCF94] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, New York, NY, USA, 1994. ACM.

[CCF+01] Han Chen, Yuqun Chen, Adam Finkelstein, Thomas Funkhouser, Kai Li, Zhiyan Liu, Rudrajit Samanta, and Grant Wallace. Data Distribution Strategies for High-Resolution Displays. *Computers & Graphics*, 25:811–818, 2001.

[CCL+01] Han Chen, Douglas W. Clark, Zhiyan Liu, Grant Wallace, Kai Li, and Yuqun Chen. Software Environments For Cluster-Based Display Systems. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 202, Washington, DC, USA, 2001. IEEE Computer Society.

[CDSY98] A. R. Calderbank, Ingrid Daubechies, Wim Sweldens, and Boon-Lock Yeo. Wavelet transforms that map integers to integers. *Appl. Comput. Harmon. Anal.*, 5(3):332–369, 1998.

[CG] The NVIDIA Geometry Shader Specification (accessed May 2, 2010). `http://developer.download.nvidia.com/opengl/specs/GL\_EXT\_geometry\_shader4.txt`.

[CHCH06] Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. Fast GPU ray tracing of dynamic meshes using geometry images. In *GI '06: Proceedings of Graphics Interface 2006*, pages 203–209, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[CN94] Timothy J. Cullip and Ulrich Neumann. Accelerating Volume Reconstruction With 3D Texture Hardware. Technical report, Chapel Hill, NC, USA, 1994.

[CSR+03] Mary Czerwinski, Greg Smith, Tim Regan, Brian Meyers, and Gary Starkweather. Toward characterizing the productivity benefits of very large displays. In *Proc. Interact*, pages 9–16. Press, 2003.

[DK] Kai-Uwe Doerr and Falko Kuester. CGLX Project (accessed May 10, 2010). `http://vis.ucsd.edu/~cglx/`.

[DTS$^+$09] Matthias Deller, Sebastian Thelen, Daniel Steffen, Peter-Scott Olech, Achim Ebert, Jan Malburg, and Jörg Meyer. A Highly Scalable Rendering Framework for Arbitrary Display and Display-in-Display Configurations. In *Proceedings of the 2009 International Conference on Computer Graphics and Virtual Reality (CGVR)*, pages 164–170, 2009.

[EBG$^+$92] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang, and Brent Welch. Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 599–607, New York, NY, USA, 1992. ACM.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16, New York, NY, USA, 2001. ACM.

[ETO$^+$10] Achim Ebert, Sebastian Thelen, Peter-Scott Olech, Joerg Meyer, and Hans Hagen. Tiled++: An Enhanced Tiled Hi-Res Display Wall. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):120–132, 2010.

[FFP02] Adam Fass, Jodi Forlizzi, and Randy Pausch. MessyDesk and MessyBoard: two designs inspired by the goal of improving human memory. In *DIS '02: Proceedings of the 4th conference on Designing interactive systems*, pages 303–311, New York, NY, USA, 2002. ACM.

[FK03] Randima Fernando and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[GMGA08] Enrico Gobbetti, Fabio Marton, Iglesias Guitián, and José Antonio. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *Vis. Comput.*, 24(7):797–806, 2008.

[goo] Google Earth (accessed July 9, 2010). `http://earth.google.com/`.

[GPSS07] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. Realtime Ray Tracing on GPU with BVH-based Packet Traversal. In *RT '07: Proceedings of the 2007 IEEE Symposium on*

Interactive Ray Tracing, pages 113–118, Washington, DC, USA, 2007. IEEE Computer Society.

[GR01] Saul Greenberg and Michael Rounding. The notification collage: posting information to public and personal displays. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 514–521, New York, NY, USA, 2001. ACM.

[GS87] Jeffrey Goldsmith and John Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Comput. Graph. Appl.*, 7(5):14–20, 1987.

[GSW01] François Guimbretière, Maureen Stone, and Terry Winograd. Fluid interaction with high-resolution wall-size displays. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 21–30, New York, NY, USA, 2001. ACM.

[HHN+02] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002.

