Studies in Interactive Visualization

About the cover: A splat field of the citation index of all IEEE Visualization conference papers. Color indicates the density of papers in different visualization topics, such as flow, volume and information visualization. Grey conic shaped icons represent my contributions to this forum.

The work reported in this thesis has been carried out at the Center for Mathematics and Computer Science (CWI) in Amsterdam under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Studies in Interactive Visualization

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr J. J. M. Franse
ten overstaan van een door het
college voor promoties ingestelde commissie,
in het openbaar te verdedigen
in de Aula der Universiteit
op woensdag 14 maart 2001, te 10.00 uur

door Robert van Liere
geboren te Beiroet, Libanon

Promotor: Prof. dr P. Klint
Faculteit: Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Contents

Preface

The work described in this thesis was done at the Center for Wiskunde en Informatica in Amsterdam. Although I have been at the CWI for more than 15 years, this work was mainly done in the period from 1997 until 2000. In this period I headed the interactive data visualization group in the cluster Software Engineering.

Many people have contributed to the work in this thesis in some way or another. First of all, I would like to thank all the people who have worked in the interactive data visualization group. Jan Harkes, Mariska Hoogenboom, Henk Huitema, Joris Kniesmeijer and Susan Krieger, for all the long hours that were required to keep the projects running. In particular, Wim de Leeuw and Jurriaan Mulder have been exceptional in providing the technical expertise in data visualization and virtual reality.

Paul Klint has been a fantastic mentor in this period. Not only as my research advisor, but also as an advisor for all the organizational problems I encountered throughout the years. Paul's constructive, enthusiast and unbiased approach to problem solving has always been greatly stimulating.

From 1993, Jack van Wijk has acted as a visualization handbook. Jack's original, yet non-mainstream, ideas have been a joy to work with. Memorable also were the morning coffee sessions in the CWI kantine, which brought us up-to-date with the daily gossip.

Roel van Driel has taught me how synergy can be achieved in research projects. I distinctly remember the beginning years and the confusing discussions we had about the word *interaction*. Apparently, biologists and computer scientists associate different meanings to that word.

I would also like to thank the other members of the *leescommissie* (Prof. Frans Groen, Prof. Hans Hagen, Prof. Bob Hertzberger, Prof. Erik Jansen) for their thorough reading of and comments on the thesis.

Finally, I would like to thank Willem, Susan, Floor, Laura, Phil, Wilma, Sabine and Floris for the warmth and kindness they provided outside my professional life. I will always remember the days and nights in the Barnesteeg as being the most lively times in Amsterdam.

Chapter 1

Introduction

> *"... If the scientist cannot produce unambiguous and exhaustive formulations, and wishes instead to exercise his intuitive judgment as the calculation develops, he can arrange for that too. He can instruct the machine to present to him the relevant characteristics of the situation, continuously or in discrete succession, as the calculation progresses, by oscilloscopic graphing. He can then intervene whenever he sees fit."*
>
> J. von Neumann, 1946.

A landmark report commissioned by the US National Science Foundation [1] defined scientific visualization as "a tool both for interpreting image data fed into a computer, and for generating images from complex multi-dimensional data sets. It studies those mechanisms in humans and computers which allow them in concert to perceive, use and communicate visual information". The key ideas are the concept of visual information and the need for direct interaction between the computer and the user.

The purpose of scientific visualization is thus to enhance existing scientific methods by increasing the scientist's ability to see data and understand the results of computations, to interrogate and navigate through datasets, and to supervise and dynamically control the computational processes. Scientific visualization has a multidisciplinary character, intersecting various disciplines in computer science (in particular computer graphics and user interfaces) and mathematics (in particular numerical mathematics and statistics), and also has connections with research in perception and industrial design.

Visual information embraces 'real' visual scenes (such as digital images), mapped data (weather maps, geographical data), visual presentations of numerical data (plots, dynamic graphics, color coded image displays), and visual methods of communication between computer and user. For centuries scientists and mathematicians have searched for methods of presenting data in ways that communicate essential information and lend insight into the underlying scientific or computational process. This need has become increasingly urgent since the computer "data explosion". Visualization of data is the obvious solution, because of the unique capabilities of the human visual system.

Interaction means that the user can control the computational processes generating the display, with immediate and direct effect. Direct interaction enhances realism (e.g., the user can 'move' through a scene). More importantly, interaction is communication in the reverse direction, user to computer. For example, a computational scientist builds mathematical models of physical phenomena that describe successive positions and interactions of real-world objects. These models compute forces, energies, velocities, temperatures and the like as a function of spatial position, and then compute new positions for all the objects. With direct interaction, the scientist would be able to change properties, positions and interactions of the objects. The goal of direct interaction would be to allow scientists to pose questions that would be inconceivable in a non-interactive computation.

The assumption is that direct control can improve the scientist's insight in the relations between input parameters and output results. There are two reasons why direct control is an attractive concept:

- Efficiency.

  Usually, the scientist will prepare an input file, run the simulation, and use visualization to analyze the output data set. In contrast, with direct interaction the user can inspect and interrogate the simulation while it is running. In some cases, the simulation can continue with the new parameter settings, in other cases the simulation must be restarted. However in both cases the efficiency of the analysis cycle has been increased. The scientist can quickly determine if the input parameters were erroneous, or can stop the simulation if, for example, instabilities occur.

- Model exploration.

  With direct interaction, the scientist is able to rapidly explore the parameter space of the model. Exploration can be performed manually or semi-automatic via agents that guide the scientist to an interesting event in the data. If a simulation step can be performed in near real-time, then the scientist can perform sensitivity analysis or cause and effect analysis on various scenarios of the model. Ultimately we can envisage physical processes being controlled by a user through, for example, controlling a microscope stage or a biotechnology process.

Interactive scientific visualization has become technically possible through the rapid development of computer hardware: high-resolution image displays, graphics hardware, memory devices, workstations, supercomputers and networks. Yet this new hardware has stimulated even greater demand for visualization techniques, because one can now capture and generate very large datasets. The ability to capture digital images and complex signal data, and to perform near-realistic simulations, has expanded the horizons of fields such as genetics, fluid dynamics, civil engineering and bio-mathematics, and has created new disciplines such as computational chemistry.

1.1 Taxonomy of visualization techniques

There are many ‘dimensions’ of graphical display information, such as size, shape, spatial position, brightness, color, and apparent visual cues such as perspective, texture, stereo, depth of field, and diffuse and specular reflection [2]. Motion and other time-varying display effects also greatly enhance three-dimensional depth perception and the ability to detect changes. Interactive visualization implies the ability to manipulate these visual images as if they were actual physical systems. Visual displays are created by mapping numerical data onto graphical primitives. Figure 1.1 provides a framework for these mappings [3]. The horizontal axis is the dimensionality of the underlying data, whether it be numerical simulation data or a dataset derived from a physical experiment or acquisition device. The vertical axis is the complexity of the geometric primitive chosen to represent the data, varying from zero-dimensional points to hyper-patches and 3D voxels including visual cues such as color and texture.

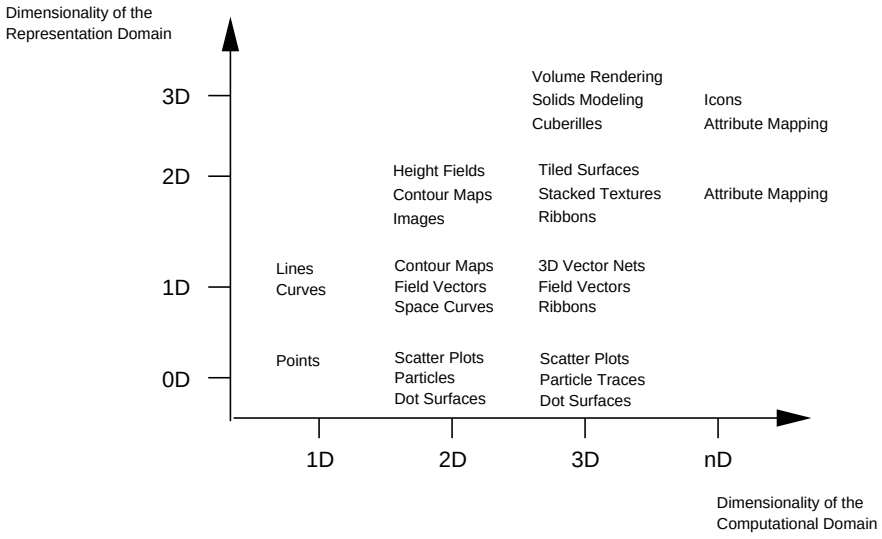


Figure 1.1: The visualization mapping space.

A collection of well known mapping techniques is depicted in the figure. For example, a user may opt to visualize some aspect of a 3-D dataset with 1-D lines using a vector net mapping technique. Alternatively, one may wish to visualize the same aspect using 2-D surfaces by stacking textures.

The whole point of scientific visualization is to provide a set of mapping techniques and guidelines which allow the scientist to pick the best ones that give the most insight into the problem that is being solved.

1.2 Thesis Outline

Substantial research has been done to understand the visualization mapping space for the presentation of results. When combined with interaction, this will narrow the gap between user and visualization system, and increase productivity. In this thesis I will address the topics:

- How and why does the application of interactive visualization result in a more effective modeling / computation / analysis cycle.
- How can the development costs of interactive visualization applications be lowered.

Both topics will be addressed in the areas of computational steering, virtual reality and information visualization.

Computational Steering

Computational steering is a form of scientific visualization which enables users to have direct control over the parameters of a simulation and to supervise and dynamically control the computational process. As an example, Marshall of the Ohio Supercomputer Center has applied computational steering to the study of a 3D turbulence model of Lake Erie [4]. Their conclusions were: “Interaction with the computational model and the resulting graphics display is fundamental in scientific visualization. Steering enhances productivity by greatly reducing the time between changes to model parameters and the viewing of the results.”

Steering has a strong relation with high performance computing and networking (HPCN). First, to gain insight in the ever increasing complexity of high performance simulations, visualization as a post processing step falls short. More advanced interactive visualization methods are needed. Second, high performance computing is needed to execute simulations and rendering at interactive speeds. High bandwidth and low latency networks are needed to interactively handle the vast amount of data produced by HPCN simulations. If interactive speeds cannot be obtained, then most of the merits of computational steering will be lost.

Computational steering is an attractive concept, but its implementation is cumbersome and time consuming. Software tools for computational steering are more demanding than those found in traditional scientific visualization environments. In a traditional visualization environment, the scientific end-user must cooperate with a specialist in user-interfaces and visualization to develop a tool for the analysis of the output of the simulation. When the tool is ready, after some weeks or months, the interests of the end user have shifted. Also, the further analysis of the data will introduce new research questions, which induce modifications of the tool. A close cooperation between end user and the visualization specialist for an extended period is required.

The Computational Steering Environment (CSE) is an environment for computational steering which provides a collection of methods, techniques, and tools that enable researchers to apply computational steering. The work on the CSE was initiated in 1995 in a joint project between CWI and the Netherlands Energy Research Foundation ECN [5]. It was designed to meet the following goals:

- Provide a steering environment for the easy and rapid integration of existing simulations.
- Provide a research environment which could be extended with new steering modules.
- Provide an environment which could be executed efficiently on a large range of platforms.

Data management is an important aspect of steering large scale and high performance simulations. In a computational steering environment, data management would be responsible for the bi-directional mapping and storage of data structures used in the simulation and those used by the visualization modules. In this thesis we describe the development of a distributed data manager for the CSE which has been designed to provide flexible and portable capabilities for the storage and remote access of scalable data sets. Also, the remote access and transport of data sets must be scalable and portable.

Virtual Environments

A virtual environment (VE) application is a computer synthesized virtual world with which a user can interact. Virtual environments differ from other computer-centered systems in the extent to which real-time interaction is facilitated, the perceived visual space is three-dimensional rather than two-dimensional, the human-machine interface is multi-modal, and the user is immersed in the computer generated environment. An important question is if scientific visualization can benefit from virtual environments and, if so, how?

The development of virtual environments is a field open to new applications, different device configurations, and new techniques for rendering and interaction. These developments are most productive when new techniques can be implemented quickly and then compared to previous implementations. However, developing even simple VE applications with limited or no software support requires considerable amounts of expertise and development time. In order to fulfill the real-time requirements posed on VR applications, programmers must learn many new and non-standard methods, related to the management of multiple I/O data streams, 3D interaction and mappings to a wide variety of device configurations.

To support the rapid development of VE applications, we have developed the Portable Virtual Reality (PVR) library and runtime system. We designed PVR to meet the following goals:

- Provide software support for the creation of simple VE applications for novice developers.
- Provide a framework for extending simple VE applications to include additional interaction techniques.
- Allow applications to be developed independently from the hardware configuration used at runtime.

Support for the rapid development of simple VE applications is critical for allowing developers to experiment with the technology and techniques common to VE applications. A framework for extensions to the default behavior allows for focused experimentation with new techniques in different aspects of the applications; further, it allows these new techniques to be easily incorporated into VE applications. One area of experimentation is determining, for a particular VE application, the best configuration of input devices (such as tracking, glove, and button devices) and output devices (such as head-mounted displays, stereoscopic or single-view projection screens, and monitors). The PVR design allows a variety of device configurations to be specified at runtime by providing a separation between the devices used and the environment model, and by defining how the device input affects the model.

What sets PVR apart from other 3-D rendering systems is that it provides support for extensions that can be developed independently and can be easily integrated. In addition, PVR allows for a variety of input and output device configurations to be selected for a particular application with little or no additional programming. Device independence is achieved by using the PVR environment model as the interface between input and output devices and the application.

Information Visualization

The term Scientific Visualization is used when data from physical phenomena is presented. Examples of such data can be found in the life sciences, computational chemistry and meteorology. Data sets from these application areas usually have inherent spatial properties, so that the mapping from data to geometry is usually straightforward. In contrast, the term information visualization is used for the presentation of abstract data [6]. Examples of abstract data can be found in applications such as digital libraries, complex documents, history profiles, and network topologies. Data sets from these applications usually have no obvious spatial properties, and the mapping from data to geometry can be defined in a number of ways. The main problem of information visualization is the mapping of non-spatial, abstract data onto effective visual forms. However, both data and information visualization make use of similar underlying techniques for data structuring, data selection, data filtering and rendering. The main difference between scientific and information visualization lies in the kind of data that is presented.

Relational structures, consisting of a set of entities and relationships between entities are commonly used to store data in data bases. Such structures are commonly modeled as graphs: the entities are modeled as vertices, relationships as edges, and additional information is associated with the vertices and edges. A large number of graph drawing algorithms have been developed that present the information in a graph effectively.

In order to study issues related to the visualization of large graphs, we have developed GraphSplatting, a technique that transforms a graph into a two-dimensional scalar field. The graph is not displayed as graphical representations of vertices and edges, but as a continuous field which preserves the main structure conveyed by the graph.

1.3 Evaluating Interactive Visualization Systems

Many questions arise about design, performance, usability, and cost-effectiveness as a system progresses from inspiration to realization. A carefully chosen program of evaluation studies, conducted throughout the development and operational life of a system, can help provide the answers to such questions, as well as reduce development time and minimize the need for expensive design changes.

There are many reasons why evaluation studies should be conducted:

- At the outset of development, evaluation studies can be used to refine the requirements for a system and to compare design concepts.
- Once a system design has been chosen, evaluation studies can be used to diagnose problems and suggest alternative approaches.
- Finally, when a system has been developed, a summative evaluation can be used to measure the capability of the system to fulfill its intended function, to compare its performance with that of alternative systems, and to assess its acceptance by intended users.

Design evaluation. The purpose of defining reference models (i.e.,, frameworks) for interactive visualization systems is to describe the components that characterize a system and to identify the issues that must be addressed. Frameworks provide a means to understand and discuss the design decisions and trade-offs made during the development of these systems. Frameworks also help to identify the level at which interoperability between systems might occur.

From an application programmer's perspective, the specific visualization system issues that a framework should address include the programmer's interface to the system, the visualization technique definition, the execution model, the data model, and how support for program monitoring and debugging is provided. From the perspective

of an end user, the issues that a framework should address include the (visual) language provided to link visualization modules together, and the user interface to each module.

Many researchers have used frameworks for comparing interactive visualization systems. For example, [7, 8, 9] compare different interactive data visualization system designs. [10, 11] are examples of comparing virtual reality system designs.

Capability evaluation. The creation of interactive visualization systems provides research and development challenges to a wide variety of disciplines, including computer science, engineering, human factors, cognitive psychology, and sensorimotor psychophysics. In each discipline, the requirements associated with creating cost-effective interactive visualization systems raise new questions that call for evaluation. Evaluation studies of interactive visualization systems are needed to help ensure that:

- The perceptual and cognitive capabilities and limitations of human beings, as well as the needs of the specific tasks under consideration, are being used as driving criteria for system design.

  This evaluation task consists of measuring the physical characteristics of the interactive visualization system and considering how these characteristics are related to those of the prospective human user. For example, the characteristics of the displays and controls (dynamic ranges, resolutions, time lags, distortions) should be measured and compared with the sensorimotor capabilities of humans determined from psychophysical studies.

  Another portion of the evaluation effort will focus on analysis of the task to be performed and examination of how well the system is designed to perform the task. Such an evaluation should take into account the physical fidelity and how such fidelity is expected to influence performance on various task components.

  For example, Hinkeley et. al. [12] and Buxton [13] survey design issues related to 3D input devices and how these design issues can effect human performance. Ware [14] and Watson [15] study the effect of lag on task performance in a virtual environment. Baker [16] explores human factors issues in using virtual environments for the analysis of scientific data. Slater [17] describes a method to evaluate the sense of presence in a virtual environment.

- Hardware and software deliver interactive visualization systems in a cost-effective manner.

  Tools to analyze interactive visualization systems can help motivate the designer about the functionality and provide insight in price/performance trade-offs for system concepts. These tools can provide a quantitative metrics which can be used to isolate and compare functions of different systems.

  Specific visualization system issues that would be addressed include the execution model of the visualization system, the effects of system lag, the effects

of the network in the case of a distributed system, and the effects of graphics hardware on the overall performance.

For example, Zyda [18] provides a quantitative discussion on how the design and implementation of a networked virtual environments can effect its performance. Bryson [19] gives a detailed analysis of static distortion of position data from 3D trackers. Funkhouser [20] discusses adaptive display algorithms for interactive frame rates during visualization of complex virtual environments. These adaptive algorithms trade-off display quality in order to obtain a time-critical performance target.

Informal evaluation. Despite the clear need for evaluation in interactive visualization systems, the types and amounts of evaluation currently taking place in this area are limited. This is undoubtedly due to the high level of enthusiasm that exists about what the technology is likely to be able to accomplish. Accordingly, the informal evaluations that take place more or less automatically as one is developing a system (i.e.,, a system is good or bad according to whether it is used or not) are sufficient. Although these forms of informal evaluation are necessary, the cost-effectiveness of the research and development is likely to be significantly increased if the task of evaluation is taken more seriously.

Instead of performing rigorous experiments on well defined components of a system, evaluation – supported only by possibly unrepresentative experiments – is performed by real end users. Brooks suggests to define three classes of results – findings, observations, and rules-of-thumb – as a means to report on these type of evaluations [21]. Findings are those results properly established by soundly-designed experiments. Observations report user-behavior. Rules-of-thumb are generalizations, even those unsupported by testing.

Many case studies have reported on how interactive visualization systems have significantly improved analysis of application data. For example, Marshall [4] shows how computational steering significantly improves insight into complex turbulence models. Bryson has shown how interactive visualization can improve the analysis of 3D unsteady fluid flows [22].

The position taken in this dissertation is to informally evaluate the developed systems by judging the effect that the system has on the overall modeling / computation / analysis cycle for the end user. The motivation for taking this position is that the formal knowledge regarding the human factors involved in designing interactive visualization systems is still minimal and that the existing knowledge is difficult to obtain. Some guidelines for system design do exist, but are somewhat fragmented and difficult to generalize.

Chapters 3 and 6 provides some initial quantitative studies by measuring some parameters of the CSE and PVR architectures. These studies are limited, since the number of relevant parameters needed to study the price/performance of a interactive visualization system is very large.

1.4 Road Map

This thesis is organized in three parts:

Part I (Computational Steering) addresses a solution to distributed data management for computational steering of high performance applications. Chapter 2 describes the data storage and transport techniques that have been implemented and integrated in the CSE. Chapter 3 is concerned with the implementation of the data storage and transport techniques. In Chapter 4, these techniques have been applied to steer a medium scale smog simulation.

Part II (Virtual Reality) addresses a solution for the development of portable virtual reality applications. In Chapter 5 the motivation, design and implementation of the Portable Virtual Reality (PVR) are discussed. Chapter 6 is concerned with the implementation of PVR. Chapter 7 demonstrates the effectiveness of virtual reality for the analysis of complex system: PVR is applied to the Fekete problem. This has led to a number of new techniques, which have been integrated into PVR.

Part III (Information Visualization) addresses a solution for the display of large graphs. In Chapter 8, the motivation and implementation of GraphSplatting are discussed. In Chapter 9, Graphsplatting is applied to the exploration of high dimensional feature spaces. We show how a graph can be used as an intermediate representation of the feature space, and discuss the importance of interaction to define the graph.

1.5 Origins of the Chapters

Most chapters in this thesis have appeared as a paper in a journal or in the proceedings of an international conference. Only minor changes have been made to each published paper. The chapters 2, 4, 5, 7, 9 have appeared in:

- R. van Liere, J.A. Harkes, and W.C. de Leeuw.
  A Distributed Blackboard Architecture for Interactive Data Visualization.
  IEEE Visualization '98, 1998.

- R. van Liere and J.J. van Wijk.
  Steering smog prediction.
  HPCN-Europe '97, 1997.

- R. van Liere and J.D. Mulder.
  PVR - an Architecture for Portable VR Applications.
  5th Eurographics Workshop on Virtual Environments, 1999.

- R. van Liere, J.D. Mulder, J.E. Frank and J.J.B. de Swart.
  Virtual Fekete Point Configurations: a case study in perturbing complex systems.
  IEEE Virtual Reality 2000, 2000.

- R. van Liere, W.C. de Leeuw and F. Waas.
  Interactive Visualization of Multidimensional Feature Spaces.
  ACM New Paradigms for Information Visualization, 2000.

Chapter 8 is pending publication.

1.6 Acknowledgements

The work on the Computational Steering Environment was initiated in 1995 by Jarke van Wijk in a joint project between CWI and the Netherlands Energy Research Foundation ECN. The work on the distributed blackboard architecture described in Chapter 2 evolved from this collaboration.

The techniques and systems presented in this thesis are the result of various projects I managed from 1997 until 2000. Although I have carried out the design and much of the implementation of these systems, many colleagues have contributed to the projects. In particular,

- Jan Harkes implemented a substantial amount of the blackboard system discussed in Chapter 2.
- Maarten van Loon provided the smog application and solver discussed in Chapter 4.
- Jurriaan Mulder implemented the workspace model in the PVR environment discussed in Chapter 5. Jurriaan also designed and implemented the moonscapes in the Fekete application discussed in Chapter 7.
- Jason Frank and Jacques de Swart provided the Fekete application and solver discussed in Chapter 7.
- Wim de Leeuw implemented the GraphSplatting technique discussed in Chapter 8.
- Florian Waas provided the image sets and features for the image similarity application Chapter 9.

References

[1] B. McCormick, T. Defanti, and M. Brown. Visualization in Scientific Computing. *Computer Graphics*, 22(6 (SIGGRAPH '88)):103–111, 1987.

[2] William S. Cleveland and Robert McGill. Graphical perception. *Journal of the American Statistical Association*, pages 548–554, 1984.

[3] C. Upson. Volumetric Visualization Techniques. In D.F. Rogers and R.A Earnshaw, editors, *State of the Art in Computer Graphics*, pages 313–350. Spriger Verlag, 1991.

[4] R.E. Marshall, J.L. Kempf, D. Scott Dyer, and C-C Yen. Visualization Methods and Simulation Steering a 3D Turbulence Model of Lake Erie. *1990 Symp. on Interactive 3D Graphics, Computer Graphics*, 24(2):89–97, 1990.

[5] J.J. van Wijk and R. van Liere. An environment for computational steering. In G.M. Nielson, H. Müller, and H. Hagen, editors, *Scientific Visualization: Overviews, Methodologies, and Techniques*, pages 89–110. Computer Society Press, 1997.

[6] S.K. Card, J.D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization*. Morgan Kaufmann Publishers, 1999.

[7] C. Williams, J. Rasure, and C. Hansen. The state of the art of visual languages for visualization. In A.E. Kaufman and G.M Nielson, editors, *Proceedings IEEE Visualization '92*, pages 202–209. IEEE Computer Society Press, 1992.

[8] W. Gu, J. Vetter, and K. Schwan. An annotated bibliography of interactive program steering. *SIGPLAN Notices*, 29(9):140–148, 1994.

[9] J.D. Mulder, J.J. van Wijk, and R. van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15:119–129, 1999.

[10] J. Wood. *Collaborative Visualization*. PhD thesis, The University of Leeds,School of Computer Studies, February 1998.

[11] G.D. Kessler, D.A. Bowman, and L.F. Hodges. The simple virtual environment: An exstensible framework for building ve applications. *Presence: Teleoperators and Virtual Environments*, 9(2):187–208, 2000.

[12] K. Hinckley, R. Pausch, J.C. Goble, and N.F. Kassell. A survey of design issues in spatial input. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '94)*, pages 213–222, 1994.

[13] W. Buxton. There's more to interaction than meets the eye. In D. Norman and S. Draper, editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*, pages 319–337. Adison-Wesley, 1986.

[14] K.W. Arthur, K.S. Booth, and C. Ware. Evaluating 3d task performance for fish tank virtual worlds. *ACM Transactions on Information Systems*, 11(3):239–265, July 1993.

[15] B. Watson, V. Spaulding, N. Walker, and W. Ribarsky. Evaluation of the effects of frame time variation on vr task performance. In *1997 IEEE Virtual Reality Annual International Symposium*, pages 38–44. IEEE Computer Society Press, March 1997.

[16] M.P. Baker. Human factors in virtual environments for the visual analysis of scientific data. Technical Report (draft), NCSA Ames, September 1995.

[17] M. Slater, M. Usoh, and A. Steed. Depth of presence in virtual environments. *Presence: Teleoperators and Virtual Environments*, 3(2):130–144, 1994.

[18] S. Sinhal and M. Zyda. *Networked Virtual Environments - Design and Implementation*. Addison-Wesley Publishing Company, 1999. ISBN 0201325578.

[19] S. Bryson. Measurement and calibration of static distortion of position data from 3d trackers. Technical Report RNR-92-011, NASA Ames, March 1992.

[20] T.A. Funkhouser and C.H. Séquin. Adaptive display algorithms for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 247–254, 1993.

[21] F.P. Brooks, Jr. Grasping reality through illusion – interactive graphics serving science. In *Proceedings of Computer-Human Interface'88 (CHI'88)*, pages 1–11. ACM Press, 1988.

[22] S. Bryson and C. Levit. The virtual windtunnel: An environment for the exploration of three-dimensional unsteady fluid flows. In *Visualization '91 (Proceedings of the 1991 Visualization Conference)*, pages 17–24, 1991.

Part I

Computational Steering

Chapter 2

A Distributed Blackboard Architecture for Interactive Data Visualization [1]

In this chapter the motivation, design and application of a distributed blackboard architecture for interactive data visualization is discussed. The main advantages of the architecture are twofold. First, it allows visualization tools to be tightly integrated with simulations. Second, it allows qualitative and quantitative analysis to be combined during the visualization process.

2.1 Introduction

The need for enhanced data modeling and data integration in visualization environments has been widely recognized and has been a topic of interest in recent workshops and conferences [1, 2]. Although visualization environments provide rich support for graphical modeling and rendering, many visualization researchers feel that integration of the visualization environment with the simulation environment remains a problem to be solved.

In this chapter a distributed blackboard architecture for interactive data visualization is discussed. The architecture has been used in various projects related to high performance computing [3] and computational steering [4].

Motivation This work was motivated by the following observations:

- State-of-the-art scientific visualization tools are not sufficiently integrated into simulation environments. In computational steering, users can investigate in-

[1] This chapter has been published as: R. van Liere, J.A. Harkes and W.C. de Leeuw, A Distributed Blackboard Architecture for Interactive Data Visualization. In R. Yagel and H. Hagen, *IEEE Visualization '98*, pages 235–244, 1998.

termediate results and can modify the computation before completion. Feedback during the investigation may be required, for example to steer computation around local minima. Users may interactively steer simulations through adjustment of an application's critical parameters. To support computational steering tight integration between simulation and visualization is required.

- Large data sets defy inspection by visualization alone. Analysis of simulation output often consists of a combination of visual inspection and numerical or statistical analysis. Scientific visualization environments lack the integration of general purpose analysis tools. Visualization combined with interactive analysis will allow the scientist to explore the data both visually and numerically. Combining the two allows the user to perform qualitative and quantitative analysis simultaneously. In addition, the output of the analysis tools can be combined with the visualization of the simulation data.

- Efficient support for very large data sets is limited. Very flexible data models have been defined that allow the import/export of data from/to the simulation environment [5, 6]. However, in order to effectively embed these data models in the visualization environment, two system related issues have to be addressed: scalability of data, and access to remote computations.

Many visualization researchers believe that state-of-the-art visualization environments do not adequately address these problems. For example, Foley and Ribarsky [7] point out that next-generation visualization environments require, amongst others, a means to bind data to geometry and a general analysis model.

Blackboard Models Blackboard models have been widely used in the AI community as a particular kind of problem solving model [8, 9]. The blackboard model allows multiple independent agents (usually called knowledge sources) to share information in a central store (called the blackboard). The model serves as a scheme for organizing reasoning steps and domain knowledge to construct a solution to a particular problem. For example, in a forward reasoning model, problem solving begins by reasoning forward from initial data towards a goal. In this case each knowledge source will contribute its specific knowledge towards the goal.

Knowledge is segmented into modules and a separate inference engine is provided for each module. Communication between modules is realized by reading and writing in the blackboard. The blackboard can be partitioned so that it contains regions with differing, but perhaps related, data structures. In this way applications can organize the solution space into one or more application dependent hierarchies.

The blackboard model does not explicitly specify a control component. It merely specifies a general problem solving behavior. The actual locus of control can be in the knowledge modules, in the blackboard itself, in a separate module, or in a combination

of these.[2]

The difficulty with this description of the blackboard model is that it only outlines the organizational principles. For those who want to build a blackboard system, the model does not specify how it is to be realized as a computational entity. However, given a problem to be solved, the blackboard model provides enough guidelines for sketching a solution.

The blackboard model can be used as a framework for implementing visualization environments. This will be substantiated in Section 2.2.6 after we discuss some details of the distributed blackboard architecture.

Chapter Format The format of the chapter is as follows. In the next sections some design details of the distributed blackboard architecture are presented. First we give an overview of the architecture, its ingredients, and its programming abstractions. Then, in Section 2.2.6 we discuss the merits of the architecture and how our original motivations are addressed. In Section 2.3, some related work that resembles the blackboard architecture is discussed. Finally, Section 2.4 shows how general analysis tools can be integrated into the visualization process using the blackboard architecture.

2.2 Distributed Blackboard Architecture

2.2.1 Overview

A simplified overview of the architecture is shown in figure 2.1. The architecture pro-

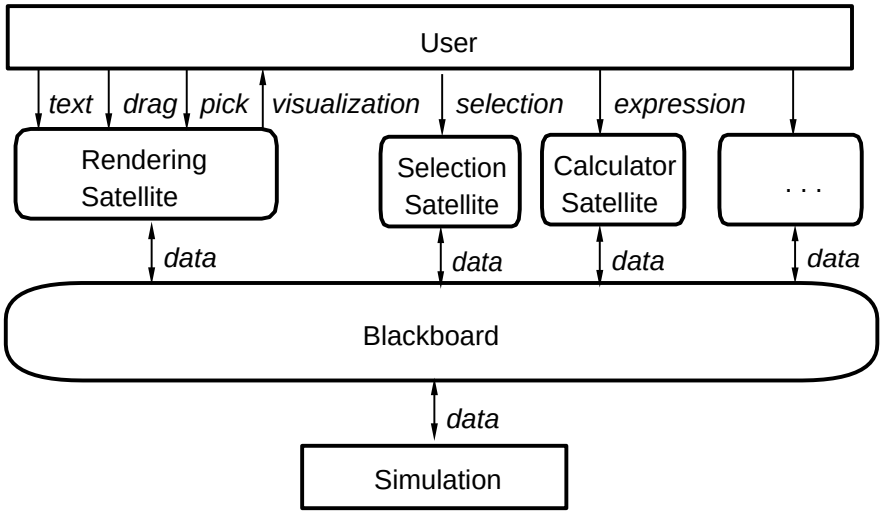


Figure 2.1: The user view of the blackboard architecture.

[2] Initial proposals considered the blackboard only as a passive memory with external control modules that monitored the changes in the blackboard. Later – efficiency related – refinements integrated the control modules into the blackboard.

vides an interface between a user and a simulation. It is centered around *blackboard* and *satellite* processes that produce and consume data. Satellites implement the simulation, analysis programs, geometry mapping and rendering algorithms. The purpose of the blackboard is twofold. First, it manages a database of variables. Satellites can create, open, close, read, and write variables. Second, it acts as an event notification manager. Satellites can subscribe to events that represent state changes in the blackboard. Whenever such an event occurs, it will publish the event to all subscribing satellites.

A large collection of general purpose satellites have been developed. For example, a data slicing satellite, a calculator, a data logger, a VTK satellite that provides all VTK functionality [10], etc. Also, a 3D interactive editor and rendering satellite that binds data to geometry has been developed [11].

2.2.2 Variables

The basic blackboard object is the variable, which encapsulates all information required to access a blackboard object. Variables are defined as a tuple consisting of four components: a name, a type descriptor, raw data and a list of attributes (see figure 2.2). Names uniquely identify the variable. The variable descriptor determines the

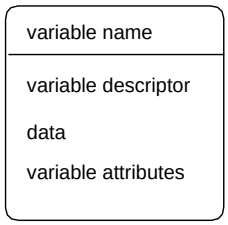


Figure 2.2: Four components of a variable.

type, size and layout of the data. The data component is the storage container for the raw data. Attributes are name value pairs that may be used to describe meta-data of the variable.

The underlying data model supports two composite data types:

- *Regular topology.*

  Data which has been generated from the following grid types; i.e., uniform, rectilinear and curvilinear grids. The regular topology data type is very similar to the data model supported by NetCDF [12]. In this case, the type descriptor contains all information concerning the shape and dimensionality of the variable.

- *Geometry lists.*

A geometry list is a list of geometric elements. Each element can be a simple geometric object, such as polygon, polyline, mesh, light, camera, etc. The functionality offered by the geometry list is very similar to that offered by the low level interface of P3D [13].

Future extensions to the data model will include explicit support for data with an irregular topology and scattered data. Note that these extensions do not affect the semantics of the variable, but only the expressiveness of the underlying data model.

Operations on variables are very similar to low level file operations: create, open, close, read, write, and getdescriptor. Simple set/get operations are available to manipulate attribute lists.

Two scatter/gather techniques are supported to optimize I/O operations on variables. First, many variables can be read/written simultaneously in one atomic I/O operation. Second, a comprehensive data mapping mechanism is provided to allow data to be sliced, sub-sampled and the like during the I/O operation. This allows satellites to use a different data layout compared to the data structure stored in the blackboard. The identity mapping allows a one-to-one copy between storage in the satellite and the blackboard.

2.2.3 Architecture

The distributed architecture consists of four building blocks: a global name manager (denoted as GNM), one or more local blackboards (LBB), one or more satellites, and typed streams.

- Global Name Manager: The GNM maintains the bookkeeping information of all LBBs, satellites and variables in the system. Only the variable names, descriptors and attributes are stored in the GNM. Variable data is not stored in the global name manager.

- Local Blackboard: A local blackboard resides on each host in the distributed environment. Local blackboards accept connections from satellites executing on the same host and other LBBs.

  Variable data is stored in the LBB, and is shared by all connecting satellites. Each LBB maintains a copy of the variable data.

  The LBB manages only those variables that are opened by the connected satellites. When a satellite opens a variable, the LBB consults the GNM to check if the same variable exists in other LBBs. If this is the case, the LBB will connect with these LBBs. A LBB-LBB connection is used to maintain variable consistency (variable consistency is addressed in the next paragraph).

- Satellites: A satellite is a process which communicates with its corresponding LBB. Satellites may create, open, close, and read/write variables, as well as subscribe to variable events.

An abstract satellite is shown in figure 2.3. Basically, it consists of an operator that transforms input data into output data. Control determines when this operation has to be carried out, or, in other words, when a satellite is triggered. Operators can also be controlled by additional parameters manipulated via user interface widgets.

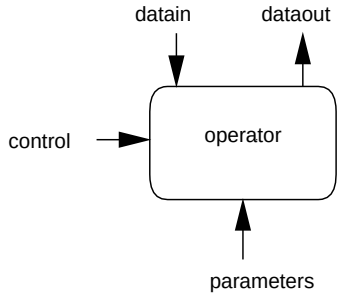


Figure 2.3: Interfaces to an abstract satellite.

Data input and output is performed by read/writing variables. Input and output triggering is discussed in Section 2.2.5.

- Command, event and data streams: A connection between a satellite and the LBB consists of a command, an event and a data stream. Commands from the satellite to the LBB are sent over the command stream. The LBB sends events to the satellites via the event stream. Data streams are used to transport data between LBBs and satellites.

Figure 2.4 shows an example configuration of the distributed blackboard architecture. This configuration shows two LBBs and four satellites. Both local blackboards are always connected to the GNM. The local blackboards share a variable, hence, they are connected.

Satellites execute in parallel but LBBs are single threaded, so that only one satellite can access the LBB simultaneously. However, access to different LBBs is concurrent.

When a satellite writes a variable, the LBB will broadcast a mutate event to all connected satellites and LBBs that share the variable. When a satellite reads a variable, the LBB will first check if the data is up to date and, if so, will service the read request. If the data is not up to date, the LBB will first get the latest copy of the data from another LBB before servicing the read request. The details of this algorithm are very similar to the cache consistency algorithms found on cache based SMP machines.

2.2.4 Programming Abstractions

Satellite programmers can access the LBB using three layered APIs. Each layer provides a higher level of abstraction. Higher layers are easier to use, but provide less functionality than the lower layer.

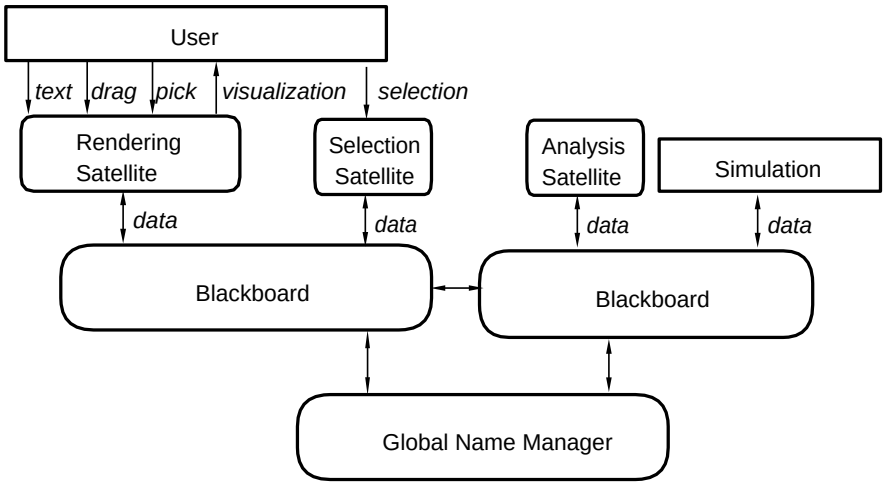


Figure 2.4: Distributed blackboard architecture.

1. The local blackboard API is a low level library interface which provides functionality for LBB communication, variable management and event management. This layer provides all details of the underlying LBB protocols. It requires detailed knowledge and is difficult to use due to the inherent parallelism in the system.

2. The high level data input output layer is built on top of the local blackboard API. Many cumbersome low level details are shielded from the user. In particular, the data input output layer hides the notion of events and has builtin support for structuring variables into sets, and support for handling efficient set I/O.

   A design goal of the data input output layer was to keep the required changes to the simulation code minimal. As an example of the data input output layer, consider the following C program:

```
simulation(float *s, float *t, int *size, float *x)
{
   int continue = TRUE;

  /* Open connection, connect and subscribe vars */
   dioOpen("borneo.cwi.nl");
   dioConnectFloat("s", s, READ);
   dioConnectInt("continue", \&continue, READ);
   dioConnectFloatArray("x", x,  1, size, UPDATE);
   dioConnectFloat("t", t, WRITE);

   /* simulation loop and update data */
```

```
while (continue)
{
   t = t + 1.0;
   calculate\_values(t, s, size, x);
   dioUpdate();
}
dioClose();
```

The structure of this example, which is typical for continuous simulations, consists of two parts. First, variables are initialized. The required changes to existing source code are limited to opening and closing a connection with the blackboard and connecting the variables via the `dioConnect` routines. Second, a main loop is entered where time is incremented and new values are calculated. The required change to the source code is a single call to exchange data. The locations where to insert these calls are easy to find; typically at the outer level of the simulation program.

The first two parameters of the `dioConnect` routines are the name of the variable and its address. For the connection of arrays the number of dimensions and their sizes must also be specified. The last parameter is used by the `dioUpdate` routine to determine the direction of the data flow. In `dioUpdate` first the event stream from the LBB is checked if variables to be read or updated have changed. If so, these variables are read from the LBB. Next the values of all unread variables are written to the LBB. The net result of `dioUpdate` is that all connected variables have the same value in the simulation and the LBB. With these few calls the user can interact with parameters (`s`) of the simulation, to stop the simulation (`continue`), monitor its progress (`t, x`) or change state variables (`x`).

To deal with more hierarchical situations, variables may be grouped into sets. In the main loop the satellite can read and write specific sets, and wait until a particular set mutates.

3. A extensible and embedded scripting language built on top of the data input output. Scripting can be used for simple operations on variables, such as slicing and logging. The advantage of scripting is its ease of use in developing satellites.

2.2.5 Satellite Control and Synchronization

Satellites cooperate via the basic input/output mechanisms that are provided for variables. Writing to a variable will cause an event to be sent to all satellites subscribed to that variable. This mechanism is used to mediate the execution of satellites. The user can specify that if a particular variable – the *input trigger variable* – is mutated, the operator has to be evaluated. The action of operator evaluation is called *triggering*. Furthermore, the user can also specify an *output trigger variable*, which is to be written to each time the operator has been evaluated. Input and output trigger variables can

be linked together to mediate the execution of satellites. In general, linking two trigger variables defines a data dependency between these two variables. Linking a number of variables results in an directed graph, which we call the *trigger graph*.

A high level trigger scripting language and script interpreter satellite have been developed to simplify the definition of trigger variables. The task of the trigger script interpreter satellite is to manage the trigger graph. As an example, consider the trigger graph shown in figure 2.5: A slicing and dicing satellite operates on simulation output,

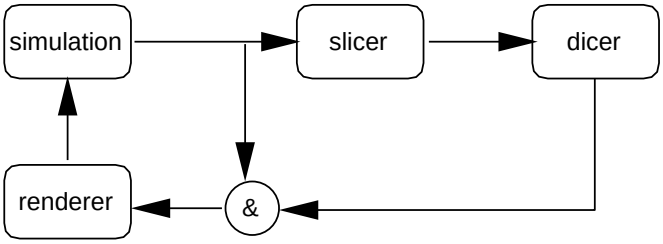


Figure 2.5: Control loop of four satellites defined by a trigger script

which in turn is the input for the rendering satellite. The simulation may only compute the next step after the rendering satellite has drawn the previous frame. The rendering satellite depends on variables computed by the simulation and the dicer. The sample script to realize this synchronized configuration is:

```
simulation > slicer
slicer > dicer
simulation & dicer > renderer
renderer > simulation
```

A trigger script is defined as a sequence of trigger rules. The syntax of each rule is :

```
rule ::= expr '>' name
expr ::= expr '&' expr |
         expr '|' expr |
         expr ';' expr |
          '(' expr ')' |
         name
name ::= satellitename
```

The trigger script interpreter satellite has been integrated in the data input output layer. Whenever the trigger script satellite interprets a trigger rule, it sets the attributes of the trigger input variable with a representation of this rule. The data input output layer uses this information to determine when a satellite is to be triggered. Note that attributes of the trigger input variable are set by the trigger script interpreter satellite and can be reset during the lifetime of the satellite.

2.2.6 Discussion

The distributed blackboard extends the centralized blackboard in many ways : it provides support for efficient data movement among heterogeneous hosts, scales for large data sets, and offers a richer data model. Although the architecture is distributed, it is important to note that the programming model is not affected, i.e., the user and programmer view of a blackboard is still a centralized data store.

The architecture has been implemented on a number of heterogeneous UNIX and NT machines. TCP/IP is used for communication between LBB's and, when possible, shared memory is used for satellite to LBB communication. Rendering satellites are available for many display types, ranging from low end laptop displays to sophisticated VR displays, such as the CAVE.

In retrospect, we believe that the architecture provides support to fulfill our original requirements:

- Integration.

  The variable naming mechanism is used to bind data structures in the blackboard to data structures in the satellite. The event mechanism is used to maintain the consistency of these data structures. The net effect is that a two-way binding exists between data in the blackboard and data in the satellite.

  For example, a rendering satellite can utilize this by binding geometry to variables in the blackboard. A simulation can also bind the same variables. Hence, computational steering is supported.

- Qualitative vs. quantitative analysis.

  Integration of data structures is not restricted to simulation and rendering satellites alone, but can be used by any satellite. General purpose analysis tools can be packaged as satellites. Hence, the analysis of simulation output can be a combination of visual inspection and numerical/statistical analysis.

- Efficiency of data transport.

  Distributed blackboards maintain local copies of a data structure. A single event will be broadcasted when a satellite mutates the data structure. The data structure will be transported to another blackboard only when it is needed. This mechanism – called 'transport by demand' – saves bandwidth if data structures are written frequently but read only occasionally.

- Ease of use.

  Using the low level libraries require knowledge about event driven and parallel programming abstractions. However, higher level libraries shield all these details and allow a programmer to easily bind for instance Fortran data structures to variables in the blackboard. In this way existing simulation code can rapidly be integrated into the environment.

Also, programmers need not know that the blackboard is distributed. Abstractions for opening and manipulation variables are very similar to the familiar file handling abstractions.

2.3 Related work

Many research and development teams have designed and implemented interactive visualization environments. Giving an in depth analysis of other visualization environments is outside the scope of this chapter – see [14] for an elaborate annotated bibliography on various aspects of interactive data visualization, including interactive program monitoring and steering. Instead of an extensive overview of related work, we discuss work dealing with issues that relate to our blackboard architecture.

Williams, Rasure and Hanson [15] provide a framework to understand design tradeoffs when developing data flow based visualization systems. Data flow systems are attractive because of the similarities with the visualization pipeline: users can easily organize their simulation, filter, map and render modules in an intuitive way. However, data flow environments do not provide support to deal directly with the underlying data, except for importing/export data from a file or simulation. Hence, the integration with the underlying data is limited.

CAVEvis [16] is a distributed real-time visualization system for streaming large scalar and vector fields into the CAVE. The governing idea is to render the geometry as fast as possible in order to maintain the highest level of interactivity. It uses separately running modules to asynchronously generate the geometry data, such as modules to generate iso-surfaces or particle paths. Sequence names and time-stamps are used to gather data related to a frame, regardless of the order in which the data is received. Our blackboard architecture does not explicitly support any form of sequencing and control. Rather, the synchronization is used to provide similar functionality.

SuperGlue [17] is a programming environment for scientific visualization whose main goal is extensibility and ease of use. The approach used is to offer a very generic set of primitive data structures and a inter-language interface, which programmers use to integrate data into the SuperGlue system.

2.4 Radon Forecasting

The distributed blackboard architecture has been applied to an atmospheric transport application. In the hope that systematic simulation errors can be found, researchers are interested in comparing simulated concentrations with actual measurements. Simulation errors can arise from modeling errors, numerical errors, visualization errors, and input errors. Using our system we want to discover systematic errors that occur due to a combination of:

- Spatial errors: geographical locations of the simulated data differ from the measured data.
- Temporal errors: the simulated data differ from the measured data in time.
- Scaling errors: the simulated data is systematically higher or lower than the measured data; e.g., due to an inaccurate emission sources.

Various automated data analysis techniques have been developed that search for regions in the simulated data that fit the measured data. Details of these techniques, which are are based on statistical comparison and fitting methods, are outside the scope of this chapter.

The goal of this particular case is the accurate forecasting of radioactive noble gas Radon (^{222}Rn) concentrations based on measured wind and emission fields. The simulated Radon concentrations were compared with measured concentrations on three islands in the Indian ocean.

Visualization Figure 2.6 gives an overview of the ongoing Radon transport simulation over the Indian ocean. Spot noise was used to display the wind fields. A rainbow color mapping was used to display the Radon concentrations and small colored circles showing the measured concentrations are drawn on the three sites where measured data is available. The three sites are located at: Amsterdam Island ($77\,\mathrm{deg}\,34'E, 37\,\mathrm{deg}\,50'S$), Crozet Island ($51\,\mathrm{deg}\,52'E, 46\,\mathrm{deg}\,26'S$), and Kerguelen ($70\,\mathrm{deg}\,15'E, 49\,\mathrm{deg}\,21'S$), all on French territory.

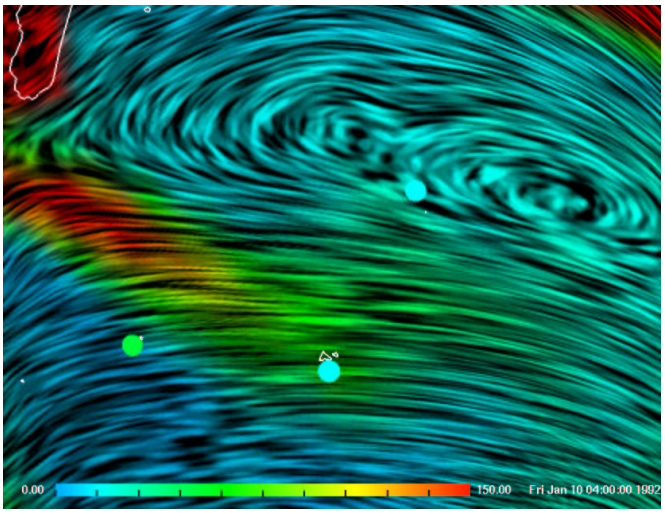


Figure 2.6: Radon concentrations over the Indian ocean. Small colored circles show measured sites data. (see color plate on page 150)

Figure 2.7 shows a sequence of snapshots of the automated point fitting process. Semi-transparent circles are drawn at points calculated by the data analysis techniques.

The opacity of the circles is mapped to the fit of the data. Transparent circles indicate points of poor fit; opaque circles indicate points of better fit. The left image of figure 2.7 depicts the points in an initial configuration around Amsterdam Island. A new configuration is derived by deleting a poor fitting point and taking for a new point at a random position close to the best fit point. In this way the process converges to a minimum, the area which best fits the data. The middle image shows the configuration after a number of steps. Finally the points may converge to a stable configuration, as indicated in the right image.

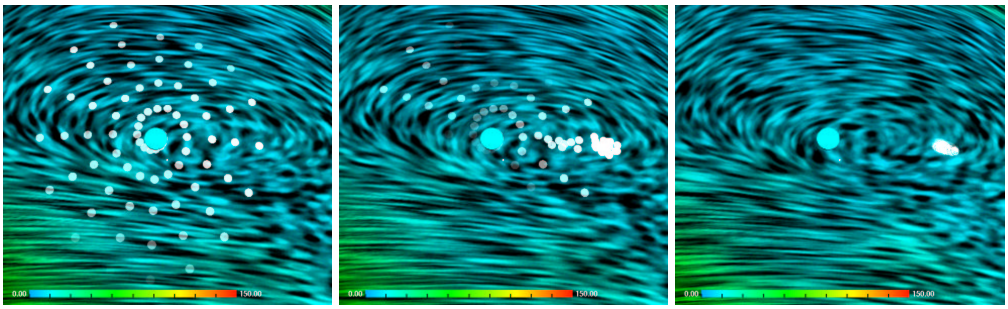


Figure 2.7: Comparative time sequence of converging point set. (see color plate on page 150)

A plotting satellite was used to show time series of a scalar value. The output of the plotting satellite is shown on the bottom of figure 2.8. The three plots show: the measured data at Amsterdam Island (top plot), the simulated data at the point of measurement (middle plot), and the best fit found by the analysis satellite (bottom plot).

The user may at any time also edit the set of points by dragging any point to a different location. This is useful if the user suspects other local minima in the data which may be missed by the analysis software.

Blackboard Figure 2.9 is a diagram of the blackboard and the satellites around it. The Radon simulation satellite creates a set of three variables containing the wind fields and the simulated scalar Radon field. After each time step this set will be dumped into the LBB. A reader satellite reads the measured site data from a data base and writes this data onto the blackboard.

The analysis satellite creates the variable containing the candidate points. It continuously dumps new candidate points into the LBB until a stop criterion is reached. In addition, the analysis satellite opens the variables created by the simulation. Whenever a new data set is produced it will read it, and if any of the candidate points have been mutated, it will read the new candidate points. The visualization satellite will read and display the data sets and candidate points.

Upon any variable mutation, the corresponding satellite will be triggered. After

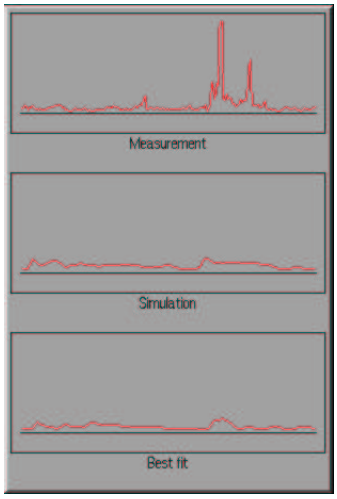


Figure 2.8: Measured data at Amsterdam Island (top) simulated data (middle) and best fit (bottom).

each time step the simulation will dump data into the LBB. Alternatively, the user may drag a point to a new position, resulting in a new set of candidate points which are written to the LBB.

Discussion The governing idea of this example is to show how the blackboard model is used in a non-trivial setting. The application combines qualitative user actions (direct manipulation of the visualization) with quantitative analysis tools (computations of numerical algorithms). Several levels of information can be differentiated: on the lowest level computed data and measured data is available, analysis satellites consume this data to produce information of a higher level. The user, in turn, can interact with the simulation or analysis satellites as a reaction to this information.

The distributed blackboard architecture is a natural framework for solving such problems. Whenever information at a certain level is mutated, the appropriate satellite recalculates its output using the new information and mutates the next level of information. Due to the blackboard architecture, data can be shared among all satellites.

2.5 Conclusions

In this chapter we presented a distributed blackboard architecture for scientific visualization. The blackboard architecture allowed us to address two important issues concerning interactive visualization environments. First, tight integration between simulation and visualization. This is realized through the name concept of a variable which tightly binds data stored in the blackboard with data in the satellite. Second, to combine qualitative and quantitative data analysis. This is realized by allowing general

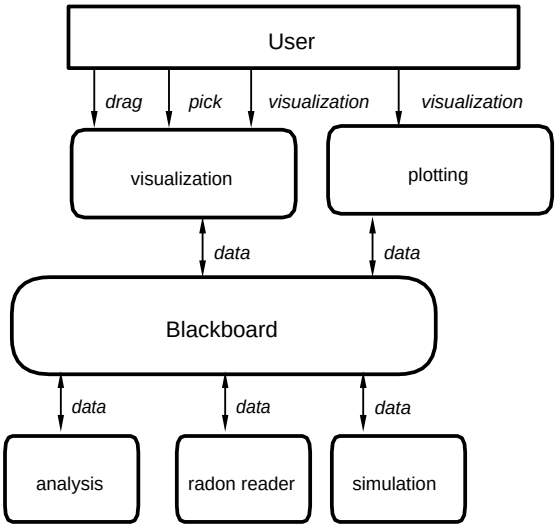


Figure 2.9: Blackboard configuration of Radon application.

analysis satellites to operate in close cooperation with the visualization satellites.

The Radon application is very simple and should be seen as an elementary case study. In the future we plan to apply the distributed architecture to a 3D ozone simulation over the Netherlands. Here, the fitting criteria will be a volume, the chemical reactions involved in computing ozone are much more complicated, and the measured ozone data is less reliable. Nevertheless, there are signs that this type of visualization based analysis will provide added value to the atmospheric researcher [18].

References

[1] J.P. Lee and G.G. Grinstein, editors. *Database Issues for Data Visualization*. Springer Verlag, 1993.

[2] L.A. Treinish. Workshop report: data structure and access software for scientific visualization. In *SIGGRAPH'90 workshop report*, volume 25(2), pages 104–118. ACM, 1991.