[hip] HIPerWall (accessed August 13, 2010). `http://hiperwall.calit2.uci.edu/?q=node/2`.

[HKRs+06] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel. *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006.

[HL09] David M. Hughes and Ik Soo Lim. Kd-Jump: a Path-Preserving Stackless Traversal for Faster Isosurface Raytracing on GPUs. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1555–1562, 2009.

[HLS] Programming Guide for HLSL (accessed May 2, 2010). `http://msdn.microsoft.com/en-us/library/bb509635%28v=VS.85%29.aspx`.

[HMRS06] Elaine M. Huang, Elizabeth D. Mynatt, Daniel M. Russell, and Alison E. Sue. Secrets to Success and Fatal Flaws: The Design of Large-Display Groupware. *IEEE Comput. Graph. Appl.*, 26(1):37–45, 2006.

[Int] MMX, SSE, and SSE2 Intrinsics (accessed May 4, 2010). `http://www.intel.com/software/products/compilers/docs/clin/main_cls/intref_cls/common/intref_sse_details.htm`.

[IYUM04] Hiroo Iwata, Hiroaki Yano, Takahiro Uemura, and Tetsuro Moriya. Food Simulator: A Haptic Interface for Biting. In *VR '04: Proceedings of the IEEE Virtual Reality 2004*, page 51, Washington, DC, USA, 2004. IEEE Computer Society.

[JHKB10] Seokhee Jeon, Jane Hwang, Gerard J. Kim, and Mark Billinghurst. Interaction with large ubiquitous displays using camera-equipped mobile phones. *Personal Ubiquitous Comput.*, 14(2):83–94, 2010.

[KEM07] Andreas Kerren, Achim Ebert, and Jörg Meyer, editors. *Human-Centered Visualization Environments, GI-Dagstuhl Research Seminar, Dagstuhl Castle, Germany, March 5-8, 2006, Revised Lectures*, volume 4417 of *Lecture Notes in Computer Science*. Springer, 2007.

[KK86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1986. ACM.

[kml] The KML Reference (accessed July 5, 2010). `http://code.google.com/apis/kml/documentation/kmlreference.html`.

[Kno08] Aaron Knoll. A Survey of Octree Volume Rendering Methods. In *Proceedings of 1st IRTG Workshop*. GI Lecture Notes in Informatics, 2008.

[KVV+04] N. K. Krishnaprasad, V. Vishwanath, S. Venkataraman, A. G. Rao, L. Renambot, J. Leigh, A. E. Johnson, and B. Davis. JuxtaView - a tool for interactive visualization of large imagery on scalable tiled displays. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 411–420, Washington, DC, USA, 2004. IEEE Computer Society.

[KW03] J. Kruger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 38, Washington, DC, USA, 2003. IEEE Computer Society.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

[LDMA+04] Johnny C. Lee, Paul H. Dietz, Dan Maynes-Aminzade, Ramesh Raskar, and Scott E. Hudson. Automatic Projector Calibration with Embedded

Light Sensors. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 123–126, New York, NY, USA, 2004. ACM.

[Lev88] Marc Levoy. Display of Surfaces from Volume Data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.

[LMS+01] Santiago Lombeyda, Laurent Moll, Mark Shand, David Breen, and Alan Heirich. Scalable Interactive Volume Rendering using off-the-shelf Components. In *PVG '01: Proceedings of the IEEE 2001 Symposium on Parallel and large-data Visualization and Graphics*, pages 115–121, Piscataway, NJ, USA, 2001. IEEE Press.

[Mal09] Jan Malburg. Design und Implementierung eines Renderframeworks für verteilte und stereoskopische Display. Master's thesis, HTW - Hochschule für Technik und Wirtschaft des Saarlandes, Fachbereich Grundlagen, Informatik, Sensortechnik, Germany, 2009.

[Max95] Nelson Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[MBH+02] J. Meyer, R. Borg, B. Hamann, K.I. Joy, and A.J. Olsen. Network-Based Rendering Techniques for Large-Scale Volume Data Sets. In *Farin, G., Hamann, B. and Hagen, H., eds., Hierarchical and Geometrical Methods in Scientific Visualization*, pages 283–296, Heidelberg, Germany, 2002. Springer-Verlag.