[3] W.C. de Leeuw and R. van Liere. Divide and conquer spot noise. In *Proceedings Supercompting '97 (http : // scxy.tc.cornell.edu / sc97 / program / TECH/DELEEUW / INDEX.HTM)*. ACM, 1997.

[4] R. van Liere and J.J. van Wijk. Steering smog prediction. In B. Hertzberger and P. Sloot, editors, *Proceedings HPCN-Europe '97*, pages 241–252. Springer Verlag, April 1997.

[5] R. Haber, B. Lucas, and N. Collins. A data model for scientific visualization with provisions for regular and irregular grids. In G.M. Nielson and L. Rosenblum, editors, *Proceedings Visualization '91*, pages 298–305, 1991.

[6] W.L. Hibbard, C.R. Dyer, and B.E. Paul. A lattice model for data display. In R.D. Bergeron and A.E. Kaufman, editors, *Proceedings Visualization '94*, pages 310–317, 1994.

[7] J. Foley and W. Ribarsky. Next-generation data visualization tools. In L. Rosenblum, R.A. Earnshaw, J. Encarnacao, H. Hagen, A. Kaufman, S. Klimenko, G. Nielson, F. Post, and D. Thalmann, editors, *Scientific Visualization : Advances and Challenges*, pages 103–126. Academic Press, 1994.

[8] R. Engelmore and T. Morgan, editors. *Blackboard Systems*. Adison-Wesley, 1988.

[9] V. Jagannathan, R. Dodhiawala, and L. Baum, editors. *Blackboard Architectures and Applications*. Academic-Press, 1989.

[10] W.J. Schroeder, K.M. Martin, and W.E. Lorensen. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. In R. Yagel and G.M. Nielson, editors, *Proceedings Visualization '96*, pages 93–100, 1996.

[11] J.D. Mulder and J.J. van Wijk. 3D computational steering with parametrized geometric objects. In G.M. Nielson and D. Silver, editors, *Proceedings Visualization '95*, pages 304–311. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[12] R.K. Rew and G.P. Davis. The UNIDATA netCDF: Software for scientific data access. In *6th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydology*, pages 33–40, Anaheim, CA, 1990.

[13] J. Welling, C. Nuuja, and P. Andrews. P3D: A lisp-based format for representing general 3d models. In *Proceedings Supercompting '90*, pages 766–774. ACM, 1990.

[14] W. Gu, J. Vetter, and K. Schwan. An annotated bibliography of interactive program steering. *SIGPLAN Notices*, 29(9):140–148, 1994.

[15] C. Williams, J. Rasure, and C. Hansen. The State of the Art of Visual Languages for Visualization. In A.E. Kaufman and G.M. Nielson, editors, *Proceedings Visualization '92*, pages 202–209, 1992.

[16] V. Jaswal. CAVEvis: Distributed real-time visualization of time-varying scalar and vector fields using the CAVE virtual reality theater. In R. Yagel and H. Hagen, editors, *Proceedings Visualization '97*, pages 301–308, 1997.

[17] J. Hultquist and E. Raible. SuperGlue: A programming environment for scientific visualization. In A.E. Kaufman and G.M. Nielson, editors, *Proceedings Visualization '92*, pages 243–250, 1992.

[18] F. Dentener. *Personal Communication*, Institute for Marine and Atmospheric Research (IMAU), Febuary, 1998.

Chapter 3

CSE Implementation Issues

An overview of the CSE architecture was presented in Chapter 2. In this chapter a more in-depth discussion on the technical issues related to the design and implementation is given.

3.1 Design

3.1.1 Data Storage and Mappings

A variable managed by the blackboard consists of a name, type, value, dimensions, and user defined attributes. Data is stored in the blackboard as an n-dimensional array (where n is 0, 1, 2, …) containing elements of the same scalar type (e.g., 8-bit character, 32-bit integer). A scalar variable is represented as a 0-dimensional array. Although, the number of dimensions is fixed during the lifetime of a variable, the value of a dimension can be changed.

If a satellite can structure data in the same way as it is stored in the blackboard, a read and write operation will result in a copy of data from the satellite to the blackboard. However, it is often the case that data in (legacy) simulations is not structured as in the blackboard. Instead of maintaining temporary copies, the satellite may define mappings between data stored within the satellite and the way the data is stored in the blackboard. In this way, a read and write operation will result in a more complex copy of data from the satellite to the blackboard. Three pieces of information are needed to perform a mapped copy operation:

- A description of the location where data is to be placed in the blackboard's array. This description is defined by three vectors: a start vector, a stride vector and a count vector.

- A description of how the data is structured in the satellite. This description is defined by an index map: a mapping between indices of the data elements and their memory addresses.

- The memory address of the first element in the satellite's data structure.

Figure 3.1.1 illustrates six examples of data mappings.

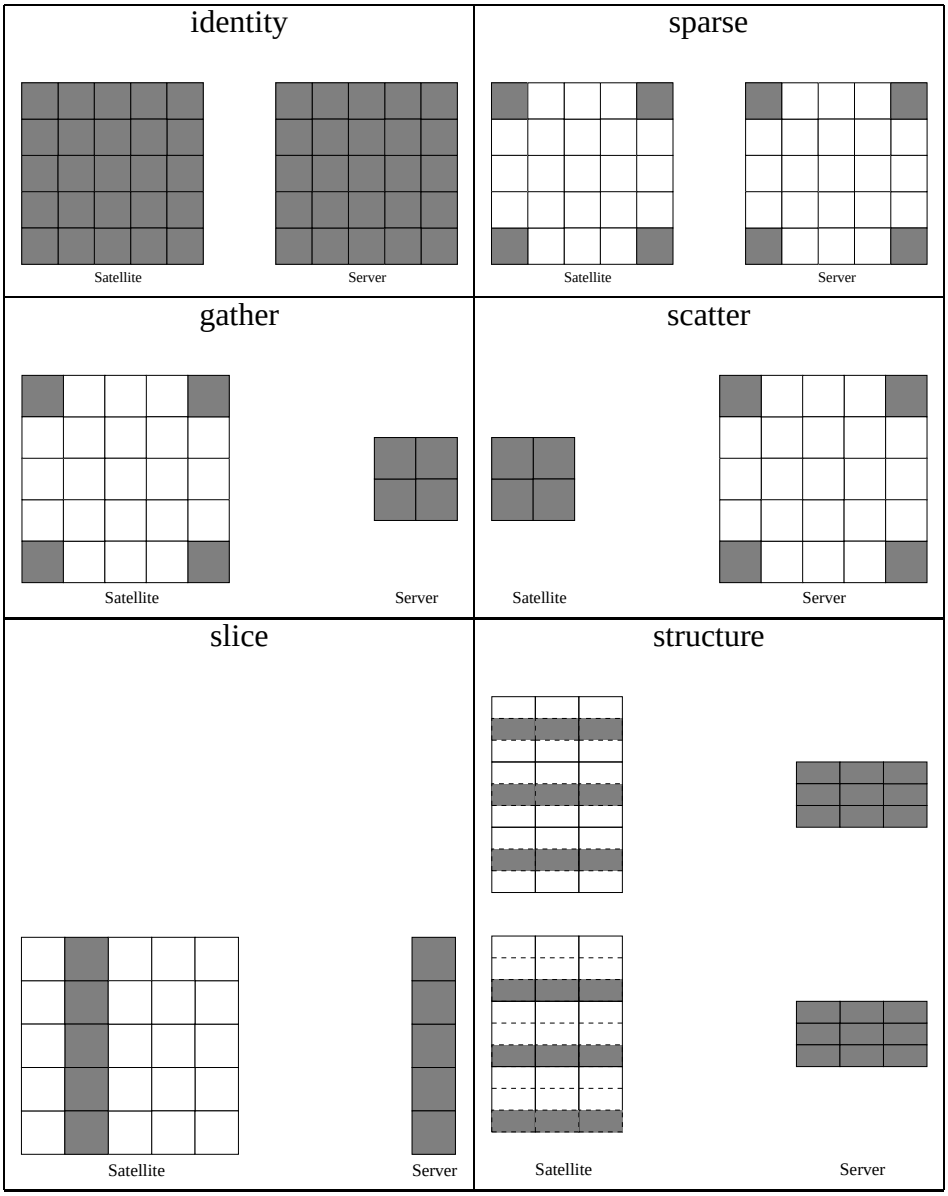


Figure 3.1: Mapping satellite variables to blackboard variables.

The example labelled 'structure' illustrates how a structure map can be used. The fields px and py in an 2D-array of structure `struct { int flags; float px, py;} coords[NX][NY];` are mapped onto `float px[NX][NY]` and `float py[NX][NY]` in the server. The specification involves (i) a description of the location where data is to be placed in the server, (ii) a description of how the data is to

be mapped, and (iii) the memory address of the first element in the satellite's data structure.

For the description of the location in the server:

```
int start  = {0, 0};
int stride = {1, 1};
int count  = {NX, NY};
```

For a description of how the data is to be mapped: The satellite's array steps 1 element in the Y-dimension and NY elements in the X-dimension.

```
int *imap = {NY*sizeof(struct coord),sizeof(struct coord)};
dmMAP map = dmMapCreate(2, start, stride, count);
dmMapSetIMAP(map, imap);
```

For a memory address of the first element in the satellite's data structure. We also need to specify a different start addresses in the write operation:

```
int px_start = &coords[0][0].px;
int py_start = &coords[0][0].py;
```

This information is used in the write operation as:

```
dmVarWrite(px_vid, &px_start, map);
dmVarWrite(py_vid, &py_start, map);
```

3.1.2 Architecture

The architecture of the CSE consists of three components: a global name manager, zero or more local blackboards, and zero or more satellites. These components are connected by and communicate via *command*, *event* and *data* streams:

- The global name manager maintains the bookkeeping information of all satellites and variables in system. It does not store variable data.

  Each blackboard is connected with the global name manager with command and event streams.

- A local blackboard resides on each machine in the distributed environment. Satellites connect to a local blackboard and communication is realized via three streams: a command stream, a data stream and an event stream. Variable data is stored in the local blackboard, and is shared by all connected satellites. Each local blackboard maintains a copy of the variable data.

  Satellites are connected with their blackboards via command, data and event streams.

- A satellite communicates only with its corresponding local blackboard. Communication to a blackboard from a satellite is done by sending commands on the command stream. Communication from blackboard to a satellite is done by receiving events from the event stream. Variable data is communicated via the data stream.

Figure 3.2 shows an overview of the architecture from the perspective of a local blackboard. Each blackboard manages a variable store. Satellites, other blackboards and one global name manager are connected with command, event and data streams. Although satellites and remote blackboards can operate concurrently, the local blackboard will process commands sequentially. In addition, each command is processed synchronously, i.e., is completed before the next command is processed. This allows a sequential consistent variable storage to be implemented.

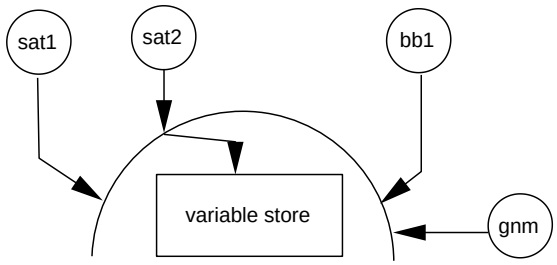


Figure 3.2: Local blackboard with sequential consistent variable store.

The variable store can be updated only when: a variable is created/opened/closed, or a variable value is updated, or a variable value is updated in a remote blackboard. Read and write operations on a variable are valid only if the variable is in a valid state. The blackboard will manage the state of every variable, see figure 3.3.

Each variable can have three states:

- VALID

  The valid state indicates that the most recent value of the variable resides on the blackboard. A satellite or remote blackboard is allowed to read and write the variable in this state.

  The variable will be set to mutated if the variable is written in a remote blackboard. The variable will be set to invalid if the value of any dimension of the variable is mutated.

- MUTATED

  The mutated state indicates that the value of the variable is out of date and a more recent value resides on a remote blackboard. The variable is set to the mutated state when a satellite connected to a remote blackboard writes to it. The remote

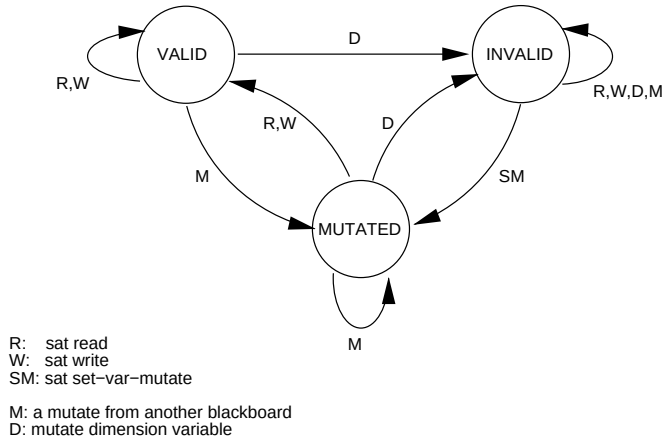


Figure 3.3: State diagram of a variable.

blackboard will send a mutate event to this blackboard, indicating that a local value is out of date. The latest copy of the value must first be retrieved from the remote blackboard before a local satellite can read the variable.

The variable will be set to the valid state when the latest copy is retrieved from the remote blackboard. The variable will be set to invalid if the value of any dimension of the variable is mutated.

- INVALID

  This state indicates that one of the dimensions of the variable has been mutated. The blackboard will send events to all satellites when a variable is set to invalid state. Each satellite must explicitly send a 'set-var-mutate' command to get the variable out of this state. The motivation for this is that the satellite will need to reallocate enough memory before performing the next read or write request. With the 'set-var-mutate' command, a satellite indicates that reallocation has been performed.

The control flow of a command between two blackboards is diagrammed in figure 3.4. Solid lines denote control within the process, dashed lines denote protocol requests and replies.

In this example, the satellite issues a read command of a mutated variable in blackboard BB1. A new copy of the value will first be obtained from blackboard BB2, before flushing the value and returning the control back to the satellite.

The global name manager will administer all satellites and variables. Variable data is not stored in the global name manager. If, for example, a new variable is opened in blackboard BB1 then the global name manager will report this to BB2 so that both blackboards can synchronize the state of the variable.

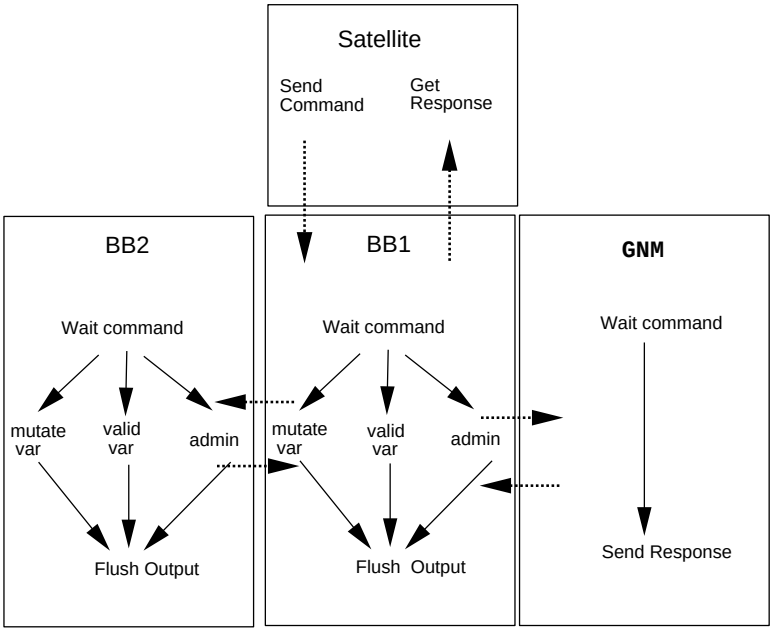


Figure 3.4: Flow of control between satellite, two blackboards and the global name manager.

3.1.3 Application Programmer Interfaces

Two application programmer interfaces allow the programmer to program satellites.

- The blackboard interface API.

  The blackboard API is a low level interface for storing and retrieving variables in the blackboard. It supports an abstraction that provides a view of data as a collection of network-transparent variables. API functions are available to create, open and close variables, read and write data to variables, and query variable properties.

  The blackboard API also provides an event system that denotes state changes in the blackboard variables. Satellites may subscribe to events, which will be sent by the blackboard when the event occurs. For example, a satellite may subscribe to mutation events on a variable. The blackboard will send an event to the satellite whenever the variable is mutated.

  Functions are available to access all blackboard functionality, including satellite connection management, variable management, variable input output (optionally with maps), variable attribute setting and querying, variable dimension handling, variable state querying, variable locking, and event management.

  Table 3.1 lists a subset of the blackboard library functions.

Return	Name	Description
dmID	dmOpen	Open blackboard connection
dmErr	dmClose	Close blackboard connection
dmVID	dmVarCreate	Create a variable
dmErr	dmVarClose	Close a variable
dmErr	dmVarsWrite	Write variables
dmErr	dmVarsSample	Read variables
dmBool	dmEventAvailable	is event available
dmErr	dmNextEvent	get event
dmErr	dmGetVarState	get variable state
dmErr	dmValidateVar	set variable in VALID state
dmVID	dmGetMutatedVids	get mutated variable ids
dmVID	dmGetInvalidVids	get invalid variable ids
int	dmGetVarAttr	get variable attribute
dmErr	dmSetVarAttr	get variable attribute
dmErr	dmGetVarName	get variable name
int	dmGetVarType	get variable type
int	dmGetVarNDims	get variable number of dimensions
int	dmGetVarDim	get variable dimension

Table 3.1: Subset of blackboard library functions.

- The data input/output API.

  The data input/output API is a high level library built on top of the blackboard API. Many cumbersome low level details are shielded from the user of data input/output library.

  The data input/output library supports an abstraction that binds a variable in a satellite to a variable in the blackboard. An update function is available, which guarantees that the variable value in the blackboard is up to date with the value in the satellite. The satellite is responsible to call the update function when necessary. The data flow of the update function is bidirectional: variables that have been mutated in the blackboard will be read, else variables will be written.

  In addition, the data input/output library hides the notion of events and has builtin support for structuring variables into sets, support for handling set updates, support for changes in variable dimensions.

  Table 3.2 lists a subset of the data input/output library functions.

  The data input/output API is designed to be simple and for allowing the rapid integration of existing simulations. An example of the flavor of the data input/output API is given in Chapter 2.

Return	Name	Description
dioERR	DioOpen	Open blackboard connection
dioERR	DioClose	Close blackboard connection
dioSID	DioOpenSet	Create a set
dioERR	DioCloseSet	Close a set
dioFID	DioOpenFilter	Create a filter
dioERR	DioCloseFilter	Close a filter
dioERR	DioAddVarFilter	Add a variable to open filter
dioVID	DioNewVariable	Create a variable
dioERR	DioDelVariable	Delete a variable
dioERR	DioUpdate	Update all sets
dioERR	DioFilterUpdate	Update filtered set of variables

Table 3.2: Subset of data input/output library functions.

3.2 Implementation

3.2.1 Portability

The implementation of the blackboard is designed to be portable and inter-operable between a variety of machine architectures. It has been ported to a variety of machine types and operating systems. Currently the blackboard has been ported to SGI (64 and 32 bit versions, IRIX), SUN (32 bit, Solaris), Cray C90 (64 bit, Unicos), and Linux (32 bit Intel machines), Windows NT (32 bit Intel machines). The minimum support required from the operating system for implementing the backboard are TCP sockets. The software can be configured to use other capabilities of the machine and operating system. For example, if the operating system supports shared memory, the implementation will use shared memory where possible.

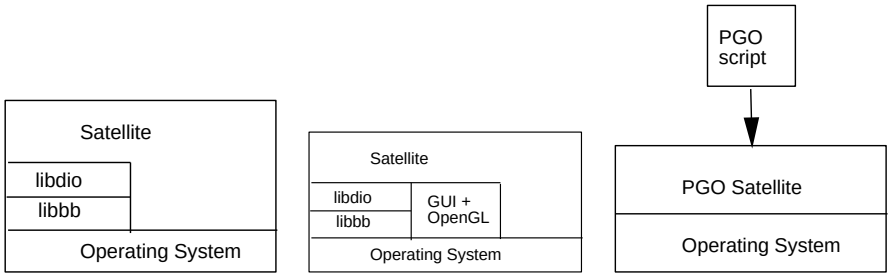


Figure 3.5: The layered implementation of CSE satellites. Left is a satellite without a user interface. Middle is a satellite with a custom user interface. Right is the PGO editor satellite.

Figure 3.5 illustrates the layered implementation of CSE satellites. The left panel shows a satellite without a user interface. These satellites can optionally use the data input/output library to connect with the blackboard, or use the blackboard library directly. The middle panel shows a satellite with a user interface. These satellites can manage the graphical user interface via a toolkit and use OpenGL for the visualization. The right panel shows the PGO editor satellite. The Parametrized Geometric Objects (PGO) editor plays a central role in the CSE. Users can create user interfaces by drawing graphical objects and binding these objects to data variables in the blackboard. The binding is bidirectional: if a variable is mutated by another satellite, then the position or appearance of the graphical object will change. If the user drags the graphical object then the value of the bound variable is mutated. A PGO interface can be saved as a script and used input for the PGO editor.[1]

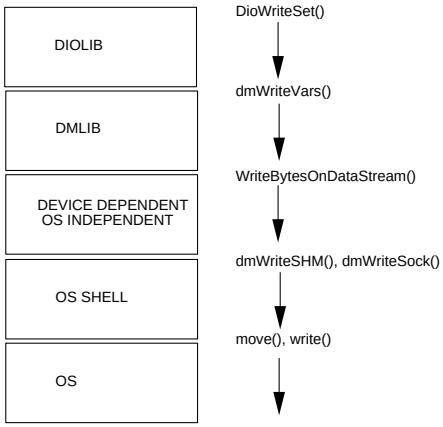


Figure 3.6: Implementation layers of portable input/output library.

Figure 3.6 shows an example of the layered implementation of the data input/output API `DioWriteSet()` function. This function is implemented as one or more blackboard API `dmWriteVars()` functions. `WriteBytesOnDataStream()` is a device independent data stream function, whose implementation will use the lower level write data abstractions, for example `dmWriteVarSHM()` or `dmWriteVarSock()` depending on the type of connection to the blackboard. Finally, `dmWriteVarSHM()` or `dmWriteVarSock()` are implemented as a block move in the shared memory case, or a write on a file descriptor in the socket case.

This implementation scheme will allow different data transport libraries to be used. For example, if the message passing interface library (MPI) is to be used, a function at the level of `dmWriteVarSHM()` will be needed.

[1] Jack van Wijk designed and implemented the 2D version of the PGO [1]. Jurriaan Mulder designed and implemented the 3D version of the PGO [2].

Module	LoC	Description
share	3084	portable stream support
GNM	1608	global name manager
BB	4102	blackboard manager
libbb	2603	blackboard API
libdio	2797	data I/O API
Utilities	**LoC**	**Description**
dmcalc	1701	data calculator + gui
dmls	283	list properties of one or more variables
dmlog	427	log of one or more variables
dmpgo	42311	PGO editor
dmtimer	202	update a variable at a given frequency
Applications	**LoC**	**Description**
smog	8141	transport equations solver + adaptive grids
smog	326	dio interface (24 variables: 4 scalar + 20 array)
smog gui	4899	User interface: custom viewer + dio interface
cfd	3712	euler equations + multigrid solver
cfd	445	dio interface (27 variables: 15 scalar + 12 array)
cfd gui	0	User interface : PGO editor

Table 3.3: Code metrics.

3.2.2 Metrics

The blackboard architecture was implemented in C. Table 3.3 gives an indication of the effort that is needed for the implementation. The table lists the module, lines of code and a short description, for the kernel modules, some utilities and two applications. Lines of code were obtained by removing all comments from the files and applying the UNIX utility 'word count' on the files. This is not exact, as the numbers include empty lines.

The first part of table 3.3 shows the number of lines of code for the four modules: the global name manager, blackboard manager and the blackboard API modules use a shared portable I/O support.

The second part of the table shows the number of lines of code for some utility satellites. For example, `dmls` is a satellite that lists the properties of all variables in the blackboard's store. `dmcalc` is a calculator which can be used for simple arithmetic on variables in the blackboard. The calculator will recompute the left hand side of an expression whenever a variable in the right hand side of the expression is mutated.

The third part shows the amount of code needed for the smog application (discussed in Chapter 4) and a 2D-computational fluid dynamics simulation. The solvers of both simulations were written in Fortran77. The smog application uses a custom viewer for the user interface. The CFD application uses the PGO editor for its user

interface. The amount of code needed to bind simulation variables to data in the blackboard was 326 and 445 lines of code, respectively.

A number of observations can be made from these tables:

- The amount of code needed to implement the utilities (excluding `dmpgo`) is modest.
- The amount of code needed for graphical user interfaces is large. However, no coding is necessary when the PGO editor is used to bind graphical objects with blackboard variables.
- The amount of code needed to bind simulation variables to the blackboard is relatively small when using the data input/output library.

3.3 Performance

The performance of a computational steering application requires the analysis of the computation time of the simulation, data transport from compute server to rendering workstation, and rendering time of the visualization. In this section we discuss the performance of the blackboard architecture by considering only the performance of data transport and rendering of the smog application. Performance measurements were performed on various network and workstation configurations.

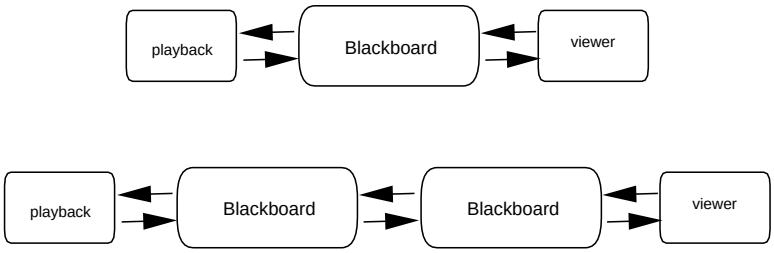


Figure 3.7: Blackboard configurations used for performance measurements. Local playback and viewer (top), remote playback and viewer (bottom).

Instead of running the smog simulation, a data base of pollutants was generated and a playback satellite was used to dump data frames into the blackboard. Figure 3.7 shows the blackboard configuration used for the performance measurements. For each time step, the playback satellite will dump a data frame (pollutant, wind field, adaptive grids) from the data base into the blackboard. The viewer satellite will read some of the data, write a synchronization variable, and render the data frame. The playback satellite will wait until the synchronization variable is mutated and dump the next data frame.

Figure 4.2 (see next chapter) shows a rendition of the data frame. The amount of data read by the viewer is approximately: $4 \times 52 \times 55 = 11.440$ floating point numbers per data frame. This amounts to approximately 45 KBytes per data frame.

Since the playback and viewer satellites are running synchronized, it is sufficient to measure the number of frames per second the viewer satellite can render. Figures 3.8, 3.9, 3.10 show the results of 6 configurations. Each plot shows the performance of the local configuration in which the playback satellite and viewer satellite reside on the same machine (solid line), and the performance of the remote configuration in which the playback satellite and viewer satellite reside on the different machines (linespoints). Performance is measured in frames per second (vertical axis) and the horizontal axis enumerate the data frames. 100 Tbase ethernet was used for the network in the case of distributed blackboards. Shared memory was used in the case of a local blackboard.

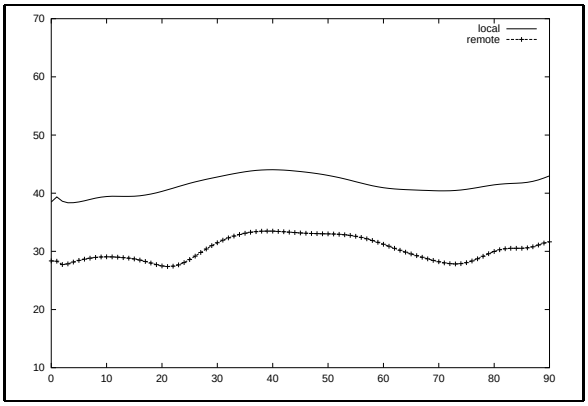


Figure 3.8: Local playback and viewer on 4 CPU SGI Onyx2 (solid line), remote playback on SGI O2000 and viewer on SGI Onyx2 (linespoints).

The plot in figure 3.8 shows the performance of a viewer on a high-end graphics workstation; a 4 CPU SGI Onyx2 with Infinite reality graphics. The Infinite reality graphics system accelerates the geometry transformations and rasterization. When executing the local configuration, the blackboard, viewer satellite and playback satellite can execute on any of the 4 CPUs.

The plot in figure 3.9 shows the performance of a viewer on a mid-range graphics workstation; a single CPU SGI Indigo-2 with impact graphics. The Impact graphics subsystem accelerates the geometry transformations and rasterization. The graphics subsystem can be seen as a co-processor which can off-load the rendering functions from the main CPU. When executing the local configuration, the blackboard, viewer satellite and playback satellite will share the CPU.

The plot in table 3.10 shows the performance of a viewer on a low-end graphics workstation; a single CPU SGI O2. The O2 graphics system accelerates only the rasterization. Geometry transformations are done by the main CPU. When executing the

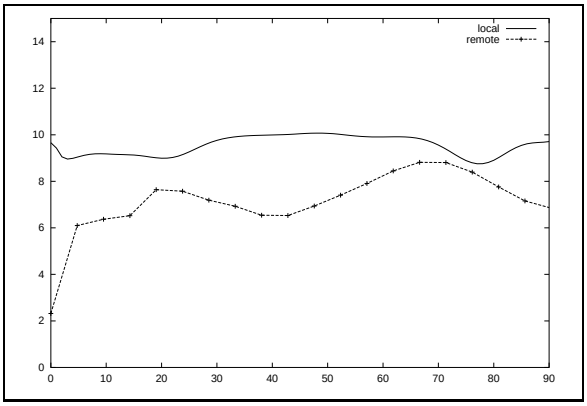


Figure 3.9: Local playback and viewer on SGI Indigo Impact (solid line), remote playback on SGI O2000 and viewer on SGI Indigo Impact (linespoints).

local configuration, the blackboard, viewer satellite and playback satellite will share the CPU. For the remote configuration, the blackboard and viewer satellite will share the CPU.

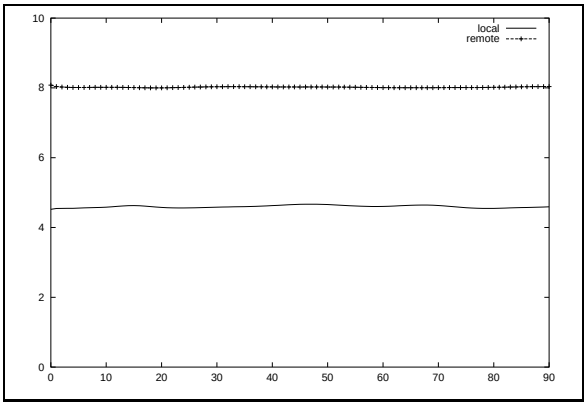


Figure 3.10: Local playback and viewer on SGI O2(solid line), remote playback on SGI O2000 and viewer on SGI O2 (linespoints).

There are many factors that influence the performance of the measured configuration; the number of CPUs on the workstation, the network bandwidth and latency, the absolute speed of the graphics subsystem, the various levels of acceleration on the graphics subsystem, etc. However, a number of general observations related to the blackboard architecture can be given:

- The average frame rates for the local configurations in tables 3.8, 3.9, 3.10 were 40, 9 and 4 respectively. The average frame rates for the remote configurations were 30, 8 and 8 respectively.

- The remote frame rate in table 3.10 is higher than the local frame rate. In this configuration all geometry transformations are done by the main CPU. The amount of work done by the CPU is (i) to read the data from disk and write the data to shared memory (the playback satellite), (ii) manage the shared memory data arenas (the blackboard), (iii) read the data from shared memory and perform the geometry transformations (the viewer satellite). For the remote configuration, the amount of work done by the CPU is (i) manage network and the shared memory data arenas (the blackboard), (ii) read the data from shared memory and perform the geometry transformations (the viewer satellite). In this configuration, the CPU time required to perform network access and for the operating systems to manage two processes outweighs the disk access and for the operating systems to manage three processes.

- Interleaving the playback satellite, blackboard, and rendering satellite will increase performance. Interleaving can be achieved in the case of a multi-processor or distributed system.

- Off-loading the CPU with a graphics subsystem will increase interleaving, since the CPU can process the next frame while the graphics subsystem is rendering the current frame.

- The frame rate fluctuations on remote configurations are larger than the frame rate fluctuations on the local configurations. This may be due to additional network latency and contention.

3.4 Retrospective Assessment

The three aspects that motivated the design of the architecture were: (i) the flexibility of the data model, (ii) efficiency of data transport, (iii) the ease of integration into legacy applications, and (iv) usability. Each aspect is assessed:

- Flexible data model.

  Data is stored in the blackboard as an n-dimensional array. The data mapping schemes from simulation data structures onto blackboard structures avoids the necessity of temporary copies of data. Flexible mappings are particularly useful for legacy HPCN applications. These applications usually have data structures that cannot be changed easily. Furthermore, additional copies of data are not desired.

- Efficiency of data transport.

  An important aspect of distributed visualization systems is when and how often data is transported over the network. The blackboard architecture uses a demand

driven transport scheme, by sending data over the network only when it is needed in a different blackboard.

The demand driven transport scheme is efficient in the cases where (for example) the update frequency of a simulation outpaces the update frequency of a remote render satellite. In these cases, only the latest data will be transported. Demand driven transport is substantially different from data flow architectures, in which each new data set is transported downstream.

- Ease of integration.

  The rapid binding of legacy code to the blackboard and taking advantage of the CSE's utilities to analyze the data is an essential aspect of incrementally gaining more insight to the simulated model. The Fortran binding of the data input/output API can be used for this. Users need only initialize the bi-directional binding of data structures with variable names in the blackboard. Updating the bind is realized with one statement in the computation main loop of the simulation.

- Usability.

  The CSE has been used for the development of other computational steering applications and techniques. Other applications can be found in [3, 4]. The CSE has been extended with various techniques [5, 6].

The implementation of the blackboard has been designed to be portable and open. Portability has been realized by executing the blackboard in a heterogeneous environment consisting of machines with different word length and endianess. The open implementation has been realized by separating the data modeling functionality from the data transport functionality. The data transport level will require an implementation of a robust byte stream. The current implementation uses either TCP sockets or shared memory at this level. Although it has not been implemented, the design allows other message passing packages (e.g., MPI, PVM) to be used at this level. In addition, the open implementation also allows other data modeling packages to be used to implement the supported data types. For example, the Hierarchical Data Format [7] can be used at this level.

References

[1] J.J. van Wijk and R. van Liere. An environment for computational steering. In G.M. Nielson, H. Müller, and H. Hagen, editors, *Scientific Visualization: Overviews, Methodologies, and Techniques*, pages 89–110. Computer Society Press, 1997.

[2] J.D. Mulder and J.J. van Wijk. 3D computational steering with parametrized geometric objects. In G.M. Nielson and D. Silver, editors, *Proceedings Visualization '95*, pages 304–311. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[3] Jack van Wijk, Robert van Liere, and Jurriaan Mulder. Bringing computational steering to the user. In F. Post H. Hagen, G. Nielson, editor, *Scientific Visualization - Dagstuhl '97 Proceedings*, pages 304–313. IEEE Computer Society Press, 2000. ISBN 0-7695-0503-1.

[4] J.J. van Wijk, H.J.W. Spoelder, W.J. Knibbe, and K.E. Shahroudi. Interactive exploration and modeling of large data sets: A case study with venus light scattering data. In R. Yagel and G.M. Nielson, editors, *Proceedings IEEE Visualization '96*, pages 433–438. IEEE Computer Society Press, 1996.

[5] J.J. van Wijk. A model for strategy in constraint solving. Technical Report SEN-R9714, Center for Mathematics and Computer Science, 1997.

[6] K.E. Shahroudi. Design by continuous collaboration between manual and automatic optimization. Technical Report SEN-R9701, Center for Mathematics and Computer Science, 1997.

[7] Hierarchical data format. Technical report, NCSA, 1999. http://hdf.ncsa.uiuc.edu.

Chapter 4

Steering Smog Prediction [1]

The use of computational steering for smog prediction is described. This application is representative for many underlying issues found in steering high performance applications: high computing times, large data sets, and many different input parameters.

After a short description of the smog prediction model, its visualization and steering are described. The amount of computation needed to solve the governing transport equations is alarmingly high. The user has a large number of options for the display of various aspects of the simulation, and also for the interactive control of its input data.

Smooth animation is very important to monitor the evolution of pollutants and for a responsive feedback to parameter changes. Here a performance of least 15 frames per second is required. We discuss techniques that allow the user to steer the numerical solver, such that an optimal tradeoff between computation speed and accuracy can be made.

4.1 Introduction

The word 'smog' is a combination of 'smoke' and 'fog'. It was originally used to describe city fogs containing large amounts of air pollutants. Environment protection agencies, such as the Dutch National Institute of Public Health and Environmental Protection (RIVM), provide detailed forecasts of the expected levels of smog. A variety of smog prediction models have been developed to determine these forecasts. Solving smog prediction models is a very time consuming process and results in large amounts of output data. However, due to the advent of high performance computing and better numerical algorithms, these computational costs are decreasing.

The next challenge is to apply computational steering to smog simulation. Computational steering allows the user to change simulation parameters while the simulation is in progress and its results are visualized. Computational steering enhances productivity by reducing the time between changes to model parameters and the viewing of

[1] This chapter has been published as: R. van Liere and J.J. van Wijk, Steering Smog Prediction. In B. Hertzberger and P. Sloot, *HPCN-Europe '97*, pages 241–252, 1997.

the resulting output. This simplifies the study of alternative scenarios, and increases insight into the model and the effect of parameter changes. For example, a user may want to investigate the effect that varying emission levels have on the computed pollutants. Another example is to study the effect of varying meteorological conditions, such as alternative wind fields.

In this chapter we demonstrate how computational steering is applied to a smog prediction model. The chosen application is of particular interest to the HPCN community because it is representative for many underlying issues found in steering high performance applications: high computation times, large data sets, and varying inputs. The original model was developed at RIVM [1], and the numerical solver was developed at CWI [2]. We have embedded the simulation into the Computational Steering Environment, CSE [3] and parallelized the solver so that it would take advantage of high performance SMP architectures.

In [3], a very simplistic smog simulation was used to illustrate some basic concepts of the CSE. The smog simulation used here is different in many ways: it uses a much more realistic chemical model, it requires orders more computation power, and generates much more output. Also, the visualization is 3D and has many more steering possibilities. One of the goals in this work was to provide smooth animation of the simulation in progress. At least 15 frames per second are required because only then will the user be able to monitor the evolution of a pollutant.

The format of this chapter is as follows: first we briefly present the underlying smog model and solver which is used to compute the model. We discuss the complexity of the problem and show why high performance computing is mandatory to interactively solve the governing equations. In Section 4.3, we present the visualization of and interaction with the simulation. In Section 4.4 we discuss implementation and performance issues. Finally, we draw some conclusions.

4.2 Smog Simulation

4.2.1 The Physical and Chemical Model

The simulation forecasts the levels of air pollution which is modeled by 15 pollutants, such as ozone (O_3), sulphur dioxide (SO_2) and nitrogen dioxide (NO_2).

For the geographical domain a model of Europe is used. The vertical stratification is modeled by four layers; the surface layer, the mixing layer, the reservoir layer, and the upper layer.

The governing equations of the model are described by a set of partial differential equations that determine the advection, diffusion, emission, wet and dry deposition, fumigation, and chemical reactions. In spherical coordinates, the full model equation

is written as

$$
\begin{aligned}
\frac{\partial c}{\partial t} &= -\frac{1}{r\cos\theta}\left[\frac{\partial(uc)}{\partial\phi} + \frac{\partial(vc\cos\theta)}{\partial\theta}\right] && \text{advection}\\
&+ \frac{\kappa_{diff}}{r^2\cos\theta}\left[\frac{1}{\cos\theta}\frac{\partial^2 c}{\partial\phi^2} + \frac{\partial c}{\partial\theta}\left(\frac{\partial c}{\partial\theta}\cos\theta\right)\right] && \text{horizontal diffusion}\\
&+ \frac{\partial}{\partial z}\left(K_z(z)\frac{\partial c}{\partial z}\right) && \text{vertical diffusion}\\
&+ S_d(c) && \text{dry deposition}\\
&+ S_w(c) && \text{wet deposition}\\
&+ F(c) && \text{fumigation}\\
&+ Q && \text{emission}\\
&+ R(c), && \text{chemical reactions}
\end{aligned}
$$

in which c denotes the vector of computed pollutants. The pollutant concentration in a layer is interpreted as the average value over that layer in vertical direction.

The chemical model describes summer smog episodes, which are characterized by high levels of ozone concentrations. The summer smog model is defined as 17 reactions between 15 different pollutants. Numerically, these reactions are defined as a set of 17 stiff ordinary differential equations.

There are many parameters that control the input conditions of the model. We name a few:

- Emission fields.

  Emissions are divided into 6 categories: emission due to combustion, space heating, refinery, chemical processes, solvents and traffic. The model distinguishes emission point sources and emission surface sources.

  Short and middle term predictions can be made by changing emissions, e.g., the effects of shutting down a power plant (modeled as a point source) can be studied.

- Meteorological parameters.

  Examples of meteorological parameters are wind fields, temperature, cloud coverage, humidity, etc. Wind fields are read from a data base provided by RIVM.

  Short term predictions can be made by changing meteorological parameters. For example, a user can study the effect of different weather forecasts.

- Geographical information.

Geographical information, such as land-sea boundaries and forest densities influence diffusion, wet and dry deposition, fumigation, and the chemical reactions processes.

Long term predictions can be made by changing geographical information. For example, to study the effect of deforestation on pollution.

4.2.2 The Solver

The solver uses an adaptive grid refinement technique to improve the efficiency of the model calculations. The tradeoff to be made in local grid refinement is calculation accuracy versus computation speed.

The geographical domain is discretized into a base grid with a resolution of 52×55 cells. When a grid cell is refined it will be split uniformly into 4 smaller grid cells. The base grid is the grid at level 1. A new grid at level N is constructed by refining grids cells at level $N - 1$. Figure 4.1 illustrates grid refinement: 6 cells a1, a2, b1, b2, b3, c3 are refined into 24 new cells. The 6 grid cells define the grid at level 1. The 24 new cells define the grid at level 2.

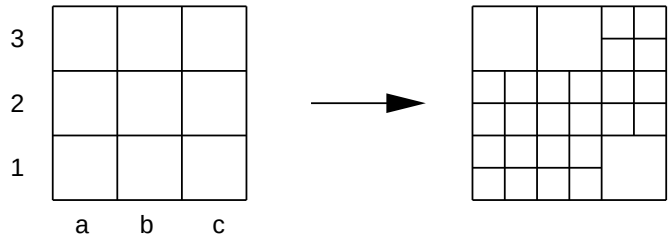


Figure 4.1: Adaptive refinement of a grid cell.

The governing equations are solved for each grid cell. Hence, for each time step, the maximum amount of work to be done is proportional to the number of grid cells :

$$\sum_{i=1}^{maxlevel} 4^{i-1} \times L \times 52 \times 55$$

in which L are the number of stratification layers. Currently L is set to 4, but this can be increased.

The amount given above is clearly the worst case situation in which the complete grid is refined until $maxlevel$ is reached. Fortunately, usually only a fraction of the maximum amount of work is done. For example, if $maxlevel = 4$, test cases have shown that the total amount of work to be done is only 13% of the number mentioned above. Higher levels of grid refinement will result in more relative savings.

The solver uses several criteria to automatically determine if and where adaptive grid refinement is to be performed. For example, grid refinement is applied to cells

where pollutant concentrations have large spatial gradients. However, these general purpose criteria do not incorporate knowledge about particular interests of the user. Hence, the user has to be enabled to steer the refinement criteria. There are various parameters that control the grid refinement algorithm: First, the tolerance level which determines when a cell should be refined. Second, a region of interest which determines where grid refinement should be performed. Third, $maxlevel$ which determines the depth of refinement. The user can select optimal values for all parameters here: the best tradeoff between accuracy and speed of computation, possibly varying per region of interest.

4.3 Visualization and Steering

The visualization and interaction is designed to be performed on a graphics workstation remote from a compute server. After each time step of the simulation the compute server will send all necessary data over the network for rendering. Upon interaction the graphics workstation will send new parameter values to the compute server.

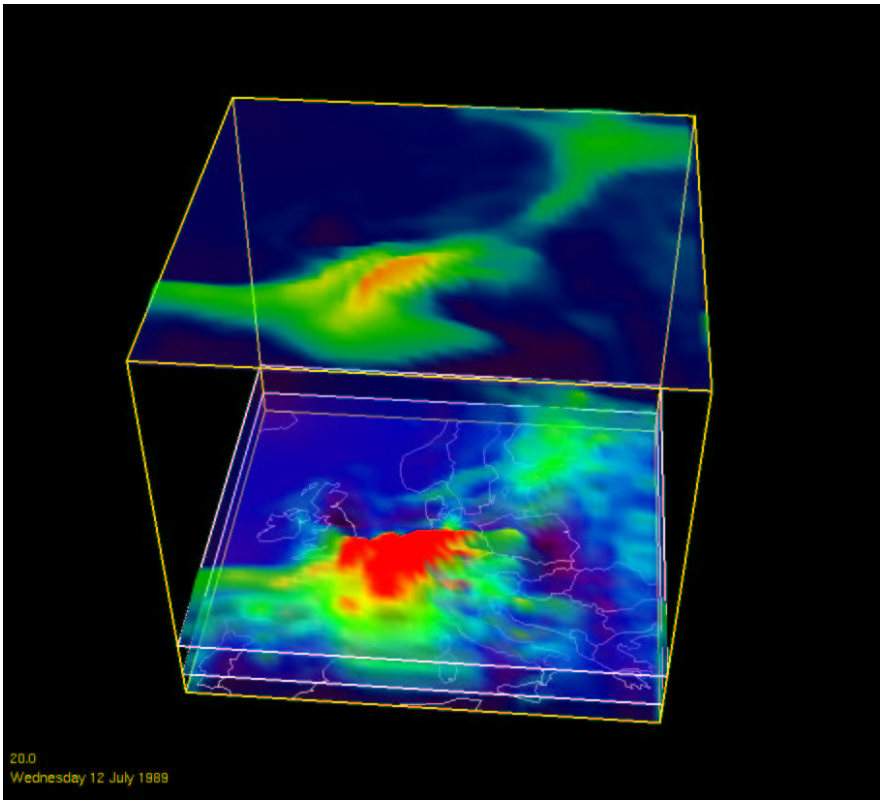


Figure 4.2: Four layers of ozone concentration over Europe. (see color plate on page 151)

- *Display*

  Figure 4.2 shows a snapshot of a time step in the simulation of a O_3 concentration field on each of the four layers. A Gouraud shaded colored mountain plot is used to display the concentration of a pollutant. Color and height are assigned according to the concentration at the corresponding grid cell. Transparency is used for the simultaneous display of all four layers; the amount of opacity is proportional to the concentration.

  The wind field is shown via small vectors per grid cell. This representation will clearly display the changes of the wind field if the frame rate is high enough. Each level of grid refinement is displayed as a uniquely colored mesh. Successive grid levels are rendered on top of the previous level.

  Figure 4.3 shows a different view of a time step. Only two of the four layers are shown. Three levels of the grid refinement are shown in the surface layer. Level 2 grids are shown in red, level 3 grids in green, and level 4 in blue. In this way the user can clearly see how the grid is refined. The wind field is shown in the upper layer.

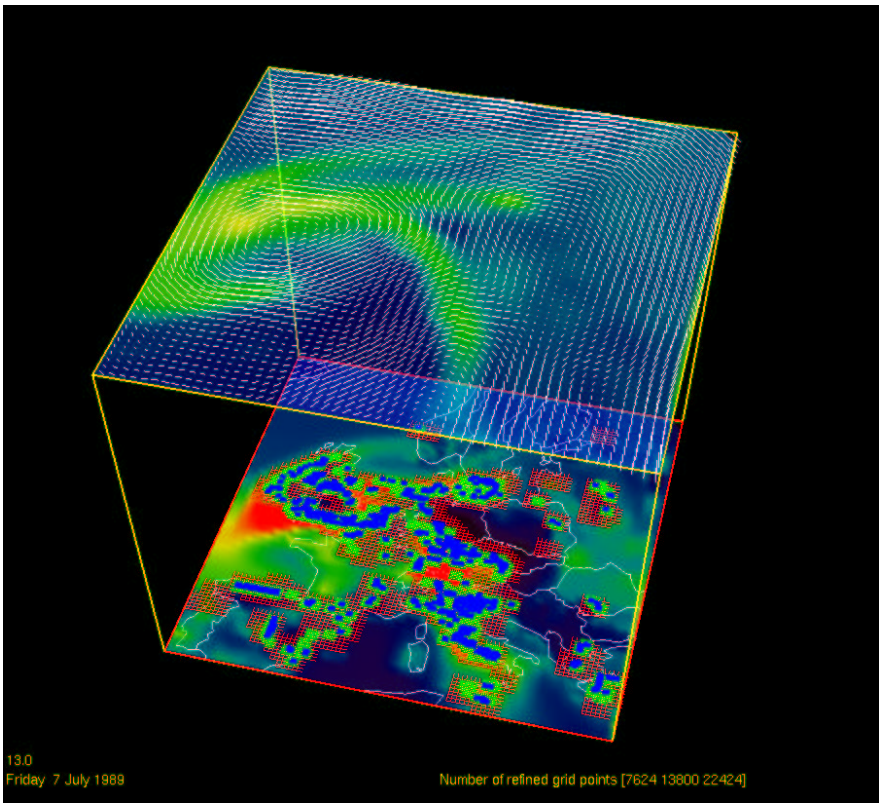


Figure 4.3: Ozone with grid (surface layer) and wind fields (upper layer). (see color plate on page 151)

Animated sequences of these visualizations reveal several important features. The evolution of individual pollutant concentrations over time can be monitored. Also, monitoring the grid refinement algorithm over time can provide valuable information for the simulation developer. The user can monitor these features only if the frame rate is high enough. We experimentally found that at least 15 frames per second were required to give the illusion of smooth motion. Below this threshold only discrete images were observed, and the motion was much less obvious.

The user can perform interaction for viewing purposes: selectively display one or more pollutant concentration layers, a wind field or a grid mesh. The mouse is used for zooming and rotation. Finally, a color map editor for mapping the value range onto color and transparency is available.

- *Steering*

  The user can steer various aspects of the model and solver. First, emission field editors have been developed that allow the user to control the emissions. The editors provide a direct manipulation interface to model emission parameters, in which geometric representations are bound to simulation data. The user may manipulate the geometry resulting in mutations of simulation data. The user can change a emission point source by dragging its position or changing the emission values. Also, filters may be defined over a geographical region which damp or amplify the emission surface sources.

  Second, several parameters that control the grid refinement algorithm are accessible through a direct manipulation interface. The user can change the tolerance of the grid refinement with a slider. Also, a region of interest defines the area in which grid refinement is to be performed. Finally, the maximum number of levels that determine the grid refinement can be set.

  Figure 4.4 shows a snapshot of the three editors. Users can directly manipulate various aspects and immediately view the effects. The emission field editors are used for cause/effect analysis; e.g., what would be the effect on the ozone concentrations when an emission changes? Steering the grid refinement process allows the user to make a tradeoff between speed and accuracy of the simulation. For example: when nothing “interesting” is happening in the simulation, the user may wish to compute with large grid resolutions.

- *Record and Playback*

  A record and playback facility is available which the user may turn on or off. In record mode a selection of pollutants, wind fields and grid levels will be dumped into a file. In playback mode the file will be read and rendered. The big advantage of the playback mode is that smooth animation frame rates can easily be realized.

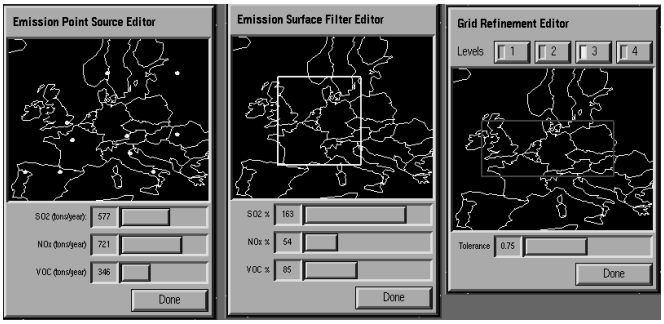


Figure 4.4: Three editors; point source (left), surface source (middle), grid refinement (right).

The playback facility may also be used in lock step with the simulation. In this case two renderers are set up to display the recorder and the simulation. Combining the playback facility with steering allows the user to experiment with alternative scenarios and compare results simultaneously.

4.4 Implementation and Performance

4.4.1 Computational Steering Environment

The Computational Steering Environment [3], CSE , is a client/server based architecture for computational steering. It provides many general purpose tools that allow the user great flexibility in defining visualizations.

An overview of the architecture of the CSE is shown in figure 4.5. The architecture is centered around a *Data Manager* that acts as a blackboard for communicating values. Separate processes (*satellites*) connect to the Data Manager and exchange data with it. The simulation, record/playback and renderer are all packaged satellites.

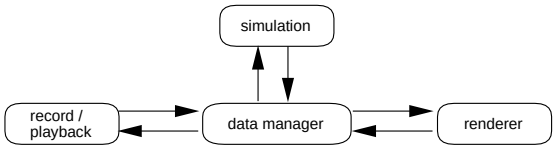


Figure 4.5: The CSE architecture

The purpose of the data manager is twofold. First, it manages a database of variables. Satellites can create, open, close, read, and write variables. For each variable the data manager stores a name, type, and value. Second, the data manager acts as an event notification manager. Satellites can subscribe to state changes in the data manager. When such a state change occurs the satellite will receive a notification from

the data manager. For example, when a satellite subscribes to mutation events on a particular variable, the data manager will send a notification to that satellite whenever the value of the variable is mutated.

The CSE is implemented as a network transparent environment, which uses TCP/IP for data movement over machine hosts. The user may choose to execute a satellite on any host. However, the most natural configuration would be to run the render satellite on a graphics workstation and the simulation satellite with the data manager remotely on a high performance compute server.

The simulation, record/playback and renderer satellites are custom satellites implemented specifically for this smog simulation. Performance was the main reason for us to choose for implementing custom satellites. The standard satellites provided by the CSE were not able to cope with the requirement of 15 frames per second.

4.4.2 Performance

Performance is analyzed by considering the computation of the simulation, transport from compute server to rendering workstation, and rendering separately.

- Computation.

  As noted in Section 4.2.2, the amount of work needed to be done in the simulation is proportional to the total number of grid cells. For each grid cell the chemical reactions, formulated as a set of stiff ODE's, must be solved.

  For example, if the maximum level of grid refinement is three, the number of cells per time step to be solved will be proportional to:

  $$4 \times\ 52 \times 55\ +\ 4 \times 4 \times 52 \times 55\ +\ 16 \times 4 \times 52 \times 55\ \approx\ 250K$$

  Hence, if the goal is 10 time steps per second, then approximately 2.5M sets of stiff ODE's, each consisting of 17 equations, must be solved. This would be the worst case. Fortunately, since the user can control when and where refinement should take place, more reasonable figures can be realized.

  The following table shows the performance of the simulation in time steps per second on the SGI Origin2000. The tolerance value and refinement region were fixed throughout the simulation. This table clearly shows that the performance is scalable for the number of available CPUs.

- Data movement.

  The amount of data sent from the data manager to the renderer is proportional to the number of cells that have been computed. Each pollutant is represented as a scalar field. Thus, assuming a floating point number is represented by 4 bytes, a bandwidth of $4 \times N$ bytes is required if N is the number of calculated cells. In

CPUs	SGI Origin2000
1	0.9
2	1.7
4	3.1
8	5.4

Table 4.1: Performance measured in steps per second.

addition, for each time step, new wind fields and grid meshes will be sent to the renderer.

Assuming 15 frames per second, the worst case will saturate even a HIPPI channel. However, when under user control, an ATM OC-3 would seem sufficient.

- Rendering.

  Data is rendered by OpenGL as quadrilateral strips. Each pollutant layer consists of 52 strips of 55 quadrilaterals for each layer. Many high end graphics workstations are able to handle these quantities at 30 frames per second.

In summary, performance is hampered by the simulation. We do not consider this a very big problem because the simulation is inherently scalable. With additional processing power we are confident that we can reach approximately 15 frames per seconds. At higher frame rates the transport of the data will become the performance bottleneck. The renderer can easily keep up with the performance requirements. Only if different visualization techniques are used (such as volume rendering) then the renderer may cause a bottleneck.

4.5 Conclusions

We have shown how a realistic and high performance smog prediction simulation is steered. Computational steering increases insight to smog forecasting. Steering combined with playback facilities provide an effective means to compare alternative scenarios.

The amount of computation needed to solve the governing transport equations is alarmingly high. Fortunately, the user can steer the underlying numerical solvers, resulting in a tradeoff between calculation accuracy versus computation speed.