[MCEF94] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[Mec] Mechdyne. CAVELib Application Programmer Interface (API) (accessed May 21, 2010). `http://www.mechdyne.com/integratedSolutions/software/products/CAVELib/CAVELib.htm`.

[MH04] Jock D. Mackinlay and Jeffrey Heer. Wideband displays: mitigating multiple monitor seams. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1521–1524, New York, NY, USA, 2004. ACM.

[MHTW00] Aditi Majumder, Zhu He, Herman Towles, and Greg Welch. Achieving color uniformity across multi-projector displays. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 117–124, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[MS05] Aditi Majumder and Rick Stevens. Perceptual photometric seamlessness in projection-based tiled displays. *ACM Trans. Graph.*, 24(1):118–139, 2005.

[MSN] XBox 360 Project Natal (accessed August 4, 2010). `http://www.microsoft.com/uk/wave/hardware-projectnatal.aspx`.

[MSSU04] A. Madhavapeddy, D. Scott, D Sharp, and E. Upton. Using cameraphones to enhance human-computer interaction. In *Sixth International Conference on Ubiquitous Computing (Adjunct Proceedings: Demos)*, 2004.

[mst] Microsoft Tag - Connecting Real Life and the Digital World (accessed July 5, 2010). `http://tag.microsoft.com/consumer/index.aspx`.

[MTSJ07] S. Mikula, I. Trotts, J. M. Stone, and E. G. Jones. Internet-enabled high-resolution brain mapping and virtual microscopy. *Neuroimage*, 35(1):9–15, 2007.

[MWM05] Elke Moritz, Thomas Wischgoll, and Joerg Meyer. Comparison of input devices and displays for protein visualization. *Crossroads*, 12(2):5–5, 2005.

[NBC06] Tao Ni, Doug A. Bowman, and Jian Chen. Increased display size and resolution improve task performance in Information-Rich Virtual Environments. In *GI '06: Proceedings of Graphics Interface 2006*, pages 139–146, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[Ngu08] Huan Thuong Nguyen. *Large-scale Volume Rendering using Multiresolution Wavelets, Subdivision, and Multi-dimensional Transfer Functions.* PhD thesis, University of California, Irvine, 2008.

[NHN07] Nirnimesh, Pawan Harish, and P.J. Narayanan. Garuda: A Scalable Tiled Display Wall Using Commodity PCs. *IEEE Transactions on Visualization and Computer Graphics*, 13:864–877, 2007.

[Nie05] Frank Nielsen. *Visual Computing: Geometry, Graphics, and Vision.* Charles River Media / Thomson Delmar Learning, 2005.

[NSS$^+$06] Tao Ni, Greg S. Schmidt, Oliver G. Staadt, Robert Ball, and Richard May. A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. In *VR '06: Proceedings of the IEEE conference on Virtual Reality*, page 31, Washington, DC, USA, 2006. IEEE Computer Society.

[oM] National Library of Medicine. The Visible Human Project (accessed May 6, 2010). `http://www.nlm.nih.gov/research/visible/visible_human.html`.

[Ope] The OpenCV Wiki (accessed June 2, 2010). `http://opencv.willowgarage.com/wiki/`.

[Phi08] Philips Electronics. Philips WOWvx Technology (accessed May 27, 2010). `http://www.business-sites.philips.com/3dsolutions/home/index.page`, 2008.

[RCB+05] George Robertson, Mary Czerwinski, Patrick Baudisch, Brian Meyers, Daniel Robbins, Greg Smith, and Desney Tan. The Large-Display User Experience. *IEEE Comput. Graph. Appl.*, 25(4):44–51, 2005.

[Rei02] Dirk Reiners. *OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications.* PhD thesis, Technical University Darmstadt, 2002.

[RGW+03] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[RLKG+09] Randi J. Rost, Bill Licea-Kane, Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. *OpenGL Shading Language.* Addison-Wesley Professional, 2009.