The performance goals have not yet been met. Smooth animation is important for monitoring the evolution of a pollutant. The bottleneck is currently the lack of resources on the compute server. However, since the simulation code seems to be scalable, we are confident that the goal of 15 frames per second can be met.

Future work will include extending the model to 32 layers, providing additional steering possibilities, and implementing more visualization techniques such as simultaneously displaying multiple pollutants.

Acknowledgements

Maarten van Loon of CWI provided the adaptive grid solver discussed in Section 4.2.2. The solver was implemented as a single threaded batch process in Fortran. The computed pollutants were analyzed using 1D-plots.

In order to meet the steering requirements and to integrate the solver in the CSE, I have adapted it in a number of ways. This required parallelization of the solver and the interfacing of the connected variables with the blackboard.

References

[1] H. Jetske van Rheineck Leyssius, F. de Leeuw, and B. Kesseboom. A regional scale model for the calculation of episodic concentrations and dispositions of acidifying components. *Water, Air and Soil Pollution*, 51:327–344, 1990.

[2] M. van Loon. *Numerical Methods in Smog Prediction*. PhD thesis, University of Amsterdam, June 1996.

[3] R. van Liere and J.J. van Wijk. Computational Steering. In H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking*, pages 696–702, Brussels, April 1996. Springer-Verlag.

Part II

Virtual Reality

Chapter 5

PVR: An Architecture for Portable VR Applications [1]

Virtual reality shows great promise as a research tool in computational science and engineering. However, since VR involves new interface styles, a great deal of implementation effort is required to develop VR applications.

In this chapter we present PVR: an event based architecture for portable VR applications. The goal of PVR is to provide a programming environment which facilitates the development of VR applications. PVR differentiates itself from other VR toolkits in two ways. First, it decouples the coordination and management of multiple data streams from actual data processing. This simplifies the programmer's task of managing and synchronizing the data streams. Second, PVR strives for portability by shielding low level device specific details. Application programmers can take full advantage of the underlying hardware while maintaining a single code base spanning a variety of input and output device configurations.

5.1 Introduction

Virtual reality shows great promise as a research tool in computational science and engineering. The analysis of 3D data sets (read from disk or computed on the fly) may benefit from the interface styles provided by virtual reality. By providing additional depth and viewing cues, the virtual reality interface can aid the unambiguous display of 3D structures. In addition, virtual reality input interfaces allow the direct and intuitive exploration of the data, as well as providing control over the application through widgets integrated into the interface.

However, the development of VR applications is not easy. In order to fulfill the real-time requirements imposed on VR applications, programmers must learn many

[1] This chapter has been published as: R. van Liere and J.D. Mulder, PVR - An Architecture for Portable VR Applications. In M. Gervautz and D. Schmalstieg, *5th EG Workshop on Virtual Environments*, pages 125–135, 1999.

new and nonstandard methods. Technical issues that make these methods more difficult than traditional 3D graphics methods are mostly related to: (i) the management of multiple I/O data streams at predictable and time critical intervals, (ii) 3D interaction with the virtual world, and (iii) the large variety of display and input devices with each device having individual characteristics and constraints. Virtual reality development environments will need to provide high level support for each of these issues, as well as support found in traditional 3D graphics toolkits, such as for modeling and rendering.

Event-driven programming models are well known to graphical user interface programmers. Instead of explicitly polling and responding to user actions, applications register event handlers and the underlying user interface toolkit will call the handler when a particular event occurs. Event-driven models can ease the programmer's task since the control and coordination of the user interface objects is left to the underlying toolkit.

In this chapter we present the design and implementation of a portable virtual reality architecture, PVR. The architecture was inspired by our work on distributed computational steering [1]. The motivation for designing and implementing PVR was to provide an application programmer an easy to use and portable development environment. Ease of usage is accomplished by decoupling the coordination of multiple data streams from actual data processing. This simplifies the programmer's task of managing, synchronizing and processing multiple I/O streams. Portability is realized by allowing the programmer to abstract from specific input and output devices.

The format of the chapter is as follows. First, in Section 5.2, we briefly survey related work. In Section 5.3 we present the underlying concepts of the PVR architecture and discuss design considerations. Finally, we provide two examples on how the environment is used: a molecular dynamics visualization and the analysis of nuclear division.

5.2 Related Work

Bryson has extensively studied the application of virtual reality interfaces in scientific visualization. Although the work has mostly been related to studies in the virtual wind tunnel, the lessons have led to generalized requirements with regard to implementation issues concerning computation, graphics and data management (e.g., [2, 3]). Requirements related to real-time performance and natural "anthropomorphic" VR interfaces are discussed in some detail.

Numerous academic and commercial efforts have focussed on the development of distributed virtual reality applications. Often these efforts have resulted in generic toolkits; many of these have similar objectives and motivations as PVR. For example, a non-exhaustive list of widely used VR toolkits is: Alice [4], AVOCADO [5], Bamboo [6], COVISE [7], DIVE [8], Lightning [9], and World Toolkit [10]. It is beyond the scope of this chapter to give an exhaustive overview of each of these toolkits.

The MR toolkit [11] was one of the first toolkits to decouple the simulation from

the rendering in order to achieve the real-time requirements of VR user interfaces. MR provides many high level features that are needed in the development of VR applications: support for distributed computing, workspace management, performance monitoring, input devices abstractions, data sharing, etc.

Researchers at IBM have used multiple workstations to support the real-time requirements of VR user interfaces [12, 13]. Their virtual world architecture addresses the requirements of virtual reality for high performance computing, concurrency and synchronization of multiple events, and flexibility. Processes communicate through a central event-driven user interface management system (UIMS). The UIMS uses a collection of rules to synchronize concurrent input/output events. It also provides flexibility by clearly separating the interaction techniques from the virtual world application.

The CAVE library is an API of C functions and macros to control the operation of the CAVE [14]. The CAVE library takes care of all the tasks that have to be performed to correctly operate the CAVE. CAVE functions keep all the devices synchronized, produce the correct perspective for each wall and provide applications with the current state of all CAVE elements.

Despite similar objectives and functionality, PVR differs from these toolkits in many ways. First and foremost is the emphasis of PVR on the ease of usage for the VR application programmer. PVR provides a modular architecture that shields the management and synchronization of multiple data streams. This is achieved by decoupling the coordination and management of multiple data streams from actual data processing. Programmers need only to supply the processing modules, while the coordination of the data streams is maintained by PVR. Second, PVR strives for portable applications. Portability goes beyond portability of low level input devices. The PVR architecture allows flexible mapping from high level interaction abstractions onto lower level device abstractions. Also, although not unique, PVR allows for the flexible mapping of logical to physical output devices.

5.3 PVR Architecture

5.3.1 Internal Structure of PVR

An overview of the PVR architecture is given in figure 5.1. The architecture consists of one or more processes attached to a bus. Processes communicate using an event-based mechanism. This mechanism is conceptually easy to understand: after a process attaches to the bus, it registers patterns with the bus that describe the events it is interested in. This allows each process to set an *event filter*. When an event is posted on the bus, it is redistributed to the interested processes. Each process is parameterized with a user supplied callback. The bus will invoke the callback whenever an event occurs that the process is subscribed to.

PVR supports a small number of predefined events, such as `REDRAW`, `MOTION`,

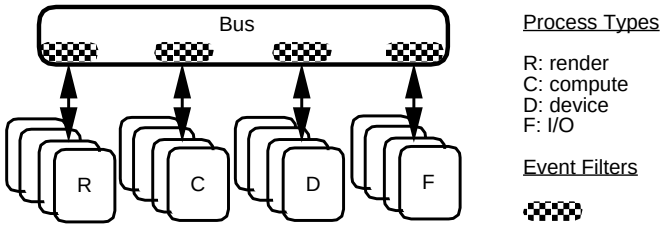


Figure 5.1: The PVR architecture consists of one or more processes attached to a bus. Each process registers event filters that describe events that the process is interested in. A process posts events to the bus to signal the completion of an action. The bus redistributes events to processes according to their event filters.

PRESS, etc. In addition, applications may define events to denote any application specific action.

Processes are categorized in one of four types:

1. Render processes are responsible for rendering. The application defined callback will be executed when a redraw is necessary. Often, particular output device configurations will require more than one render process; i.e., the CAVE will use one process per wall whereas a head mounted display (HMD) might have two.

2. Device processes are responsible for device handling. An event will be posted each time the state of a device changes (i.e., button press, sensor motion, etc). The event report will contain the information of the device.

3. File processes are responsible for all file I/O. Having designated file processes is useful, since I/O tends to be very slow. File processes are also responsible for I/O transactions with external services. For example, performing transactions with data base management systems, CORBA servers, or external simulations.

4. Compute processes are responsible for all other required computation. This may be the complete simulation itself, or the computation of visualization techniques (streamlines, iso-surfaces, etc). It is the programmer's choice to determine the granularity of the compute processes.

A shared data facility allows multiple processes to share data structures. This facility allows processes to allocate and lock shared data structures in a simple and flexible way. Pointers to shared data structures can be stored in events.

5.3.2 PVR's Mechanisms

Rather than discussing the architecture in detail, we highlight three mechanisms unique to PVR.

1. Device independence.

   Virtual reality environments may have a plethora of display, tracking, and input devices, each operating in their own physical coordinate space. PVR is designed to be portable, allowing applications to take full advantage of the underlying hardware while maintaining a single code base spanning multiple device configurations. For example, applications are developed independently of the tracker geometry, room geometry and device configurations.

   PVR uses the notion of a workspace that removes the application's dependency on the physical location of the devices that it uses. Workspace management functionality provides two sets of transformation mappings; a mapping between application workspace and standard workspace and a mapping between standard workspace and device workspace. These two mappings allow applications to use device independent input and output spaces. The mapping between standard workspace and device workspace is defined in a configuration file. The application can define the mapping between application workspace and standard workspace. PVR events encapsulate device information, so processes that register device events are device independent.

2. Synchronization and Data Ordering.

   Events are used by a process to signal the end of an action. In this way a sequence of processes can be set up; each process sends an event to the next process in the sequence. However, when processes operate on one data set in parallel, additional mechanisms must be provided. For example, consider the case that one process computes iso-surfaces and a second process computes streamlines in a time dependent data set. Both geometries must be rendered in the same frame.

   PVR provides a data framing mechanism to order data from parallel processes. Applications may define a frame structure which consists of a number of data slots and an event. Processes may put data in a slot. As soon as all slots are filled the bus will distribute the event. This signals that the frame is complete. The familiar notion of a barrier can be implemented with the framing mechanism.

   A time line for a frame with four slots is diagrammed in figure 5.2. Processes may fill data into a slot at any time and in any order. Only when the last slot is filled will the bus invoke the callback of those processes that have registered EVENT.

   The bus can also manage a stack of frames. If a process refills a slot before the frame is filled, the bus will place the data in the next frame in the stack. In this way processes do not need to wait until a frame is complete, but can continue to process new data.

3. Distributed Virtual Reality.

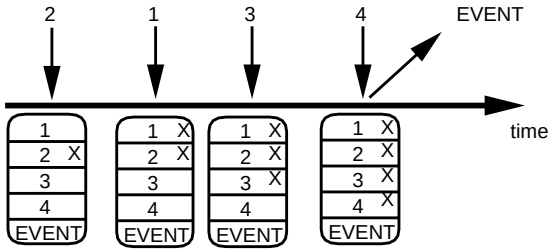


Figure 5.2: A frame with four slots and event EVENT. Processes add data to a slot. The bus will distribute EVENT as soon as all slots have new data, signaling a new frame.

In addition to the single user virtual reality, multi-user distributed virtual reality can provide added value in those cases where collaboration is needed. Two cases are distinguished. First, multiple remote environments are linked together to form one single distributed environment in which users have the ability to manipulate a common data set. Second, multiple users can participate in a single environment.

PVR includes a mechanism for distributed applications, whereby several users on different machines can share a virtual environment. This is achieved by connecting remote busses together. A posted event will be redistributed to a remote bus, according to event registration patterns on the remote bus (see figure 5.3).

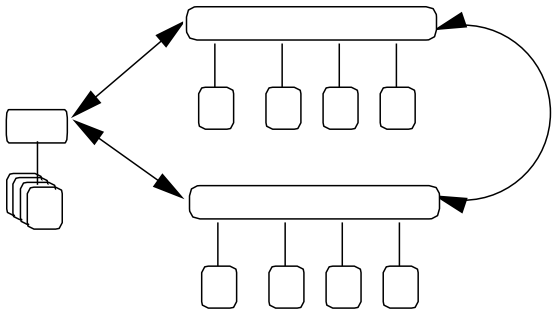


Figure 5.3: Multiple busses are connected to form a distributed architecture. Events are redistributed according to event filters on the remote bus.

From the perspective of a process, it makes no difference whether the application is distributed. The bus will invoke the process's callback whenever an event occurs that is designated for the process, irrespective of whether the event was posted from a process on the local or remote bus.

5.4 Applications

5.4.1 Molecular Dynamics Visualization

The application is a molecular dynamics simulation of a mixture of molten lithium and potassium carbonates at 1200 Kelvin[2]. The movements of 158 Li^+ ions, 98 K^+ ions and 128 CO_3^{2-} ions in a cubic box are calculated during 2500 time steps. Of particular interest is the study of the coordination number; this is the number of CO_3^{2-} ions in a neighborhood of a Li^+ or K^+ ion. Typically, Li^+ ions are enclosed by 4 carbonates, and the K^+ ions by 6 carbonates.

Figure 5.4 shows two snapshots of the interface. Small spheres are used to display Li^+ions, triangles are used to display the carbonate ions. A convex hull is used to visualize the coordination between a Li^+ ion and related CO_3^{2-} ions. The dashed line shows a measuring tape (each line segment represents 0.25 nm.), which is used to monitor distances between ions.

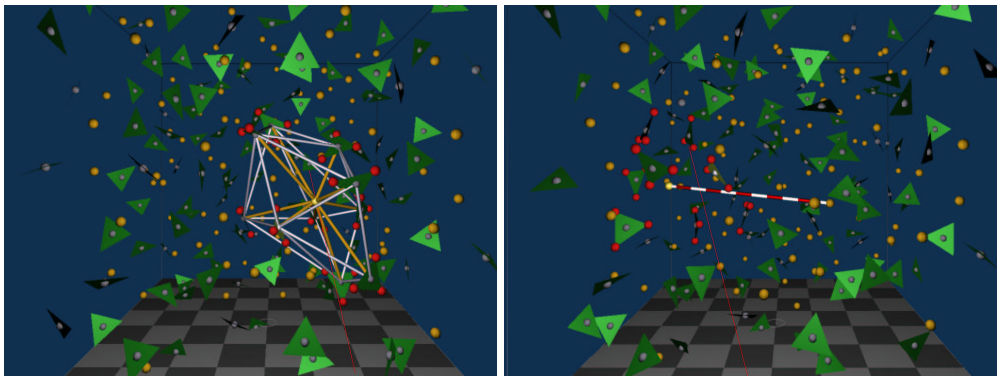


Figure 5.4: Molecular dynamics of a mixture of molten lithium and potassium carbonates. Left uses a convex hull to visualize a coordination of a selected Li^+ ion. Right uses a measuring tape to monitor distances between two selected ions. (see color plate on page 152)

Virtual reality techniques are used to study the structure and the dynamics of the simulated molten carbonates. Immersive environments are very suited to perceive valuable spatial information from the displayed data. For example, in what pattern the ions are diffusing, the dynamic nature of a coordination, in what way are carbonate ions oriented towards the metal ions, etc. In addition, immersive environments can provide added functionality to directly manipulate the data, such as direct manipulation of ions for selection and direct manipulation for measuring distances and angles between ions.

[2]Thanks to Jos Tissen (Unilever) and Jack van Wijk (TU Eindhoven) for providing the data. The 3D visualization techniques and their implementation are discussed in [15].

The bus configuration for the molecular dynamics visualization is shown in figure 5.5. It consists of six processes: one file, one render, two compute and two device processes. The file process reads in a sequence of data sets from disk. Every time step a data set is read, the file process will send an application specific event (`TIME_STEP`) to the bus. One device process manages head movements and posts a `HEAD_TRACK` event when applicable. The second device process manages the spatial joystick and posts a `JOYSTICK` event when the position, orientation or button state of the joystick has changed. One compute process computes the convex hull (triggered by the `TIME_STEP` and `JOYSTICK` events). The second compute process computes distances between atoms (triggered by the `TIME_STEP` and `JOYSTICK` events). Both compute processes post a `REDRAW` event. The render process redraws the scene (triggered by the `REDRAW` event). If appropriate, multiple redraw requests will automatically be collapsed into a single redraw.

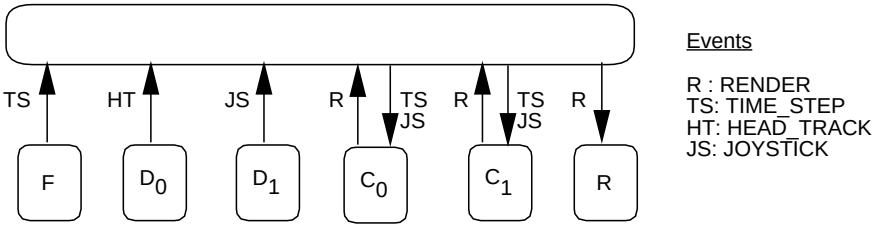


Figure 5.5: PVR Molecular Dynamics Visualization configuration. The flow of incoming and outgoing events is shown.

PVR programmers can improve application performance by interleaving (e.g., reading the next data set while rendering the previous one) and parallelism (executing multiple techniques simultaneously). When necessary, the programmer can change the process granularity by joining/splitting callback procedures. For the molecular dynamics application, the following typescript is a profiling synopsis that has been collected on a 2 CPU SGI Onyx2 system:

```
Total time:
      48.41
Total number of events:
       16171
Event latency:
       avg_latency= 0.01, max_latency= 0.05
Total time per thread:
      <D0+D1, 0.43><R, 47.6><C0+C1, 0.71><F, 6.25>
```

Given times are in seconds. PVR latencies are defined as the time interval between sending an event to the bus and receiving the event by a process.

5.4.2 Nuclear Division

One of the problems studied by researchers at the E.C. Slater Institute of the University of Amsterdam is that of nuclear division, i.e., *mitosis*. In the process of mitosis, two sets of microtubles interact to form the mitotic spindle. Of particular interest is the monitoring of the microtubles in the spindle over time.

Data sets have been acquired from a confocal light microscope and displayed using PVR[3]. Figure 5.6 shows two snapshots of the interface. The two spindle poles are clearly shown at 5 and 11 o'clock. The render process uses OpenGL Volumizer for the actual volume rendering. Simple colormap editing operations are available to allow interactive manipulation of the transparency values. Currently there is no interaction other than colormap manipulation and simple viewing transformations. The combination of stereoscopic graphics and head tracking allow the researcher to walk around the spindle, thus effectively perceiving the complex spatial relationships of the microtubules in the spindle.

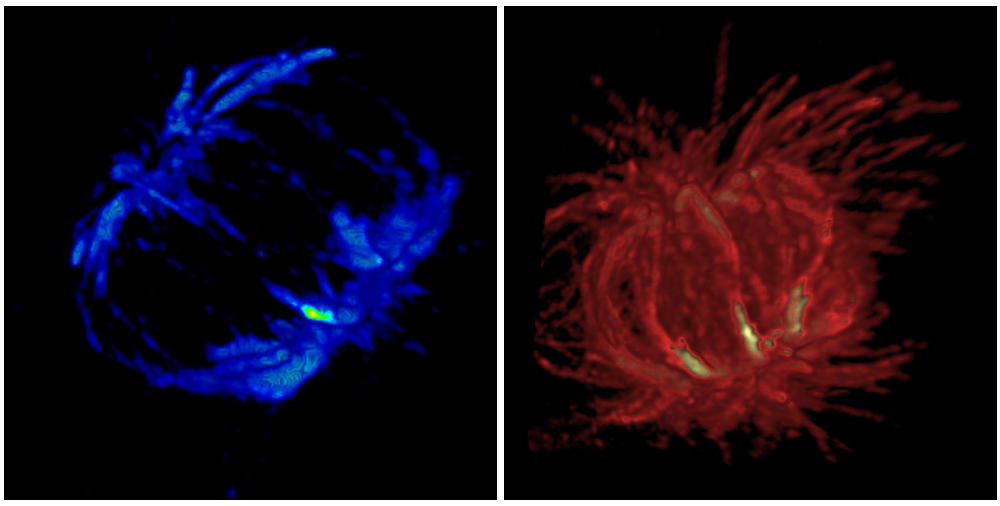


Figure 5.6: Microtubules in a spindle during nuclear division. Left uses a blue-yellow color map with high transparency values, right uses a red-yellow color map with lower transparency values. Head tracked stereo display allow researchers to effectively perceive the complex spatial relationships of the microtubules. (see color plate on page 152)

The bus configuration for the spindle is shown in figure 5.7. It consists of four processes: one file, one render and two device processes. The file process reads in the data sets from disk (currently, only one data set is read). After a frame is read, the file process will send an application specific `TIME_STEP` event to the bus. One device

[3]Thanks to Roel van Driel (ECSI, UvA) and Hans van der Voort (Scientific Volume Imaging B.V.) for providing the data.

process manages the head positions. This is exactly the same callback procedure as in the molecular dynamics application. The second device process edits a new colormap, and sends the application specific NEW_CMAP event to the bus. The render process redraws the scene (triggered by the REDRAW, or NEW_CMAP events). The NEW_CMAP event indicates that a new colormap must be used by the rendering process.

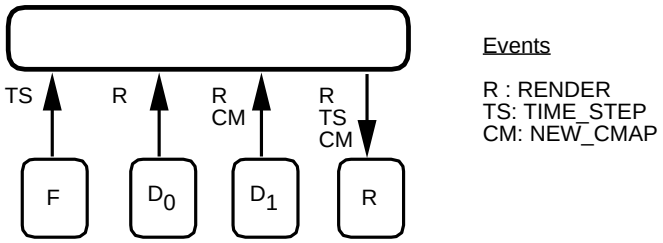


Figure 5.7: PVR Nuclear Division configuration.

5.5 Conclusions

In this chapter we introduced PVR, an architecture for portable VR applications. PVR facilitates the development of highly concurrent yet synchronized virtual reality interfaces by decoupling the coordination of the data streams from actual data processing. Two VR applications were used to illustrate PVR's governing ideas.

PVR differentiates itself from other VR toolkits in two ways. First, PVR is completely event driven. Programmers only supply callback routines which are executed whenever necessary. Application specific events can be posted to signal the completion of an application action. Second, PVR strives for portability by shielding low level device specific details from the programmer. A single code base can be used to span a variety of workspace and input device configurations.

In the future we will extend PVR with a number of new concepts. These will include the usage of the Fahrenheit scene graph for distributed VR applications, composition of higher level virtual input devices and other forms of high level interaction [16], incorporation of multiple 2D and 3D workspaces in a single application, and event tracing/monitoring tools.

References

[1] R. van Liere, J.A. Harkes, and W.C. de Leeuw. A distributed blackboard architecture for interactive data visualization. In R. Yagel and H. Hagen, editors, *Proceedings Visualization '98*, pages 235–244. IEEE Computer Society Press, 1998.

[2] S. Bryson. Virtual reality in scientific visualization. *Computers & Graphics*, 17(6):679–685, 1993.

[3] S. Bryson. Real-time exploratory scientific visualization and virtual reality. In L.J. Rosenblum et al, editor, *Scientific Visualization: Advances and Challenges*, pages 65–86. Academic press, 1994.

[4] R. Deline. Alice: A rapid prototyping system for three-dimensional interactive graphical environments. Technical report, Computer Science Department, University of Virginia, 1993.

[5] F. Hasenbink. Avocado system. Technical report, GMD Department of Visualization and Media Systems Design, Bonn St. Augustin, 1997.

[6] K. Watson and M. Zyda. Bamboo - a portable system for dynamically extensible, real time, networked, virtual environments. In *1998 IEEE Virtual Reality Annual International Symposium*, pages 252–260. IEEE Computer Society Press, March 1998.

[7] D. Rantzau and U. Lang. A scalable virtual environment for large scale scientific data analysis. *Future Generation Computer Systems*, 14(4):215–222, 1999.

[8] C. Carlsson and O. Hagsand. Dive - a platform for multi-user virtual environments. *Computers & Graphics*, 17(6):663–669, 1993.

[9] J. Landauer, R. Blach, M. Bues, A. Rosch, and A. Simon. Towards next generation virtual reality systems. In *Proceedings of the IEEE Conference on Multimedia Computing and System*, Ottawa, 1997. IEEE Computer Society Press.

[10] World Toolkit Development Team. World toolkit. Technical report, Sense8, 1997. http: // www.sense8.com/ products/ worldtoolkit.html.

[11] C. Shaw, J. Liang, M. Green, and Y. Sun. The decoupled simulation model for virtual reality systems. In *Proceedings of the CHI 92 Conference on Human Factors and Computing Systems*, pages 321–328, 1992.

[12] P. Appino, J. Lewis, L. Koved, D. Ling, D. Rabenhorst, and C. Codella. An architecture for virtual worlds. *Presence*, 1(1):1–17, 1992.

[13] C. Codella, R. Jalili, L. Koved, and J. Lewis. A toolkit for developing multi-user, distributed environments. In *Virtual Reality Annual International Symposium*, pages 401–407. IEEE Computer Society Press, 1993.

[14] D. Pape, C. Cruz-Neira, and M. Czernuszenko. CAVE users guide. Technical report, Electronic Visualization Laboratory, University of Illinois at Chicago, 1996. http: // www.evl.uic.edu /pape / CAVE / prog / CAVEGuide.html.

[15] J.J. van Wijk and J.T.W.M. Tissen. Visualization of molecular dynamics. In *Proceedings of the Fourth Eurographics Workshop on Visualization in Scientific Computing*, Abingdon, UK, 1993.

[16] J.D. Mulder. Remote object translation methods for immersive virtual environments. Stuttgart, Germany, June 1998. Presented at the 1998 Virtual Environments Conference & 4th Eurographics Workshop.

Chapter 6

PVR Implementation Issues

An overview of the PVR architecture was presented in Chapter 5. In this chapter a more in depth discussion on the technical issues related to the design and implementation is given.

6.1 Design

6.1.1 PVR device independence

Virtual environments have different display, tracking, and input devices. PVR attempts to shield application programmers from the management of the different physical characteristics of these devices. For instance, the programmers should not need to set up the appropriate projections, view ports, etc. Neither should programmers need to convert tracker and input device reports from the device's specific coordinate systems and formats into the coordinate system and format as used by the application. In PVR, device independence is realized by introducing an *Application Workspace* (AW), a *Standard Workspace* (SW), and *Device Workspaces* (DWs).

- Standard Workspace and Device Workspaces.

  The SW is defined by a right handed local coordinate system. The orientation of this coordinate system is defined as the x-axis pointing to the north and the z-axis to the east. The units used in this coordinate system are meters.

  Upon startup, a PVR application requires a configuration file that describes the mapping between this SW and the DWs; the workspaces used by each display, tracker, and input device. The application programmer uses the SW for the modeling and rendering of his application.

  Display devices are either statically positioned in the SW (such as a desk top monitor or the projection walls of a CAVE) or they are dynamically positioned (such as the display screens in a Head Mounted Display system). Statically positioned display devices are defined by their location in the SW, their orientation

in the SW, their display size in meters, and their display size in pixels. Dynamically positioned display devices are linked to an input DW (usually a tracker). Therefore, these display systems are defined by their location in the tracker's DW, their orientation in the tracker's DW, and their display size in meters and in pixels. In addition, it has to be defined to which tracking device the display is linked.

PVR distinguishes two categories of input devices: Spatial input devices (e.g., trackers, 6DOF-mouse) and non-spatial input devices (e.g., buttons, keyboards, joysticks). Spatial input devices typically provide position and orientation information in reference to a DW. The location and the orientation of this DW has to be mapped into the SW. In addition, different devices can provide that information in different formats and/or metrics. Therefore the data format and metrics used by the device has to be defined as well. Non-spatial input devices do not provide any spatial information and therefore are not related to a DW. So, no additional information is needed to map the device's information to the SW.

- Application Workspace and Standard Workspace.

  To incorporate existing applications into PVR, the Application Workspace is used which defines a mapping into the SW. This way, extensive rewriting of existing applications to be adapted to fit the SW, can be avoided.

  New applications can be developed in the SW. However, it is often more convenient to use an application specific workspace with an application specific position, orientation and scale. In addition, to port existing applications into PVR, it would be helpful if the existing application workspace could be used. Furthermore, in many applications navigation functionality is required. This can be accomplished by moving the application workspace around in the SW.

  The default AW is aligned with the SW. Three routines are provided to translate, rotate and scale the AW with respect to the SW. These operations can be performed in either AW or SW coordinates. For example, the AW can be rotated about one of its own principle axes, or about one of the SW's principle axes.

  Information from spatial input devices is provided in matrix form defining the trackers position and orientation in AW, SW, or DW coordinates. In addition, utility functions are provided to convert position and orientation vectors from one workspace to another.

  Rendering in a PVR application is performed with standard OpenGL. However, the programmer must not perform any window or projection commands. These are all handled by PVR internally.

6.1.2 PVR architecture

An overview of the PVR architecture was given previously in figure 5.1. The architecture consists of one or more processes attached to a bus. Processes communicate by a publish and subscribe mechanism. This mechanism is conceptually easy to understand: a process subscribes to patterns that describe the events it is interested in. When a process publishes an event the PVR bus broadcasts the events to all subscribing processes. Each process is parameterized with a callback which is invoked whenever the event occurs:

- Events, event filters and data frames.

  Applications can define events to denote any application specific action. PVR defines a small number of events for convenience, e.g., the `DRAW`, `MOTION`, `PRESS` events. These events are typically used by processes to invoke the rendering process or by processes to subscribe to input device state changes.

  Data associated with the event is also stored and carried in the event. For example, the `MOTION` event published by a spatial input device will contain information about the device identification, and the matrix that determines the device's position and orientation. The `PRESS` event will also contain information about the device identification and the button value. Specific application defined data can also be included in the event.

  Event filters are used by a process to specify only those events the process subscribes to. The PVR bus will send only those events included in the filter to the process. Filters are implemented by setting a filter mask.

  The simple event mechanism allows a publishing process to send a stream of events to a subscribing process. However, when multiple processes operate on one data set in parallel, additional mechanisms must be provided to order events. For example, when one process computes iso-surfaces and a second process computes streamlines in a time dependent data set. Both geometries must be rendered in the same frame. PVR provides a data framing mechanism to order data from parallel processes. Applications may define a data frame which consists of a number of data slots and an output event. Processes may put data in a slot. As soon as all slots are filled the bus will distribute the event. This signals that the frame is complete. The familiar notion of a barrier can be implemented with the framing mechanism. Processes may fill data into a slot at any time and in any order. Only when the last slot is filled will the bus invoke the callback of those processes that have subscribed to the event.

- Processes and PVR bus.

  The PVR bus will be configured as one or more render processes, device handling processes and zero or more compute processes. Compute processes are

responsible for all other required computation; for example the complete simulation itself, or the computation of visualization techniques (streamlines, isosurfaces, etc).

6.1.3 API Example

The following example gives a flavor of the decoupling of control and processing in PVR:

```
#include "pvr.h"
#include "pvr_util.h"

int renderCB (PVR_EVENT *ev, void *arg)
{
  handleRender ();
  return 1;
}

int inputCB (PVR_EVENT *ev, void *arg)
{
  int idx = *(int *)arg;

  switch (idx)
    {
    case 0:
      handleHeadSensor (ev);
      break;
    case 1:
      handleWandSensor (ev);
      break;
    case 2:
      handleButton (ev);
      break;
    }
  pvr_send_event (PVR_DRAW, NULL, PVR_NOWAIT);
  return 1;
}

void
initRender (void *arg)
{
  handleInitRender ();
  pvr_set_proc_cb (renderCB, NULL);
  pvr_set_event_filter (ALL_DEVICES, ALL_INDICES, PVR_DRAW);
}

void
initDevice (void *arg)
{
  int idx = *(int *)arg;

  switch (idx)
    {
    case 0:
      pvr_attach_device(PVR_SENSOR, PVR_UNIT1, PVR_MOTION);
      pvr_set_event_filter (PVR_SENSOR, PVR_UNIT1, PVR_MOTION);
      break;
    case 1:
      pvr_attach_device(PVR_SENSOR, PVR_UNIT2, PVR_MOTION);
      pvr_set_event_filter (PVR_SENSOR, PVR_UNIT2, PVR_MOTION);
      break;
```

```
    case 2:
      pvr_attach_device (PVR_BUTTON, PVR_UNIT1, PVR_PRESS|PVR_RELEASE);
      pvr_set_event_filter (PVR_BUTTON, PVR_UNIT1, PVR_PRESS|PVR_RELEASE);

    }
  pvr_set_proc_cb (inputCB, arg);
}

#define NDEV 3

int
main (int ac, char **av)
{
  int i;
  int a[NDEV];

  pvr_init (ac, av);
  pvr_attach_render_process (initRender, NULL);
  for (i = 0; i < NDEV; i++)
    {
      a[i] = 1;
      pvr_attach_process (initDevice, (void *) a[i]);
    }
  pvr_main_loop ();
  return 1;
}
```

In this example, four processes (a render process, two sensor processes, and a button process) are attached onto the PVR bus. The call to

```
pvr_attach_render_process (initRender, NULL)
```

attaches the render process. Upon initialization of this process, the callback `initRender(NULL)` is called. The calls to

```
pvr_attach_process (initDevice, (void *)&a[i])
```

attaches three other processes. Each process is initialized via the callback `initDevice(a[i]).`

`initRender` first initializes the application specific scene rendering (via `handleInitRender()`). The routine

```
pvr_set_proc_cb (renderCB, NULL)
```

sets the callback routine for this process to `renderCB`. The routine

```
pvr_set_event_filter (ALL_DEVICES, ALL_INDICES, DRAW)
```

specifies that this process subscribes to the event which can come from all devices categories and device indices.

The three calls to `initDevices` attach the appropriate input devices to each of the three processes. Each process sets the callback `inputCB`, and sets an event filter.

`inputCB` is called each time one of the sensors have changed or if the button device is pressed/released. After the device is handled the process will post a `DRAW`

event to the bus to indicate that the scene must be redraw. `pvr_post_event` can specify a `WAIT` or `NOWAIT` which indicates that the process will wait until the event is processed by the subscribed process (i.e., synchronous mode), or will not wait (i.e., asynchronous mode)

`renderCB` is called each time a `DRAW` event is received.

6.2 Implementation

6.2.1 Portability

The PVR architecture sketched in the previous section can be implemented in a number of ways. The current implementation uses a number of portable supporting software packages (see figure 6.1). The implementation relies on POSIX threads for process support, the X11 window system and OpenGL for windowing and rendering support, and the University of North Carolina's tracker library for a portable tracking interface. Applications may make use of higher level rendering packages can be used on top of OpenGL. For example, Inventor, OpenGL Optimizer or OpenGL Volumizer can be used within the rendering thread. However, packages like Inventor can only be used in configurations with one render thread.

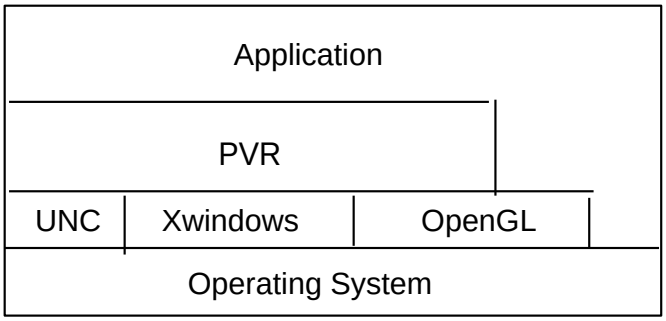


Figure 6.1: The layered implementation of the PVR architecture.

The current implementation supports an Electrohome Retrographics Display and the CAVE as output devices. A port to a multi-user Dextrous solution is underway. The Electrohome is driven by one render thread, while the CAVE is driven by four render threads.

6.2.2 Metrics

The PVR architecture was implemented in C. Table 6.1 gives an indication of the effort that is needed for the implementation. The table lists the PVR module, lines of code and a short description, for the kernel modules and three applications: a simple

Kernel Module	LoC	Description
BUS	3156	PVR bus management
WS	4545	PVR workspace management
SSD	2196	Physical to logical input device mappings
UTIL	803	Generic navigation and interaction techniques
Applications	**LoC**	**Description**
dino	245	PVR bus initialization and event handling
dino	280	drawing routines
inventor_viewer	296	PVR bus initialization and event handling
inventor_viewer	106	interaction routines
inventor_viewer	640	drawing routines
fekete	384	PVR bus initialization and event handling
fekete	496	snap dragging routines
fekete	114	interaction routines
fekete	1500	drawing routines
fekete	700	simulation code

Table 6.1: PVR code metrics

dinosaur object which the user can drag via generic navigation techniques, an inventor viewer which reads standard inventor files and uses generic navigation techniques to fly-through the inventor scenes, and the Fekete application discussed in Chapter 7.

The first part of table 6.1 shows the number of lines of code for the four modules. The BUS module implements the PVR bus administration, scheduling, and event handling. The WS module implements the workspace management. The SSD module implements the server serial devices. The UTIL module implements various navigation techniques.

The second part of the table shows the amount of code needed for the dino application, inventor viewer, and fekete application (discussed in Chapter 7).

A number of observations can be made from these tables:

- A substantial amount of code (approx. 4.5 KLoC) is needed to implement the application, standard and device workspaces. This amount of code would be needed if application programmers require a portable input/output device layer.

- Generic navigation techniques are primitive techniques that are used by most applications.

- The amount of code needed for drawing a scene is large.

6.3 Performance

In this section we discuss the performance of PVR when applied to the Fekete application. Performance measurements are performed on a 4 CPU SGI Onyx2 with Infinite reality graphics.

The PVR bus configuration is shown in figure 6.2. Each PVR process can execute on any of the four CPUs. The process `sim` implements the Fekete solver. After each iteration, `sim` will send a `REDRAW` event to the bus. The process `head_tracker` implements the sensor handlers. Each time a new sensor position or orientation is received, a `REDRAW` event will be sent to the bus. In this configuration there are two sensors; one that tracks the user's head and one that tracks the input wand. The process `joy_stick` implements the wand button handlers. Each time a wand button is used (via the push, motion, release sequence) a `REDRAW` event will be sent to the bus. The process `render` will redraw the Fekete point configuration upon each `REDRAW` event. The PVR bus will flush all `REDRAW` events in `render`'s event if multiple `REDRAW` events are pending in that queue; i.e. multiple `REDRAW` events will be collapsed to one event.

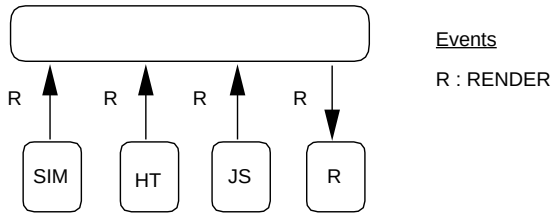


Figure 6.2: PVR bus configuration of the Fekete application. The processes `sim`, `head_tracker`, `joy_stick` and `render` are labeled SIM, HT, JS, and R.

The Fekete point configuration is stored in shared memory. Process `sim` will lock the memory upon writing a new configuration. Process `render` will lock the memory upon reading a configuration. Locking of global data structures may cause additional rendering latencies.

Figures 6.3 shows four performance plots. The upper left plot shows the total number of `REDRAW` events sent to the PVR bus. Performance is measured in number of events per second (vertical axis) as a function of time in seconds (horizontal axis). The Fekete application was run for 144 seconds. The upper right, lower left and lower right plots shows the number of `REDRAW` events sent by the `head_tracker`, the `joy_stick` and the `sim` processes.

As discussed in Chapter 5, the simulation is idle when the user drags a point to a new position. This is reflected in the lower two plots; the process `sim` is idle when the `joy_stick` process is active. The head tracking and joystick sensors are polled at 30 hertz.

Figures 6.4 shows two performance plots related to rendering speeds and latencies.

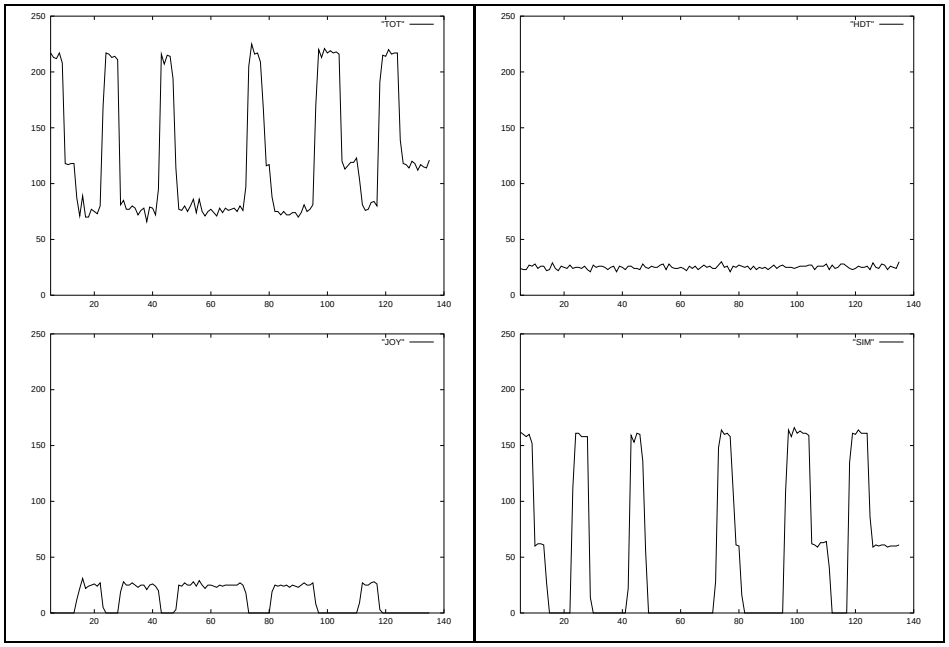


Figure 6.3: Number of REDRAW events sent by PVR processes; all REDRAW events (upper left), events sent by head tracker (upper right), events sent joy stick events (lower left) and events sent by simulation (lower right).

The left plot shows the frame rate of the render process. Performance is measured in frames per second (vertical axis) as a function of time (horizontal axis). The right plot concerns latency of the render process for REDRAW events. Latency is defined as the time interval between sending a REDRAW event to the bus and receiving the event by the render process. The average latency is computed as the sum of all latencies divided by the number of REDRAW events. The right plot shows the average latency per second (vertical axis) as a function of time (horizontal axis).

A number of general performance observations related to the PVR architecture can be given:

- The average frame rate in table 6.4 is 58, which is approximately 29 stereo frames per second.

- The frame rate fluctuations of the render process are minimized, due to the decoupling of render process with other processes. The fluctuations shown in figure 6.4 are due to lack of redraw events sent to the bus during user interaction with the joystick. More events can be generated if the joystick would be polled more frequently.

- Device latencies are minimized by allocating a separate PVR process for every device. Each device process can continuously poll the device and send an event a

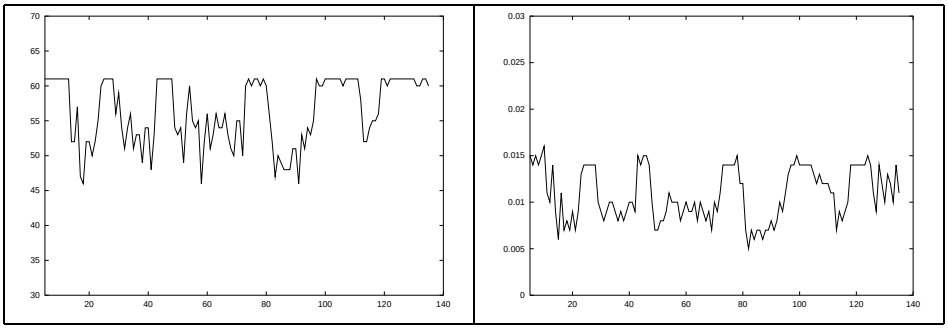


Figure 6.4: Frame rate of rendering process (left) and average rendering latencies (right).

bus as soon as a state change occurs. In particular, the head movement latencies are minimized, due to the separation of the head sensor handling process.

The average latency of redraw events in figure 6.4 is 0.01 seconds.

- Time critical applications can be implemented within the PVR architecture through a scheduler which manages a time budget for each process. An example of a time critical PVR application can be found [1].

6.4 Retrospective Assessment

The three aspects that motivated the design of the architecture were: (i) simplification of multiple I/O data streams management (ii) 3D interaction with the virtual world, and (iii) the large variety of display and input devices with each device having individual characteristics and constraints. Each aspect is assessed:

- Management of multiple I/O data streams.

  PVR is a multi-threaded architecture that allows the rendering process to be decoupled from the computation processes. This allows applications to achieve the real-time requirements of VR user interfaces, since computations will not interfere with rendering.

  Events are used for two purposes: to signal state changes in devices and to carry the state of a device. Hence, the behavior of PVR devices are modeled as a stream of events. The PVR bus is responsible for the coordination and management of multiple event streams. Decoupling the coordination from the data processing of an event stream simplifies the programmer's task of managing and synchronizing the data streams. PVR programmers need only to provide a callback routine which the PVR bus will invoke when the event occurs.

- 3D interaction.

Many navigation techniques can be defined as transformations with respect to the standard workspace. For example, a fly-through can be programmed as a set of translations and rotations of the eye position of the application workspace with respect to the standard workspace.

Reuse and extension is stimulated by providing a set of basic 3D techniques and a modular mechanism for integrating these into an application. Reuse of basic 3D interaction and navigation techniques allows for rapid development of applications. In addition, the modular mechanism of the PVR architecture allows the basic 3D techniques to be combined and extended into higher level interaction techniques.

- Variety of output and input devices.

  Device independence is supported by taking advantage of the mappings between application, standard and device workspaces. This allows programmers to develop and maintain a single code base spanning a variety of input and output device configurations.

- Usability.

  PVR has been used for the development of other VR applications and VR related techniques. Other applications can be found in [2, 3, 4]. Additional techniques used to extend PVR can be found in [1, 5, 6].

References

[1] Henk Huitema and Robert van Liere. Time critical computing and rendering of molecular surfaces using a zonal map. In J.D. Mulder and R. van Liere, editors, *Proceedings EG Workshop on Virtual Environments*, pages 115–124. Springer Verlag, 2000.

[2] H. Huitema and R. van Liere. Interactive visualization of protein dynamics. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings IEEE Visualization 2000*, pages 465–468. IEEE Computer Society Press, 2000.

[3] W. de Leeuw, R. van Liere, P. Verschure, A. Visser, E. Manders, and R. van Driel. Visualization of time dependent confocal microscopy data. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings IEEE Visualization 2000*, pages 473–476. IEEE Computer Society Press, 2000.

[4] Robert van Liere and Wim de Leeuw. Exploration of large image collections using virtual reality devices. In D.S. Ebert and C.D. Shaw, editors, *Proceedings on New Paradigms for Information Visualization (NPIVM'99)*, pages 83–86. ACM Press, 2000.

[5] Jurriaan Mulder and Robert van Liere. Fast perception-based depth of field rendering. In K.Y. Wohn, editor, *Proceedings ACM Symposium in Virtual Reality Software and Technology(VRST 2000)*, pages 129–133. ACM Press, 2000.

[6] Jurriaan Mulder and Robert van Liere. Enhancing fishtank virtual reality. In Steven Feiner and Daniel Thalmann, editors, *Proceedings of IEEE VR 2000*, pages 91–98. IEEE Computer Society Press, 2000.

Chapter 7

Virtual Fekete Point Configurations [1]

Virtual environments have shown great promise as a research tool in science and engineering. In this chapter we study a classical problem in mathematics: that of approximating globally optimal Fekete point configurations. We found that a highly interactive virtual environment, combined with a time-critical computation, can provide valuable insight into the symmetry and stability of Fekete point configurations. We believe that virtual environments provide more natural interfaces to complex systems, allowing users to perceive, interpret and interact with the problem more rapidly.

7.1 Introduction

An ever increasing demand exists for detailed investigation of complex systems. To simulate real-life phenomena accurately, models are established which can be hard to solve due to their size and complexity. The solutions to these models often involve large data sets, from which it is difficult to extract not only whether the model describes the physical reality accurately, but also whether the model was solved correctly. In order to test the sensitivity of the solution to changes in parameter settings, the amount of data increases even more, because multiple data sets have to be compared. Often it occurs that only a small subset of the data set contains the interesting features of a simulation, but it is difficult to determine this subset beforehand.

Virtual environments have shown great promise for the visualization of these data sets in which exploration plays an important role. Explorative environments allow users, motivated by insufficient knowledge of what is contained in the data, to interactively develop insight into their problem. Due to the interactive nature of these environments, they play a predominant role for gaining insight into complex systems, in which a small perturbation of a parameter setting can result in very different output solutions.

[1] This chapter has been published as: R. van Liere, J.D. Mulder, J.E. Frank and J.J.B. de Swart, Virtual Fekete Point Configurations: a Case Study in Perturbing Complex Systems. In Steven Feiner and Daniel Thalmann, *IEEE Virtual Reality 2000*, pages 189–196, 2000.

In this chapter we study a classical problem in mathematics: that of approximating globally optimal Fekete point configurations. This problem contains many ingredients of a complex system. In the next section we provide some related work. Section 7.3 provides a mathematical formulation of the Fekete problem, and briefly discusses a very efficient solution method. In Section 7.4 we describe the visualization and interaction techniques used to build the virtual environment. Moreover, we discuss some time-critical performance issues that have been used to meet our performance requirements. Finally, in Section 7.5, we discuss the merits of using virtual environments for analyzing the Fekete problem.

7.2 Related Work

The analysis and determination of elliptic Fekete point sets has attracted the attention of theoretical and numerical mathematicians and researchers in scientific modeling. The problem was originally proposed by Fekete [1], and has been studied for almost 75 years. It also represents a long-standing numerical challenge: Pardalos states it as one of the open problems in global optimization [2]. There are many scientific applications that can be modeled as the solution of the Fekete problem (and its possible modifications). Erber and Hockney have studied a problem related to the Fekete problem to find equilibrium configurations of equal charges on a sphere [3]. Practical applications include problems in structural chemistry, the design of multi-beam laser implosion drives, the optimum placement of communication satellites, and packing and covering problems.

Recently, virtual environments have been used to study mathematical problems and objects. For example, Francis et al. have used distributed CAVEs to study the eversion of a sphere in a collaborative setting [4]. Roseman has used the CAVE to display surfaces in 4-dimensional spaces [5]. Both systems exploit the additional spatial dimension inherent in a virtual environment to get extra information about the structure of the displayed mathematical objects. The focus of our work differs in that we exploit VR interaction styles to gain insight into behavior of complex systems.

Bryson has extensively studied the application of virtual reality interfaces in scientific visualization. Experiments have been performed to study time management, time-critical computing and time-critical algorithms [6, 7].

7.3 The Fekete Problem

7.3.1 Formulation

The problem can be formulated as follows: given the unit sphere B in the Euclidean real space $\mathbb{R}^3$, and a positive integer n, find the n-tuple of points (unit length vectors)

$$q(n) = \{q_i,\ i = 1, \ldots, n\}\ , \qquad q_i = (q_{i1}, q_{i2}, q_{i3})^T$$

on the surface S^2 of B, which maximizes the product of distances between all possible pairs $\{q_i, q_j\}$, $1 \leq i < j \leq n$. In other words, we are interested in finding the global maximum of the function

$$f_n(q(n)) = \prod_{1 \leq i < j \leq n} \|q_i - q_j\| \,, \qquad q_i \in S^2$$

where $\|\cdot\|$ indicates the Euclidean norm. A set of vectors $q^*(n) = \{q_i^*,\ i = 1, \ldots, n\}$, where $q_i^* \in S^2$, which satisfies the relations

$$f_n^* = f_n(q^*(n)) = \max_{q(n)} f_n(q(n)) \,, \tag{7.1}$$

is called a set of elliptic Fekete points of order n [1]. We refer to equation (7.1) as the Fekete (global optimization) problem.

According to the classical theorem of Weierstrass, the Fekete optimization problem has one or more globally optimal solutions. The traditional approach to computing $q^*(n)$ is with a constrained optimization package. For large n, this can be a very expensive computation, see the comparison in [8].

7.3.2 Configuration Symmetries

Due to symmetry and rotations, there are infinitely many solution vector sets $q^*(n)$ which satisfy equation (7.1). In order to compare geometric properties of stable point configurations, it is useful to rotate configurations into a rotation-free canonical form.

For notational purposes we represent arbitrary point configurations in spherical coordinates. The n-tuple $q(n)$ –consisting of corresponding unit vectors q_i, $i = 1, \ldots, n$– is denoted by

$$\begin{aligned} q_{i1} &= \cos(\theta_i)\sin(\phi_i) \\ q_{i2} &= \sin(\theta_i)\sin(\phi_i) \\ q_{i3} &= \cos(\phi_i) \end{aligned}$$

in which $0 \leq \theta_i < 2\pi$ and $0 \leq \phi_i \leq \pi$.

For each q_i, $i = 1, \ldots, n$, define the associated partial energy

$$e_i = \ln \prod_{j \neq i} \|q_i - q_j\|. \tag{7.2}$$

We may use this partial energy to give the points a canonical ordering with respect to certain rotations (see [3]). Suppose that the n points are ordered such that their partial energies form a non-decreasing sequence, that is,

$$e_1 \leq e_2 \leq \ldots \leq e_n.$$

Choose a point from the set with the lowest energy and rotate the configuration so that this point is placed at the north pole $\theta = \phi = 0$. Next choose the point from the set

of points with the second lowest partial energy which is closest to the north pole and rotate the entire configuration so that this second charge is at zero longitude, $\theta = 0$.

Two point configurations are deemed equivalent if and only if the positions of the points in the rotation-free canonical form are equivalent.

7.3.3 Alternative Formulation

Shub and Smale [9] refer to numerical difficulty of finding the globally optimal configuration $q^*(n)$, for a given—not too small—n. Difficulties arise due to several reasons: the above mentioned various symmetries of the function f_n, and, more essentially, its inherent multi extremality. The number of local extrema increases drastically with n [3]. Furthermore, the difference in extremal values of $f^*(n)$ is very small, and the energy landscape is shallow near extrema, making any perceivable numerical solution procedure inherently tedious.

An alternative formulation of the problem has been presented in [8]. The n-tuple $q(n)$ is represented as a set of n repellent particles, which move on the unit sphere under influence of an adhesion force. To describe the dynamic behavior of the particles, the coordinates of the particles are parameterized by a time variable t. Introducing the velocity vectors p_i, the resulting system can be described by the following set of differential-algebraic equations:

$$\begin{aligned} \dot{q}_i &= p_i \\ \dot{p}_i &= -\kappa p_i - \nabla_{q_i} V(q) - 2q_i\lambda_i \\ 0 &= q_i \cdot q_i - 1 \end{aligned} \tag{7.3}$$

Here, the notation $\dot{q} \equiv dq/dt$ is used to denote differentiation with respect to time, $-\kappa p$ represents the adhesion force, $2q_i\lambda_i$ represents the normal force associated with the constraint $\|q_i(t)\| = 1$, and the potential energy of the repellent force field $V(q)$ is defined by

$$V(q) = -\ln \prod_{1\le i<j\le n} \|q_i - q_j\|.$$

Due to the adhesion force, the solution of the dynamical system (7.3) will approach a steady state such that $\nabla V(q^*) = 0$. If, in addition, $V''(q^*)$ is positive definite, then q^* will be a local minimum of $V(q)$. Clearly, minimization of $V(q)$ is equivalent to maximization of $f_n(q(n))$.

7.3.4 Solver

A very efficient algorithm is proposed that exploits the special structure and retains certain physical properties of the dynamical system. To give a flavor of the numerics involved, we briefly discuss this solution method.

The dynamical system (7.3) can be seen as a constant energy (Hamiltonian) system to which damping has been added. The constrained Hamiltonian dynamics are described by

$$\begin{aligned} \dot{q}_i &= p_i \\ \dot{p}_i &= -\nabla_{q_i} V(q) - 2q_i\lambda_i \\ 0 &= q_i \cdot q_i - 1 \end{aligned} \tag{7.4}$$

where λ_i is a Lagrange multiplier. Research conducted over the last ten years has shown the importance of preserving the symplectic structure to effectively simulate such a system [10]. The remaining part of (7.3) is a system of pure damping:

$$\begin{aligned} \dot{q} &= 0 \\ \dot{p} &= -\kappa p \end{aligned} \tag{7.5}$$

with exact solution $q(t) = q(0), p(t) = \mathrm{e}^{-\kappa t}p(0)$. Our approach is to use a symmetric splitting method to propagate the solution to (7.3), with three steps:

1. Solve (7.5) exactly over an interval of length $h/2$.
2. Solve (7.4) approximately over an interval of length h with the symplectic leapfrog method described below.
3. Solve (7.5) exactly over an interval of length $h/2$.

Here it is assumed that the output of each step serves as the input to the succeeding step. Repeated application of the algorithm produces a discrete representation of the solution which is second order accurate in the time step size h.