[Roh05] Michael Rohs. Visual Code Widgets for Marker-Based Interaction. In *ICDCSW '05: Proceedings of the Fifth International Workshop on Smart Appliances and Wearable Computing*, pages 506–513, Washington, DC, USA, 2005. IEEE Computer Society.

[RS04] Christof Rezk-Salama. *Volume rendering techniques for general purpose graphics hardware.* PhD thesis, Friedrich-Alexander University, Erlangen Nürnberg, Germany, 2004.

[RS06] Bruno Raffin and Luciano Soares. PC Clusters for Virtual Reality. In *VR '06: Proceedings of the IEEE conference on Virtual Reality*, pages 215–222, Washington, DC, USA, 2006. IEEE Computer Society.

[RTD04] Daniel M. Russell, Jay P. Trimble, and Andreas Dieberger. The Use Patterns of Large, Interactive Display Surfaces: Case Studies of Media Design and Use for BlueBoard and MERBoard. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4*, page 40098.2, Washington, DC, USA, 2004. IEEE Computer Society.

[sai] Saitek (accessed June 29, 2010). `http://www.saitek.com/`.

[SBY$^+$06] Lauren Shupp, Robert Ball, Beth Yost, John Booker, and Chris North. Evaluation of viewport size and curvature of large, high-resolution displays. In *GI '06: Proceedings of Graphics Interface 2006*, pages 123–130, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[SDS95a] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for Computer Graphics: A Primer, Part 1. *IEEE Comput. Graph. Appl.*, 15(3):76–84, 1995.

[SDS95b] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for Computer Graphics: A Primer, Part 2. *IEEE Comput. Graph. Appl.*, 15(4):75–85, 1995.

[SEDD09] Daniel Steffen, Achim Ebert, Matthias Deller, and Peter Dannenmann. five: Enhancing 3D Wall Displays with a 2D High-Resolution Overlay. In *INTERACT '09: Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction*, pages 173–186, Berlin, Heidelberg, 2009. Springer-Verlag.

[SG03] Benjamin Schaeffer and Camille Goudeseune. Syzygy: Native PC Cluster VR. In *VR '03: Proceedings of the IEEE Virtual Reality 2003*, page 15, 2003.

[SGH$^+$99] Norbert A. Streitz, Jörg Geissler, Torsten Holmer, Shin'ichi Konomi, Christian Müller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-LAND: an interactive landscape for creativity and innovation. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 120–127, New York, NY, USA, 1999. ACM.

[Shu95] Donald R. Shupe. *Inferential Statisics: An Introduction to the Analysis of Variance*. Mcgraw-Hill College, 1995.

[Sim] Simple DirectMedia Layer. SDL: Simple DirectMedia Layer (accessed May 25, 2010). `http://www.libsdl.org`.

[sta] Stallion Display Cluster (accessed May 19, 2010). `http://www.tacc.utexas.edu/resources/visualization/#overview`.

[SVR$^+$04] Nicholas Schwarz, Shalini Venkataraman, Luc Renambot, Naveen Krishnaprasad, Venkatram Vishwanath, Jason Leigh, Andrew Johnson, Graham Kent, and Atul Nayak. Vol-a-Tile - A Tool for Interactive Exploration of Large Volumetric Data on Scalable Tiled Displays. In *VIS '04: Proceedings of the Conference on Visualization '04*, page 598.19, Washington, DC, USA, 2004. IEEE Computer Society.

[SWND07] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2.1*. Addison-Wesley Professional, 2007.

[SWS09] Alireza Sahami Shirazi, Christian Winkler, and Albrecht Schmidt. Flashlight interaction: a study on mobile phone interaction techniques with large displays. In *MobileHCI '09: Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–2, New York, NY, USA, 2009. ACM.

[SZ04] Luciano P. Soares and Marcelo K. Zuffo. JINX: an X3D browser for VR immersive simulation based on clusters of commodity computers. In *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*, pages 79–86, 2004.

[TMEH09] Sebastian Thelen, Joerg Meyer, Achim Ebert, and Hans Hagen. A 3D Human Brain Atlas. In Nadia Magnenat-Thalmann, editor, *Modelling the Physiological Human*, volume 5903 of *Lecture Notes in Computer Science*, pages 173–186. Springer Berlin / Heidelberg, 2009.