The leapfrog method for the constrained system (7.4) is defined by first solving simultaneously (for the unknowns q_i^{n+1}, $p_i^{n+1/2}$ and λ_i):

$$\begin{aligned} q_i^{n+1} &= q_i^n + hp_i^{n+1/2} \\ p_i^{n+1/2} &= p_i^n - \tfrac{h}{2}\nabla_{q_i} V(q^n) - 2q_i^n\lambda_i \\ 0 &= q_i^{n+1} \cdot q_i^{n+1} - 1 \end{aligned}$$

for $i = 1, \ldots n$, and then solving simultaneously (for the unknowns $p_i^{n+1/2}$ and μ_i):

$$\begin{aligned} p_i^{n+1} &= p_i^{n+1/2} - \tfrac{h}{2}\nabla_{q_i} V(q^{n+1}) - 2q_i^n\mu_i \\ 0 &= q_i^{n+1} \cdot p_i^{n+1} \end{aligned}$$

for $i = 1, \ldots n$, where the Lagrange multipliers λ_i and μ_i are chosen (in the case of λ_i by solving a scalar quadratic equation) to satisfy the algebraic constraints.

This method is shown to exactly preserve the evolution of the symplectic structure, suggesting that we obtain the "proper" equilibrium configuration associated with a given initial condition.

If the trajectories of two points should make a close approach (or if during interaction two points should be placed nearby each other), the resulting accelerations may

cause the algorithm to break down unless the step-size is adjusted accordingly. To do this while maintaining the symmetric structure of the overall integration scheme requires care. For this purpose we have used an adaptive version of leapfrog based on introducing a transformation of time [11].

7.4 Virtual Fekete Point Configurations

We have implemented the Fekete problem in a virtual environment. The implementation consists of two tightly coupled components: the user interface and the computation. Interaction with the computation is performed by changing parameters or the state of a configuration. These changes will immediately be taken into account by the computation, i.e., computational steering is supported.

The goal of the interface is to provide a highly interactive environment which will allow the user to study the dynamics of Fekete point configurations. Since there are multiple rotation-free stable configurations, the user must have the ability to study the sensitivity to perturbation of stable configurations. In this section we present various interactive visualization techniques used to implement the interface. In Section 7.5, we discuss the merits of using virtual environments for analyzing the dynamics of Fekete point configurations.

7.4.1 Presentation

Figure 7.1 shows the basic user interface of the virtual environment. A point configuration is presented as a set of small spheres, a convex hull and trajectories from each sphere. The positions of the spheres indicate the Fekete points. The convex hull is used to visualize connectivity relationships between points. In addition, the convex hull is used as an important depth cue to determine the position of a point. Trajectories show the path the points have taken during a finite number of time steps. The mean velocity of a point can be estimated by examining the length of its trajectory.

The user interface also supports the concept of a *local moonscape*, see figure 7.2. A local moonscape computes and displays the energy of the configuration in the neighborhood of a point. At a stable configuration the energy values on the local moonscape will always be greater or equal to the energy value of the point. However, when a configuration has not yet reached a stable state, the energies in the local moonscape will differ. The point will tend to move to those areas on the moonscape that have lower energies.

The moonscape is drawn as a shaded mesh. Height and color are used to indicate when the energy is higher (green) or lower (red) than the energy in the point.

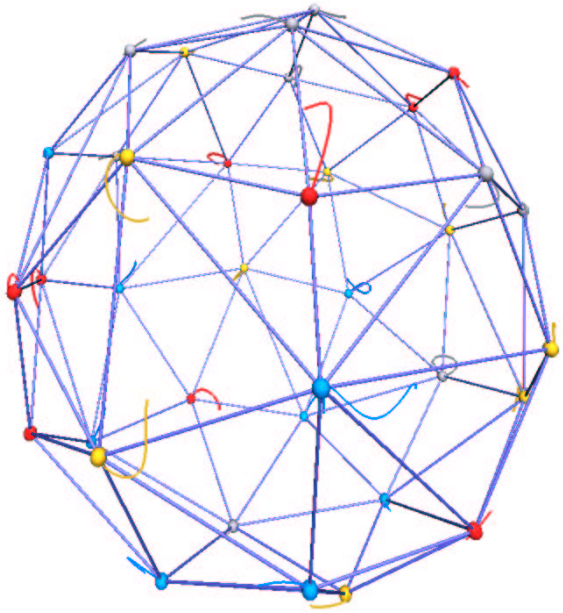


Figure 7.1: A 36 point configuration with convex hull and trajectories. (see color plate on page 153)

7.4.2 Interaction

Users may perturb a configuration by dragging a selected point to a new position. During this interaction the configuration will not be updated. Only when the interaction is finished (defined by releasing the selected point), will the computation immediately continue with the perturbed configuration. In this way the user can gain insight into the stability of a configuration, i.e., the perturbed configuration may converge to the previous configuration or it may converge to a different configuration.

In addition to the simple method of perturbing a configuration, we have added a technique that we call *snap convergence*. This technique computes and displays the converged configuration *while* the user is perturbing a configuration. Figure 7.3 illustrates snap convergence. A selected point (indicated by the yellow sphere and a red line as a pointer from the wand) in the current configuration is being dragged to a new location. The upper left hand corner of the image shows a list of stable configurations that have been encountered. One configuration in the list is highlighted (in yellow) to indicate the configuration which the current configuration will converge to if the selected point is to be released. When the user drags the point to a new location a different configuration may be highlighted.

Also, a history of red or yellow crosses are drawn on the path taken by the dragged point. The color of the cross indicates the configuration which will be converged to that location. In this way, a user can interactively construct colored regions which

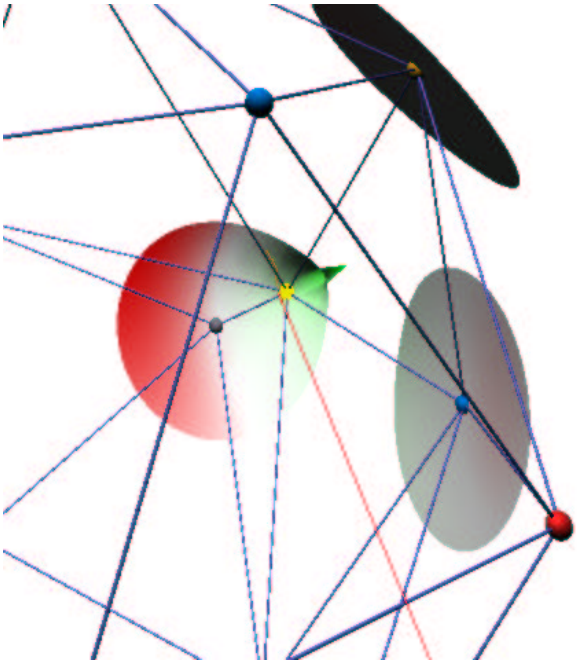


Figure 7.2: Local moonscape around a point. The moonscape visualizes the stability around the point. (see color plate on page 153)

denote the energy surface.

Snap convergence can be used to gain insight into how much a configuration must be perturbed before it will converge to a different configuration. Snap convergence annotated with colored crosses can be used to provide an indication of the global energy landscape.

7.4.3 Performance Issues

The Fekete environment is implemented on top of PVR, a modular library for developing portable VR applications [12]. PVR provides support for multiple processing, allowing the rendering module(s) to be decoupled from the computational and device management modules.

The goal was to implement the Fekete virtual environment which would achieve an update rate of minimally 10 frames/sec. Time critical computational algorithms have been essential for obtaining this responsiveness. We describe some computational issues that are performance critical, and discuss tradeoffs that have been made to realize the performance goals:

- Computation.

  Solving the system (7.3) by standard differential-algebraic equations software works well in that these packages find larger values for $f^*(n)$ in less computing

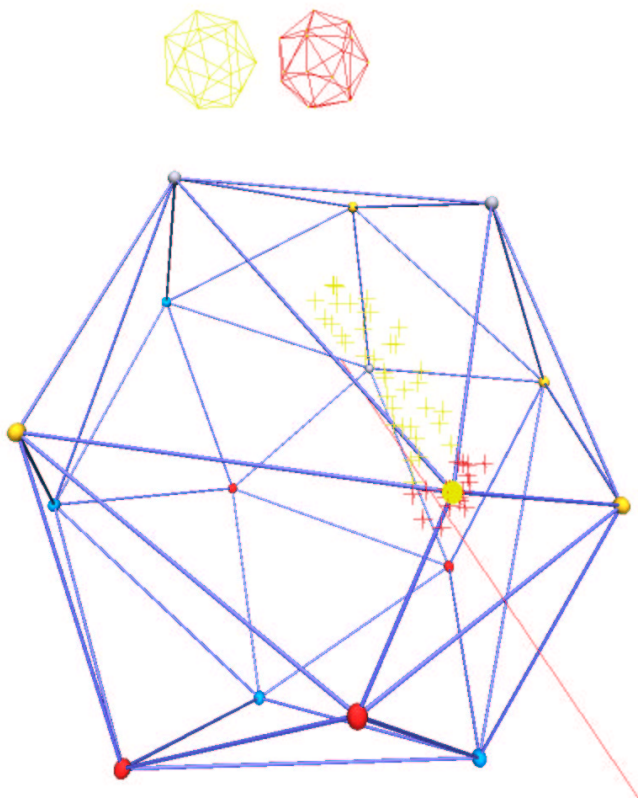


Figure 7.3: Snap convergence during perturbation of a configuration. The small red configurations on the upper left are a list of converged configurations. The yellow configuration is the configuration to which the current configuration will converge. Small yellow and red crosses are drawn on the path taken by the dragged point to indicate the stable configuration at the location of the cross. (see color plate on page 154)

time than off-the-shelf optimization packages. The algorithm given in Section 7.3.4, however, performs better than standard differential-algebraic equations solvers by orders of magnitude. For example, for a random initial configuration and a relatively small number of points, the algorithm can find a stable configuration in a few seconds.

In addition, the algorithm is, as far as we know, the most robust way to solve the Fekete problem. We rely on its robustness property when perturbing a configuration.

- Local moonscapes.

  Local moonscapes define the energy values in a neighborhood of a point. A moonscape is defined as a mesh, parameterized with its radius, and the number of rings R it contains. Each ring r adds 2^{r+1} points to the mesh. Each subsequent ring r is connected to ring $r-1$ by a triangle strip of $2^r \times 3$ triangles. The first

ring defines a triangle fan of four triangles with the particle as the center point. Hence, $2^{R+2} - 4$ additional energy computations are required to compute the moonscape.

Increasing the number of rings will result in a more accurate representation of the energy in the neighborhood of a point. The trade-off is performance versus accuracy. We have chosen to adaptively decrease the accuracy when a configuration is not yet in a stable state, i.e., when the animation is (quickly) changing. When the configuration converges, the accuracy of the moonscape will increase.

- Snap convergence.

  A configuration is perturbed by dragging a point to a new location. Snap convergence does an on-the-fly convergence computation of a configuration while it is being perturbed. For each new location a new converged configuration will need to be computed. Snap convergence is realized by running the computation in time-critical mode. It is important that the solver can compute a converged configuration a few times per second.

  In order to achieve this, the solver has been adapted in three ways. First, the time step parameter has been relaxed. This allows larger time steps to be taken, resulting in faster computations. However, in some cases, larger time steps may compromise the accuracy and robustness of the computation. Second, the tolerance used in comparing rotation free canonical forms is relaxed, so that similar configurations can be found faster. Third, the number of iterations needed to converge to a solution is bounded to an upper limit. This allows the computation to always return in a maximal number of iterations.

  The techniques used to speed up snap convergence compromise the computation, in that the correct converged configuration may not be found. This is a trade-off between computational accuracy and interactive performance. It is also possible that a converged configuration may not be found within the time constraints posed by snap convergence. In this case, no configuration will be highlighted.

7.5 Discussion

The virtual environment has been used to study the dynamics of the alternative formulation of the Fekete problem. In Table 7.1 we list some important notions that provide insight into the mathematics of the model (left) and the corresponding interactive visualization techniques used to display these notions. For example, the convex hull is a very useful technique to gain insight into regularities and symmetries of a configuration. Also, combining a moonscape with dragging and animating the solution of a configuration gives insight into the stability and sensitivity to perturbation of a configuration.

insight to problem	visualization techniques
stability	moonscape
regularities/symmetry	convex hull
velocity of points	trajectories, animation
interactions between points	hull, moonscape, animation
surface count	hull
convergence paths	trajectory
sensitivity to perturbation	dragging + animation
global energy landscape	snap convergence

Table 7.1: Notions that provide insight to the Fekete problem (left) and their corresponding interactive visualization techniques (right). Motion parallax, spatial input and interactive performance are essential ingredients to provide a natural interface for the visualization techniques.

7.5.1 Why virtual environments?

The remaining question is: what is the added value of a virtual environment for studying the Fekete problem? First, we list some concrete benefits we found:

- Motion parallax (obtained via head coupling with the display) has proved to be an essential visual depth cue to interpreting the convex hull. Users can search for regularities in configurations by simply moving the head to a different location. It is doubtful that this effect can be achieved via non-head coupled displays by using 3D graphics rotation techniques, such as “rocking” or variations of “grand tour”.

- Spatial input is used when perturbing a configuration. Dragging a point with spatial input devices has proven to be more intuitive, efficient, and accurate than with two dimensional input devices.

- Other advantages of using a virtual environment are the additional depth cues when interpreting trajectories and small local disturbances on a moonscape.

Also, a number of more general features can be identified:

- Due to the more “natural” environment, the user can perceive and interpret visualization cues more rapidly. For example, by taking advantage of motion parallax it was possible to observe the slow convergence of a number of points to their equilibrium locations in a plane, by moving the observation point to a location on the plane. This was done quite subconsciously by the observer, and is something which would probably never have been noticed using more conventional visualization techniques.

- Due to the intuitive representation of the dynamics of configurations, more information can be represented in the interface. For example, the history of a configuration (i.e., trajectories), the current configuration (i.e., hull) and the neighborhood of a point (i.e., moonscapes) can be shown in one image. It is difficult to provide his amount of information on a 2D display due to, among others, hidden lines and surfaces.

- The fast and intuitive interaction invites the user to experiment and explore the dynamics of a configuration.

7.5.2 Evaluation

The implemented virtual environment is being used extensively. Users find that the environment provides a more natural interface, allowing them to perceive, interpret and interact with point configurations more rapidly. The explorative nature environment of the environment allows users to see relationships and test hypotheses of configurations in detail. In addition, various pathological configurations that converge very slowly could be analyzed.

The environment is used in two ways. First, as discussed above, it allowed users to gain insight into the complex dynamics of the Fekete problem. Second, the environment was used for the development and validation of the model and the solver themselves. Developers were able to test and debug many solving strategies, in particular, how to deal with variable stepsizes during integration.

Nevertheless, there are also a number of drawbacks of this environment. First, the current interface contains only minimal quantitative information. Quantitative values such as point positions and velocities, plots of energy history, etc are absent. Providing this information in a fixed area in the workspace would not be an adequate solution, since the user would need to switch viewing directions to examine this information. Second, the development of the interface was not done by the same people that developed the numerical codes. Programming VR interfaces is still very cumbersome for experienced programmers that do not have the required skills.

7.6 Conclusions

In this chapter we discussed a virtual environment that allows the user to analyze the dynamics of Fekete point configurations. We found that a highly interactive environment, combined with a time-critical computation, provides valuable insight into the symmetry, stability and sensitivity of the point configurations. We believe that the virtual environment provides a more natural interface to complex problems, allowing users to perceive, interpret and interact with the problem more rapidly.

Acknowledgements

Jason Frank provided the solver discussed in Section 7.3.4. The solver was implemented as a batch process in C. A computed equilibrium was analyzed by printing the initial conditions, the number of steps required to produce the equilibrium, and the positions of the points at the equilibrium. In order to allow for snap convergence (see Section 7.4.2) and to integrate the solver in the PVR, I have adapted the solver in a number of ways.

Jurriaan Mulder developed the moonscape technique discussed in Section 7.4.1.

References

[1] M. Fekete. Über die Verteilung der Wurzeln bei gewisser algebraischen Gleichungen mit ganzzahligen Koeffizienten. *Mathematische Zeitschrift*, 17, 1923.

[2] P.M. Pardalos. An open global optimization problem on the unit sphere. *Journal of Global Optimization*, 6:213, 1995.

[3] T. Erber and G.M. Hockney. Complex systems: Equilibrium configurations of n equal charges on a sphere. *Advances in Chemical Physics*, XCVII:495–594, 1997. ISBN 0-471-16285-X.

[4] G. Francis, J.M Sullivan, R.B. Kusner, K.A. Brakke, C. Hartman, and G. Chappell. The minimax sphere eversion. In H.C. Hege and K. Polthier, editors, *Visualization and Mathematics*, pages 3–20. Springer, 1997.

[5] D. Roseman. What should a surface in 4-space look like? In H.C. Hege and K. Polthier, editors, *Visualization and Mathematics*, pages 67–82. Springer, 1997.

[6] S.T. Bryson and S. Johan. Time management, simultaneity and time-critical computation in interactive unsteady visualization environments. In R. Yagel and G.M. Nielson, editors, *Proceedings IEEE Visualization '96*, pages 255–262. IEEE Computer Society Press, 1996.

[7] S. Bryson. Time-critical computational algorithms for particle advection in flow visualization. In *Visualization '99 (Proceedings of the IEEE 1999 Visualization Conference, Late Breaking Hot Topics)*, 1999.

[8] W.J.H. Stortelder, J.J.B. de Swart, and J.D. Pintér. Finding elliptic Fekete point sets: Two numerical solution approaches. *CWI Quarterly*, 12(1):63–76, 1999.

[9] M. Shub and S. Smale. Complexity of Bezout's theorem III. Condition number and packing. *Journal of Complexity*, 9:4–14, 1993.

[10] J.M. Sanz-Serna and M.P. Calvo. *Numerical Hamiltonian Problems*. Chapman & Hall, 1994.

[11] W. Huang and B. Leimkuhler. The adaptive Verlet method. *SIAM Journal on Scientific Computing*, 18:239–256, 1997.

[12] R. van Liere and J.D. Mulder. PVR - an architecture for portable vr applications. In M. Gervautz, A. Hildebrand, and D. Schmalstieg, editors, *Virtual Environments '99, Proceedings of the Virtual Environments Conference & 5th Eurographics Workshop*, pages 125–135. Springer Verlag, 1999.

Part III

Information Visualization

Chapter 8

GraphSplatting: Visualizing Graphs as Continuous Fields [1]

In this chapter we introduce GraphSplatting, a technique which transforms a graph into a two-dimensional scalar field. The graph is not displayed as a graphical representation of vertices and edges, but as a continuous field which preserves the main structure of the graph.

There are three advantages to using GraphSplatting to represent a graph. First, arbitrarily large graphs can be visualized without cluttering. Second, application dependent information contained in the graph can be used to control the transformation. Finally, various data visualization and interaction techniques can be used to analyze the data.

We illustrate GraphSplatting in several applications.

8.1 Introduction

The main problem of information visualization is the mapping of complex, non-spatial, abstract data onto effective visual forms. Relational structures, consisting of a set of entities and relationships between those entities is an important class of such data. Such structures are commonly modeled as graphs: the entities are vertices, and the relationships are the edges. A large number of graph drawing algorithms have been developed that present the information in a graph effectively. Effective graph drawings can help the user to understand the underlying abstract data.

Graph drawing algorithms consist of two steps: layout and rendering. Graph layout is concerned with the placement of vertices and edges, such that properties in the graph are spatially conveyed. Graph rendering is concerned with mapping vertex and edge positions onto graphical objects and rendering these objects on the display. Usually graph drawing algorithms use discrete graphical objects to represent vertices and

[1] This chapter has been submitted for publication: R. van Liere and W.C. de Leeuw. GraphSplatting: Visualizing Graphs as Continuous Fields.

edges. Points or icons are used for the vertices, while curves are used to represent edges.

A key issue in graph drawing is how to handle very large graphs. Cluttering due to the large number of objects to display is very hard to avoid. Above a number of vertices it becomes impossible to display all vertices and edges. It will be impossible to discern between individual graphical objects and, as a result, only parts of the graph can be displayed in a single image.

In this chapter we introduce GraphSplatting, a technique that represents a graph as a continuous scalar field. There are three advantages to this approach. First, arbitrary large graphs can be represented by the field. The visual forms that represent the graph are not a collection of individual graphical objects, but a resolution independent continuous field. Second, various properties conveyed in the graph can be used to construct the scalar field. In this way, the user can choose to highlight different aspects conveyed in the graph. Third, new interaction techniques can be defined for analyzing properties in the field.

GraphSplatting is designed to be used in combination with other graph rendering methods. A continuous representation is often useful for obtaining an overview of the data and, after zooming into a detail, it can be combined with discrete graphical objects.

The chapter is organized as follows. First we discuss related work. In Section 8.3 we describe an interactive system which incorporates GraphSplatting. The system uses a new interactive graph layout algorithm based on a mass spring system. Our GraphSplatting implementation uses the output of this algorithm. In Section 8.4 we give the details of the GraphSplatting technique. We discuss the construction of the scalar field, show how this field can be rendered and discuss various interaction techniques on the field. In Section 8.5 we illustrate GraphSplatting with two applications. The first application is the analysis of the citation index of all IEEE Vis'9X papers. We show that clustering of citations leads to specific topics in visualization. The second application is the analysis of output from the UNIX profiler, `gprof`. These applications illustrate the flexibility and generality of GraphSplatting.

8.2 Related work

Visualization of graph data is well known. Excellent surveys of using very large graphs in the context of information visualization can be found in [1, 2].

Mass spring algorithms are well known for graph layout [3, 4]. For example, Gross et al. developed a system that is based on a quantification of the similarity of related objects. This similarity metric drives the spring stiffness parameters of the mass-spring system which, in its equilibrium state, will reveal multidimensional relations and adjancencies in terms of spatial neighborhoods.

Landscape visualization techniques have been used for text document visualization. Wise et al. developed the ThemeScape technique, which conveys information

about topics in text documents [5]. ThemeScapes are abstract, three-dimensional landscapes of information that are constructed from document corpora which augment a 2D landscape of text with a height dimension showing the strength of a theme in a given region. Elevation depicts theme strength, while other features of the terrain map such as valleys and peaks represent detailed interrelationships among documents and their composite themes. The authors claim that ThemeScapes are useful to rapidly gain a summary of the complete document corpus.

The work reported in this chapter was inspired by the research summarized above. Our goal was to develop an interactive graph rendering technique that could be used for a global overview of a large graph. In contrast to other application dependent techniques, GraphSplatting is general purpose and makes use of standard data visualization techniques for rendering and interaction.

The term splatting was first used by Westover for sampling and reconstruction for feed forward volume rendering [6]. The technique constructs a two dimensional image from three dimensional data. Splatting is realized by projecting a weighted footprint around a input sample onto a plane. GraphSplatting is similar to volume splatting in the sense that the final result is the sum of a number of scalar splats. However, as graph vertices are represented as points, there is no need for projection of a footprint.

8.3 Interactive Graph Visualization

Figure 8.1 shows the data flow of our interactive graph layout and rendering system. Raw data, in the form of entities and relationships, is passed to a mass spring layout algorithm, The computed vertex positions are passed to the GraphSplatting algorithm. The constructed scalar field is rendered using traditional two dimensional scalar field visualization techniques. In addition, vertex positions can bypass the GraphSplatting algorithm and be rendered directly as discrete objects.

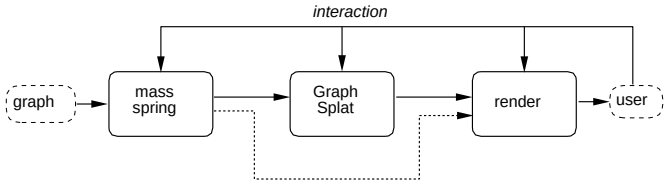


Figure 8.1: Interactive graph layout and rendering data flow.

Interaction is provided at three points in the pipeline. First, parameters that drive the mass spring algorithm are controlled by the user. These parameters are discussed in the next section when we present the details of the mass spring algorithm. Second, parameters that drive the GraphSplatting algorithm are controlled by the user. This will be discussed in Section 8.4. Third, various forms of direct manipulation on the resulting image are provided. Users can drag a collection of vertices to other positions,

thus effecting the mass spring layout algorithm. Finally, viewing transformations such as zooming and panning are also provided.

Entities are represented as a tuple of typed attribute name and value pairs

$$entity = << attr_1, val_1 >, \ldots, < attr_n, val_n >> \tag{8.1}$$

Selection of a subset of the graph to be displayed is supported in a number of ways. First, a SQL-like interface on attributes of the entities allows users to select only a subset of the graph to be displayed. Second, selection is supported with various interaction techniques with the resulting image. Groups of vertices can be selected by specifying a square region of the image. Also, vertices within an area bounded by a contour line can be selected. These interaction techniques are discussed in Section 8.4.2.

8.3.1 Mass spring layout

Mass-spring systems can be used as force directed algorithms for graph layout. The graph is viewed as a system of points with forces acting between the points. Edges are modeled as springs, and vertices are equally charged points which repel each other. The stiffness of springs is parameterized with application dependent data. A minimization algorithm computes an equilibrium configuration of the points with locally minimal energy. Mass-spring systems can be used to represent spatial adjacencies of points which are similar to each other in information space. The minimum energy in the system corresponds to an appealing layout of the graph.

Using mass-spring algorithms for graph layout is not new. Unfortunately, to compute one step in the energy minimization algorithm, most mass-spring systems are $O(N^2)$, with N being the number of vertices in the graph. This makes the usage of such systems non-interactive. Our system differ from most other approaches in that it is intended for interactive usage. Therefore, our mass spring systems differs from others in that we do not take all possible connections between vertices into account when computing the force between two vertices. Instead, the repelling factor only takes vertices into account that are within a certain radius of a vertex. This simplification (in combination with the spatial subdivision implementation described below) allows for an interactive algorithm to be implemented.

The governing equations of our mass-spring model are captured as follows. The force vector between two vertices at positions p_i and p_j is defined as the sum of an repelling and attracting force.

$$\mathbf{F}_{ij} \;=\; \mathbf{F}_{ij}^{R} + \mathbf{F}_{ij}^{A} \tag{8.2}$$

After introducing the symbol $\mathbf{v}_{ij} \;=\; \frac{p_i - p_j}{|p_i - p_j|}$ as the unit vector in the direction from p_i to p_j, we define the repelling force as

$$\mathbf{F}_{ij}^{R} \;=\; -\,c_0 \sum_{k \,\in\, S_i^R} (1 \;-\; \frac{|p_i \;-\; p_k|}{R}) \cdot \mathbf{v}_{ij} \tag{8.3}$$

in which

$$S_i^R \;=\; \{p_k \mid \quad |p_i \;-\; p_k| \;<\; R\} \tag{8.4}$$

The set S_i^R denotes all vertices that are within the radius R from vertex i.

A spatial subdivision of a uniform grid is used to speed up the calculation of the set S_i^R. Each cell of the subdivision contains a list of vertices located in the cell. In this way only a limited number of vertices which are in cells close to p_i have to be tested, resulting in an efficient computation of S_i^R.

The attracting force is defined as

$$\mathbf{F}_{ij}^A \;=\; \sum_{k \,\in\, S_i^A} w_{ik}\, |p_i \;-\; p_k| \cdot \mathbf{v}_{ij} \tag{8.5}$$

in which

$$S_i^A \;=\; \{p_k \mid is\_connected(p_i,\; p_k) \;=\; true\} \tag{8.6}$$

The set S_i^A denotes those vertices that are connected to vertex i. Each vertex in S_i^A contribute to the attracting force on vertex i. The resulting force on vertex i is the sum of these forces.

The system is parameterized with two parameters: w_{ij} and c_0. The weighting factor w_{ij} can be interpreted as the spring. w_{ij} is a user-defined parameter associated with each edge of the graph connecting vertex i with vertex j. The parameter c_0 can be interpreted as the global repulsion constant. c_0 is interactively controlled by the user.

8.4 GraphSplatting

GraphSplatting is a technique to transform the graph into a continuous field. The governing idea is that the density of vertices and edges is an important characteristic of the graph. Splatting is used to project each vertex and edge of the graph onto a two-dimensional scalar field. Instead of showing the individual vertices and edges, the variations in density are shown. Vertices and edges of the graph are represented in the field by a splatting function. Each vertex contributes to the field with a two-dimensional Gaussian shaped basis function. Each edge contributes to the field with a function which is Gaussian perpendicular to the edge and constant in the direction of the edge. The resulting field is constructed by adding all the contributions. This field is called the *splat field*.