[TMM$^+$09] Sebastian Thelen, Jörg Meyer, Ariane Middel, Peter-Scott Olech, Achim Ebert, and H. Hagen. Tag-Based Interaction with Large High-Resolution Displays. In *Proceedings of the 4th IASTED International Conference on Human-Computer Interation (IASTED-HCI)*, 2009.

[VCBR08] Tamas Vajk, Paul Coulton, William Bamford, and Edwards Reuben. Using a mobile phone as a "Wii-like" controller for playing games on a large public display. *Int. J. Comput. Games Technol.*, 2008:1–6, 2008.

[VMD08] Vincent Vidal, Xing Mei, and Philippe Decaudin. Simple Empty-Space Removal for Interactive Volume Rendering. *J. Graphics Tools*, 13(2):21–36, 2008.

[wan] Wanda (accessed June 29, 2010). `http://www.wandavr.com/`.

[War00] Colin Ware. *Information visualization: perception for design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[War04] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[WBS07] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 26(1):6, 2007.

[Wei06] Daniel Weiskopf. *GPU-Based Interactive Visualization Techniques (Mathematics and Visualization)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[WFKH07a] Ingo Wald, Heiko Friedrich, Aaron Knoll, and Charles D. Hansen. Interactive Isosurface Ray Tracing of Time-Varying Tetrahedral Volumes. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1727–1734, 2007.

[WFKH07b] Ingo Wald, Heiko Friedrich, Aaron Knoll, and Charles D. Hansen. Interactive Isosurface Ray Tracing of Time-Varying Tetrahedral Volumes. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1727–1734, 2007.

[Who] The Whole Brain Atlas (accessed July 7, 2010). `http://www.med.harvard.edu/AANLIB/home.html`.

[WSBW01] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive Rendering with Coherent Ray Tracing. *Comput. Graph. Forum*, 20(3), 2001.

[Xer] Xerces. Xerces: A C++ XML Parser (accessed May 25, 2010). `http://xerces.apache.org/xerces-c`.

[YHN07] Beth Yost, Yonca Haciahmetoglu, and Chris North. Beyond visual acuity: the perceptual scalability of information visualizations for large displays. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 101–110, New York, NY, USA, 2007. ACM.

[YNTT03] Yasuyuki Yanagida, Haruo Noma, Nobuji Tetsutani, and Akira Tomono. An unencumbering, localized olfactory display. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 988–989, New York, NY, USA, 2003. ACM.

[ZHWG08] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time KD-tree construction on graphics hardware. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–11, New York, NY, USA, 2008. ACM.

BIOGRAPHICAL INFORMATION

PERSONAL DETAILS

Name	Sebastian Thelen
Date of birth	August 2, 1980
Place of birth	Trier, Germany
Nationality	German

EDUCATION

01/2008 – present	PhD student at the Department of Computer Science, Computer Graphics and HCI Group, University of Kaiserslautern, Germany
01/2008 – present	Stipend of the International Research Training Group (IRTG) “Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling, and Engineering”
08/2008 – 03/2009	Research visit at the University of California, Irvine, U.S.A.
2001 – 2007	Study of Computer Science at the University of Kaiserslautern, Germany Degree: Diplom-Informatiker (Dipl.-Inf.) (Grade point average 1.0, graduated with distinction) Master Thesis: ”Development of a Framework for the Visualization, Exploration and Comparison of Biological Data”
1991 – 2000	Max-Planck-Gymnasium Trier, Germany (Grade point average 1.7)
1987 – 1991	Grundschule Trier-Irsch, Germany

WORK EXPERIENCE

01/2008 – present	Teaching assistant at the University of Kaiserslautern, Germany (Computer Graphics, Human-Computer Interaction, Scientific Visualization)
09/2007 – 12/2007	Internship at ”Procaess GmbH”, Landau, Germany
10/2003 – 07/2007	Teaching assistant at the University of Kaiserslautern, Germany (Software Development, Algorithms, Logic)
08/2003 – 09/2003	”Umweltzentrum für Energie und Nachhaltigkeit” at the ”Handwerkskammer Trier” - Implementation of a data base application
08/2002 – 09/2002	”Initiative Region Trier e.V.” - Maintenance of a web portal

List of Publications

[KTW$^+$11] Aaron Knoll, Sebastian Thelen, Ingo Wald, Charles D. Hansen, Hans Hagen, and Michael E. Papka. **Full-Resolution Interactive CPU Volume Rendering with Coherent BVH Traversal** (to appear). In *Proceedings of IEEE Pacific Visualization 2011*, 2011.

[EHB$^+$11] Achim Ebert, Hans Hagen, Torsten Bierz, Matthias Deller, Peter-Scott Olech, Daniel Steffen, and Sebastian Thelen. **SEE MORE - Improving the Usage of Large Display Environments**. In *Virtual Realities - Dagstuhl Seminar 2008, Dagstuhl, Germany*, pages 161–180. Springer, 2011.

[TMEH09b] Sebastian Thelen, Jörg Meyer, Achim Ebert, and Hans Hagen. **Giga-Scale Multiresolution Volume Rendering on Distributed Display Clusters** (to appear). In *Workshop Proceedings, Human Aspects of Visualization, INTERACT09*, 2009.

[ETO$^+$10] Achim Ebert, Sebastian Thelen, Peter-Scott Olech, Joerg Meyer, and Hans Hagen. **Tiled++: An Enhanced Tiled Hi-Res Display Wall**. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):120–132, 2010.

[OCT$^+$10] Peter-Scott Olech, Daniel Cernea, Sebastian Thelen, Achim Ebert, Andreas Kerren, and Hans Hagen. **V.I.P. – Supporting Digital Earth Ideas Through Visualization, Interaction and Presentation Screens**. In *Proceedings of the 7th Taipei International Digital Earth Symposium (TIDES 2010)*, pages 36–49, 2010.

[TMEH09] Sebastian Thelen, Joerg Meyer, Achim Ebert, and Hans Hagen. **A 3D Human Brain Atlas**. In Nadia Magnenat-Thalmann, editor, *Modeling the Physiological Human*, volume 5903 of *Lecture Notes in Computer Science*, pages 173–186. Springer Berlin / Heidelberg, 2009.

[TMM$^+$09] Sebastian Thelen, Jörg Meyer, Ariane Middel, Peter-Scott Olech, Achim Ebert, and H. Hagen. **Tag-Based Interaction with Large High-Resolution**

Displays. In *Proceedings of the 4th IASTED International Conference on Human-Computer Interaction (IASTED-HCI)*, 2009.

[DTS$^+$09] Matthias Deller, Sebastian Thelen, Daniel Steffen, Peter-Scott Olech, Achim Ebert, Jan Malburg, and Jörg Meyer. **A Highly Scalable Rendering Framework for Arbitrary Display and Display-in-Display Configurations**. In *Proceedings of the 2009 International Conference on Computer Graphics and Virtual Reality (CGVR)*, pages 164–170, 2009.

[OML$^+$09] Peter-Scott Olech, Ariane Middel, Max Langbein, Sebastian Thelen, Achim Ebert, Joerg Meyer, and Hans Hagen. **Enhancing the Planner's Toolkit – New Display Technologies for Planning Support**. In *International Urban Planning and Environment Association (IUPEA) 8th International Symposium (UPE8)*, 2009.

[ETOH08] Achim Ebert, Sebastian Thelen, Peter-Scott Olech, and Hans Hagen. **An Enhanced Tiled HighRes Display Wall (poster presentation)**. In *IEEE Visualization, VisWeek 08 Conference Compendium*, pages 32–33, 2008.

[TBM$^+$07] Sebastian Thelen, Torsten Bierz, Britta Müller, Hans Hagen, Achim Ebert, Eckard Friauf, and Jörg Meyer. **A Framework for the Visualization of Brain Structures**. In *Visualization of Large and Unstructured Data Sets*, pages 54–63, 2007.