Figure 8.2 illustrates the mapping primitive. The figure shows a cross section of the Gaussian splatting function. The width of the Gaussian (σ in figure 8.2) determines the 'smoothness' of the splat field. A large value of σ will result in smoothing out the details of the graph. Using a small value for σ will presents more detail of the graph. In the limit case $\sigma = 0$ the original vertices and edges will be represented as points and edges. The user can interactively control the width of the splats with a global parameter.

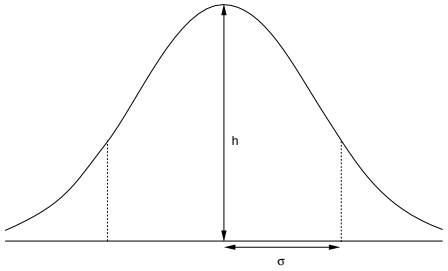


Figure 8.2: A cross section of the Gaussian splat.

The height (h in Figure 8.2) of each splat can be used for mapping an attribute of the vertices or edges. In this way, different properties of the graph can highlighted. Vertices with large attribute values will contribute more to the splat field.

GraphSplatting in itself is not intrinsically 2D, the equations for the splats could be easily extended to 3D. We choose to present the information in 2D because visualization of and interaction with 3D fields is more complicated. Also, there is no inherent three dimensionality in the data such as in 3D flow or medical volume data.

8.4.1 Splat field construction

In this section we define the construction process of the two dimensional splat field more formally. The 2D continuous function M is constructed by summing the contributions of the vertices $M_v(x, y)$ and edges $M_e(x, y)$:

$$M(x, y) \;=\; M_v(x, y) \;+\; s_e M_e(x, y) \tag{8.7}$$

The user controllable parameter s_e is used to scale the contribution of edges in the splat field. For vertices, a field is calculated by summing the contributions of individual 2D basis functions placed at vertex positions $< x, y >$:

$$M_v(x, y) \;=\; \sum_{i=1}^{N_v} f_i(x, y) \tag{8.8}$$

Each basis function is modeled as a Gaussian function:

$$G^\sigma(x,\; y) \;=\; e^{-(x^2+y^2)/\sigma^2} \tag{8.9}$$

The center of the Gaussian is placed at the vertex position. The width and height of the contribution are controlled by σ and v_i respectively. v_i can be constant or chosen from a numeric attribute of the data associated to the vertex.

$$f_i(x, y) \;=\; v_i G^\sigma(x - p_{i,x}, y - p_{i,y}) \tag{8.10}$$

For edges, a splat is used which is the convolution of an edge with a Gaussian function.

$$M_e(x,y) \;=\; \sum_{i=1}^{N_e} f_i(x,y) \tag{8.11}$$

For each edge the splat is oriented in the direction of the edge and scaled proportional to the length. For an edge with length l starting at the origin along the x-axis, the edge splat equation becomes:

$$f_i(x,y) = \begin{cases} \frac{e_i}{l} e^{-(|y|^2)/\sigma^2} & \{0 \le x \le l\} \\ \frac{e_i}{l} e^{-(x^2+y^2)/\sigma^2} & \{x < 0\} \\ \frac{e_i}{l} e^{-((x-l)^2+y^2)/\sigma^2} & \{x > l\} \end{cases} \tag{8.12}$$

e_i can be constant or chosen from a numeric attribute of the data associated to the edge.

Figure 8.3 shows a height field representation of the splats for a vertex described by equation 8.9 (left) and an edge described by equation 8.12 (right).

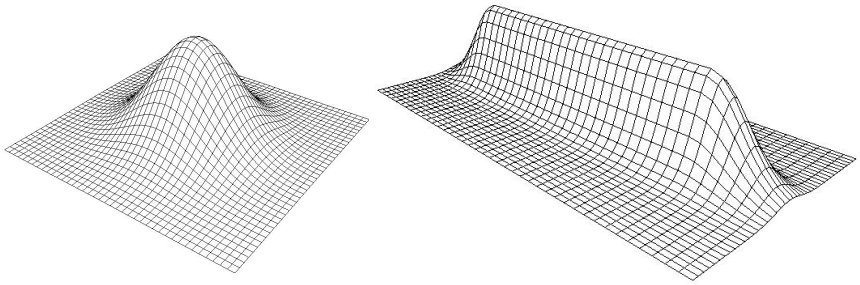


Figure 8.3: Height field representation of the splat of a vertex (left) and an edge (right).

The computation of the splat field is done on a grid with a user controlled resolution. Higher resolution grids give a better representation of the field, however they take longer to compute. In the current implementation the contributions of each vertex splat to a particular pixel is estimated by using the distance from the vertex to the center of the pixel. For a line splat, the distance to the edge is calculated. In a final normalization step, the splat field is scaled into a fixed range for zero to one. This step facilitates the visualization of the splat field easier.

Figure 8.4 illustrates these equations. The top left panel shows a graph with 4 vertices and edges. The other panels give the GraphSplatting representation of the graph. Each panel uses a different value of σ, the width of the Gaussian. Due to the normalization, a larger value for σ results in larger dark areas.

A number of user controllable parameters are used to construct the splat field. Parameters should be functional in the sense that setting a parameter value should serve a specific purpose:

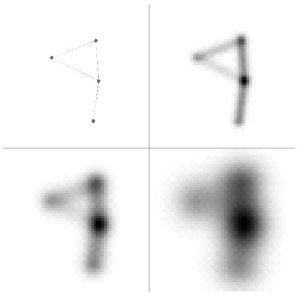


Figure 8.4: A graph of 4 vertices and edges displayed as discrete objects (top left) and as a continuous splat field using various width values for the Gaussian function. Large values in the splat field are represented as dark pixels in the image.

- s_e: The relative edge strength determines the influence of the edges in the splat field. When the user is mainly interested in clustering of vertices in the graph and not in connection patterns, this value should be set low. If, on the other hand, interconnection patterns are important a high value for this parameter can be chosen.

- σ: The width of the splat determines the level of detail presented in the splat field. For an initial overview of the field, a high value for σ should be chosen. To reveal more detail the value can be set to a lower value.

- v_i: Numeric attributes of vertices can be used to represent importance of vertices. If no mapping is used, all vertices are equally important, no mapping is necessary, and therefore v_i is constant. If the user is interested in the distribution of a certain attribute associated with vertices, mapping this value to v_i results in a splat field in which vertices (or clusters of vertices) that have an high attribute value are highlighted.

- e_i: The same principle holds for mapping of an attribute of edges. For some types of graphs the edges have a strength attribute which can be represented in the splat field.

8.4.2 Splat field visualization

Standard visualization techniques can be used to visualize the function M. For example color mapping, 3D height plots, or contours can be used to visualization the splat field.

Although the GraphSplatting technique constructs a continuous scalar field, it is important to realize that the underlying data is a graph. As such, the analysis of the field is often combined with a representation of discrete graphical objects. These objects can be rendered into the field. Using visualization techniques for scalar fields in combination with a the graph has a number of specific advantages and disadvantages :

- Height map. A three dimensional height map of the field is valuable to show the main structure of the field. A drawback of this visualization method is that interaction is difficult. For example, picking vertices or specification of a region is difficult in three dimensions.

- Color coding. A two-dimensional view of the field in which the value is shown using color is useful for interaction. Different color codings can be used (for the black and white images in this chapter the value was mapped to the gray level. For the color plates a color coding was used.). By combining the splat field with a discrete representation of the graph, interaction with individual vertices can be realized. Point-and-click of vertices to get additional information, dragging a vertex to a new position, etc, are interactions that are best performed in 2D.

- Iso-value contours. Contours can be used in addition to other display techniques. It can be used to show the boundary of specific clusters. A contour can be used as a criterion to select all data items within the area contained bounded by the contour.

The system offers all the described visualization techniques and their combinations. In this way a user can decide which techniques is applicable for a specific question.

8.5 Applications

We have used GraphSplatting for the visualization of graphs. Graphs vertices are displayed as discs and edges as straight lines. A user can point and click on vertices for detailed inspection of associated the vertex data.

8.5.1 IEEE Vis'9X citation index

We have applied GraphSplatting to the analysis of the IEEE Vis'9X citation index. The input data set are BibTeX entries of all papers in the proceedings of the IEEE Vis'9X conferences and all references to papers in this set from other papers in the set. The data set consists 599 BibTeX entries and 881 references. The graph represents papers as a vertices and references as edges.

The goals of the visualization were twofold:

- To test the hypothesis that topics in visualization could be identified by only analyzing the density of the references. The motivation of this hypothesis is that papers in one topic often refer to other papers in the same topic.
- To identify *influential* visualization papers. The influence of a paper is loosely defined as a paper which has influenced a body of other papers.

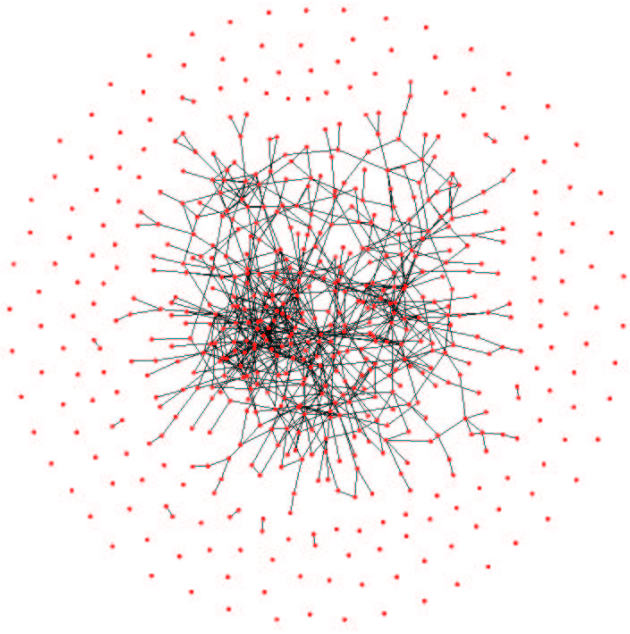


Figure 8.5: All papers published in IEEE Vis'9X conferences, represented as discs and references between the papers represented as lines.

Figure 8.5 shows the output of the spring mass algorithm. The spring stiffness was constant for all edges. As can be seen, aside from the papers which are not referenced and do not reference papers, there is a single main graph which can not be partitioned without breaking edges. The number of elements in the graph is too large and cluttered to get insight into the structure of the graph.

Figure 8.6 shows the splat field as a height map. The main clusters of the graph show up as peaks in the graph. Higher peaks mean higher density of edges and vertices. These peaks can be interpreted as strongly represented topics at visualization. The most prominent peak coincides with papers that are related to the topic of flow visualization. Lower peaks can be discerned for other visualization topics such as volume rendering, visualization systems, simplification, and information visualization.

Figure 8.7 shows a slightly zoomed in 2D rendition of the same splat field. One can clearly see various clusters of papers. For example, the papers in the large dark region in the middle of the image deal with flow visualization. The (smaller) region

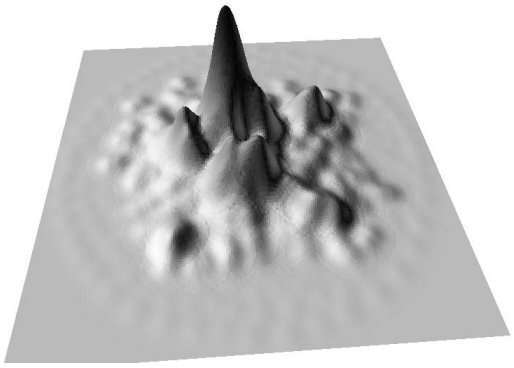


Figure 8.6: Height map representation of the graph showing the main topic clusters. (see color plate on page 155)

below are papers describing visualization systems. The region on the right are volume visualization papers. Discrete discs (red dots) and lines (yellow lines for incoming references and blue lines for outgoing references) are used for representing vertices and edges are also drawn as annotation. The region at the top left contains information visualization papers. The distance to the other peaks in the field clearly illustrates the distinction between information visualization and other data visualization topics.

Figure 8.7 also illustrates some interaction techniques. The contour line around the right region is used as a selection criterion. In this way all papers in an area, in this case volume visualization, can be selected. Also, the influence of a paper is shown by drawing the edges representing references to the selected papers. The user can also pick individual papers and show all information related to that individual paper.

Temporal slicing of the input data is supported through the selection mechanism. The YEAR attribute is used to select all papers within a range of years. A slider can be attached to this operation to implement a sliding window.

8.5.2 Gprof

We have applied GraphSplatting to the analysis of the call graph produced by a profiling program. For this we used the UNIX profiler `gprof`. `Gprof` provides information about the dynamic behavior of the program. For example, the number of times a function is called, the amount of time the program is in a function, etc. The goal of the visualization is to analyze various dynamic aspects of the program.

In this particular case, the program was the Linux kernel module of the Matrox G400 driver. This driver utilizes the G400 hardware to accelerate 3D graphics rendering. The profile was collected during a 15 minute session in which various 3D programs were executed. The call graph generated by the profile was used to define the input graph. Functions are represented as vertices and edges represent called functions. The input graph consists of 762 vertices and 1001 edges.

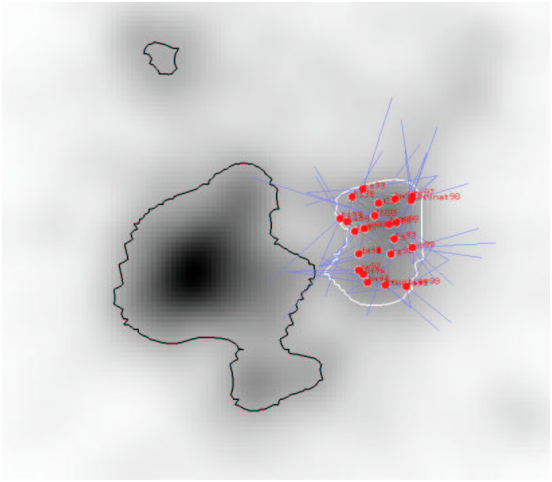


Figure 8.7: Interacting with the citation index. The influence of a group of papers is drawn with yellow (incoming) and blue (outgoing) references. Papers in the region selected by the highlighted contour on the right are shown as discs. (see color plate on page 155)

The mass spring algorithm uses the relative number of calls to a function to parameterize the spring stiffness. In this way if function A calls function C a hundred times and function B calls C only once, the strength of the spring from A to C is 100 time larger than the spring from B to C. This mapping assures that functions which get called by more than one function are "pulled" towards the most frequent caller.

Figure 8.8 shows a 2D rendition of the splat field. Discrete representations of vertices (red dots) and edges (light grey lines) are also drawn. Clusters of functions can be identified in the graph as well as the splat field. In the splat field no attribute mapping for vertices is used: the height of the vertex splats (v_i in equation 8.10) is taken to be constant. For edges, the relative number of calls was used as the scaling factor (e_i in equation 8.12).

Figures 8.9 and 8.10 provide insight in the performance of the G400 driver. The time spent in each function was used to modulate the splat height of the vertices (v_i in equation 8.10). Edges do not contribute to this splat field (s_e in equation 8.7 is set to zero). Discrete discs (red dots) are used for representing vertices are also drawn as annotation.

In the splat field the most time consuming functions and functions related to them are easy to identify. Using selection based on contours (see figure 8.10) one can easily gain more information on the functions which are crucial for program performance. Yellow lines for caller functions and blue lines for callee functions are used for representing edges. The bottleneck for this particular case were lighting computations (which the G400 does in software). This can be seen in the top left part of the image.

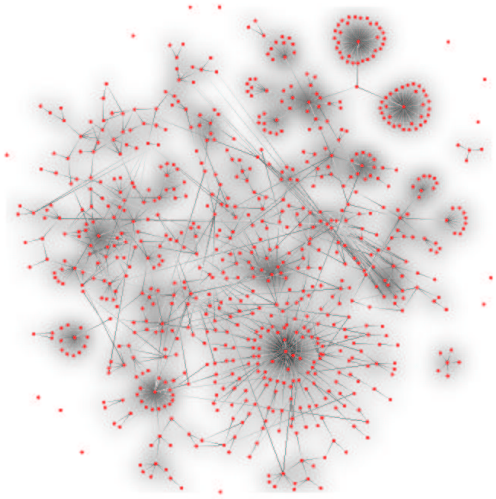


Figure 8.8: Overview of the call graph and associated splat field. The relative number of calls of a function from its calling function is mapped to the spring stiffness. (see color plate on page156)

8.6 Conclusions

In this chapter we have introduced GraphSplatting, a method which transforms a graph into the splat field, a two-dimensional continuous scalar field. There are three advantages to using GraphSplatting to represent a graph. First, arbitrarily large graphs can be visualized without cluttering. Second, application dependent information contained in the graph can be used to control the transformation. Third, various data visualization and interaction techniques can be used to analyze the data.

GraphSplatting is designed to be used in combination with other graph rendering methods. The continuous representation is often useful for obtaining an overview of the data and, after zooming into a detail, can be combined with discrete graphical objects. In addition, GraphSplatting is used to interactively gain an overview of different aspects conveyed of the graph. By varying the mapping parameters, the user can change the information conveyed by the splat field without effecting the layout of the graph.

In the future we will investigate how alternative metrics can be used to drive the splat field construction. In addition, we will use graphics hardware to accelerate the splat field construction. These hardware accelerated techniques are very similar to those presented in [7].

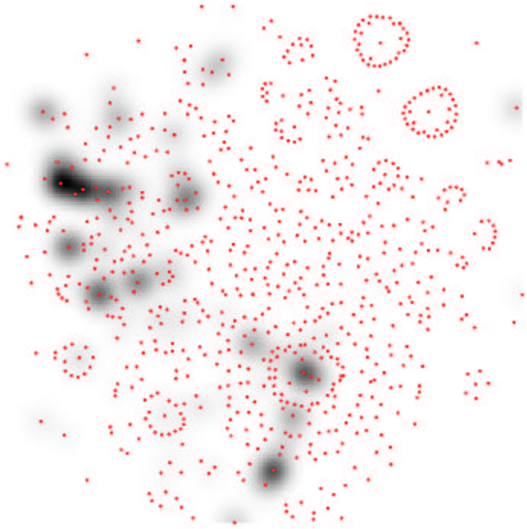


Figure 8.9: Dynamic profile of the Matrox G400 driver. The height of the splats is scaled with the amount of time spent in the corresponding function. (see color plate on page156)

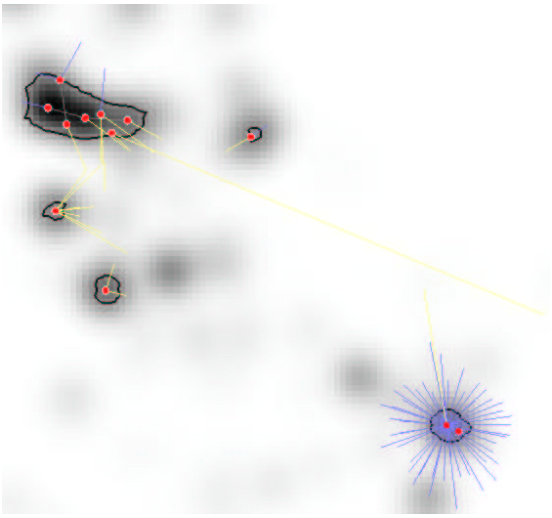


Figure 8.10: Detail of the time mapped profile splat field of a program. The most time consuming functions are selected using contour selection.

References

[1] I. Herman, G. Melancon, and M.S. Marshell. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–44, 2000.

[2] G. di Battista, P. Eades, R.A. Tamassia, and J.G. Tollis. *Graph Drawing*. Prentice Hall, 1999.

[3] R.J. Hendley, N.S. Drew, A.M. Wood, and R. Beale. Narcissus: Visualizing information. In S.K. Card, J.D. Mackinlay, and B. Shneiderman, editors, *Readings in Information Visualization*, pages 503–511. Morgan Kaufmann Publishers, 1999.

[4] M.H. Gross, T.C. Springer, and J. Finger. Visualizing information on a sphere. In *Proceedings Symposium on Information Visualization*, pages 11–16. IEEE Computer Science Press, 1997.

[5] J.A. Wise, J.J. Thomas, K. Pennock, and D. Lantrip. Visualizing the non-visual: Spatial analysis and interaction with information from text documents. In S.K. Card, J.D. Mackinlay, and B. Shneiderman, editors, *Readings in Information Visualization*, pages 442–450. Morgan Kaufmann Publishers, 1999.

[6] L. Westover. Footprint evaluation for volume rendering. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 367–376, 1990.

[7] W.C. de Leeuw and J.J. van Wijk. Enhanced spot noise for vector field visualization. In G.M. Nielson and D. Silver, editors, *Proceedings IEEE Visualization'95*, pages 233–239. IEEE Computer Society Press, October 1995.

Chapter 9

Interactive Visualization of Multidimensional Feature Spaces [1]

Image similarity models characterize images as points in high-dimensional feature spaces. Each point is represented by a combination of distinct features, such as brightness, color histograms or texture characteristics of the image, etc. For the design and tuning of features, and thus the effectiveness of the image similarity model, it is important to understand the interrelations of individual features and the implications on the structure of the feature space.

In this chapter we discuss an interactive visualization tool for the exploration of multidimensional feature spaces. Our tool uses a graph as an intermediate representation of the points in the feature space. A mass spring algorithm is used to layout the graph in a 2D space in which arrangements of similar images are attracted to each other and dissimilar images are repelled.

The emphasis of the visualization tool is on interaction: users may influence the layout by interactively scaling dimensions of the feature space. In this way, the user can explore how a feature behaves in relation to other features.

9.1 Introduction

Visual information retrieval systems allow images to be retrieved from data repositories subject to a user-defined query. Although the preferred mode of querying an image is semantic, queries are usually based on syntactic features of the image (such as color, texture and object shape). The discrepancy that results from using syntactic features to satisfy semantic queries causes a basic problem with the traditional query/response style of interaction. In addition, syntactic features are context-sensitive in that a feature may successfully be used in one context, but can be inadequate in a different context.

[1] This chapter has been published as: R. van Liere, W.C. de Leeuw and F. Waas. Interactive Visualization of Multidimensional Feature Spaces. In C. Shaw and D. Ebert, *New Paradigms for Information Visualization*, pages 58-71, 2000.

Hence, it depends on the image set which feature (or combination of features) is most useful for a search.

Image similarity models for visual information retrieval are a well studied subject. Such models usually represent an image as a point in a multidimensional feature space where *similarity* of two images is expressed by the distance between their points in the feature space. A larger similarity/dissimilarity corresponds to a smaller/larger distance of the points. Traditionally, image retrieval systems return a list of images, sorted by similarity to the query image. This list is then presented to the user as a list of thumbnail images. Unfortunately, such a presentation can be disorienting since relationships between the images of the answer set are largely ignored and only the similarity to the query image is assessed. More appropriate would be a presentation of the multidimensional space where the similarity relationships of all images in the vicinity of the query image, i.e., the answer set, are preserved and presented in a way that is easy and intuitive to grasp for the user.

In this chapter we represent the multidimensional feature space of images with a graph. A vertex in the graph represents a point in the space, while edges represent similarity relationships between images. The graph is displayed in such a way that vertices with strong similarities are attracted to each other and dissimilar vertices are repelled. The advantage of this presentation is that it gives a global overview of points in the feature space as well as similarity relations among points. In addition, the method allows a user to interactively scale each dimension of the feature space. By interactively scaling a dimension, the user can explore how a feature behaves in relation to other features.

The focus of our research is not to develop interfaces for end users of visual information retrieval systems. Rather, we are developing a framework in which feature developers can experiment with features on wide varieties of image sets. Our framework allows developers to gain insight into the weak and strong points of an individual feature, as well as insight into combinations of features. We believe that interactive interfaces, in which developers continuously control one or more dimensions of the feature space, are very intuitive for understanding the effect that features have on the underlying similarity model.

The remainder of this chapter is organized as follows: After reviewing related work, in Section 9.3, we review the building blocks of our system. In Section 9.4 we present experimental results obtained with a real-world data set, detailing different scaling effects. We discuss possibilities and limitations of application of our system in Section 9.5 and present our conclusions in Section 9.6.

9.2 Related Work

User interfaces for visual information retrieval systems have gained much attention recently, see e.g., [1, 2]. Research is underway in defining new ways of representing the content of visual archives and the paths followed during a retrieval session. In

retrieving visual information, high level semantic concepts are often used together with perceptual features in a query.

Mass-spring algorithms are well known for graph layout [3, 4, 5]. For example, Gross *et al.* developed a mass-spring based system in which a similarity metric is quantified for objects in financial applications. This similarity metric drives the spring stiffness parameters of the mass-spring system which, in its equilibrium state, will reveal multidimensional relations and adjacency in terms of spatial neighborhoods.

Given a set of n objects in a K-dimensional space and a dissimilarity measure between objects, Multidimensional scaling (MDS) computes a configuration of points in a low-dimensional Euclidean space so that the Euclidean distances between two points match the original dissimilarities between the corresponding objects as precise as possible [6]. The MDS procedure is realized by applying a least-squares technique to an objective function that penalizes the overall disparity between distances and dissimilarities. A minimum of the objective function yields the desired configuration. A number of commercial and research prototype image retrieval systems, which are based on MDS to display similarity, have been developed, including QBIC [7] and a research system built at Stanford Vision Laboratory [8]. QBIC displays the returned images as a list sorted by dissimilarity from the query. The Stanford system applies MDS to the dissimilarity matrix and places image thumbnails at the coordinates of the resulting two-dimensional projection.

9.3 Methods

9.3.1 Similarity Metrics

For any given image, a feature is expressed as a k-dimensional vector $f_i = \langle v_{i1}, v_{i2}, \ldots, v_{ik} \rangle$. The dimension of the vector may vary significantly from feature to feature. For example, the brightness of an image maybe noted as a single value, i.e., $k = 1$, whereas a color histogram may consist for instance of 128 values, i.e., $k = 128$.

Now, given a set of M features, we define the *feature vector* of an image as the composition of the single feature values:

$$F = \langle v_{11}, \ldots, v_{1k_1}, v_{21}, \ldots, v_{2k_2}, \ldots, v_{M1}, \ldots, v_{Mk_M} \rangle$$

Accordingly, the dimension of F is $K = \dim(F) = \sum_{i=0}^{M} \dim(f_i)$. For some $\lambda_i \in [0; 1]$, let S be a matrix of the shape

$$S = \begin{pmatrix} A_1 & & & \mathbf{0} \\ & A_2 & & \\ & & A_3 & \\ & & & \ddots \\ \mathbf{0} & & & A_M \end{pmatrix}$$

with matrices $A_i = \lambda_i \cdot I_{\dim(f_i)}$ where I_n denotes the identity matrix of order n.

Using S as a scaling matrix, we can compute the distance matrix D for a set of images $I_1, \ldots, I_n$ as

$$D = \begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{pmatrix}$$

with $d_{ij} = \| S\,|(F_i - F_j)| \|$. F_i and F_j are the feature vectors of image I_i and I_j, respectively. The d_{ij} can be interpreted as the *scaled similarity metric* between image I_i and I_j.

Examples. To illustrate the concept of features, we survey some features widely used in the literature.

Color-based features. Color-based features operate solely based on the color information contained in the image. The most prominent representatives of which are color histograms where the images are usually first dithered and the incidence of the single colors is determined afterwards [9, 7, 10]. Color histograms have been frequently used in related work as reference technique to assess the performance of new features [11]. Also simpler features that reduce even to a single scalar value are conceivable like the information of how many different colors occur, the brightness, or the contrast of an image. Especially with cliparts, i.e., non-photographic images, these very simple features are often of high distinctive power. On the other hand, color histograms can be refined to reflect also some spatial information of the image by allowing for color transitions, i.e., these histograms do not record the number of pixels per color but the number of pairs of neighbored pixels that make up a certain color transition. For example, with a color palette of 16 colors the transition histogram covers 256 color transitions. Depending on the particular feature, the quality can be enhanced by applying filters like despeckle or blur filters to the image as a pre-processing.

Texture-based features. Texture-based features capture structures within the images. The most typical representatives are approximations with periodic functions like the Fourier transform. Here, the feature vector corresponds to the sequence of coefficients found. Other important members of this class include Gabor filters and wavelets. Texture-base features are particularly successful when applied to genres of images where color information is of lesser importance, e.g., air photography [12].

9.3.2 Layout

A graph is used as an intermediate representation of the points in the feature space. Vertices represent the points in the K-dimensional feature space while edges model the similarity relationship between points.

For the layout of the graph a mass-spring system is used. Edges are modeled as springs. A minimization algorithm computes an equilibrium configuration of the points with minimal energy. Unfortunately, to compute one step in the energy minimization algorithm, most spring mass systems are in $O(N^2)$, where N is the number of vertices. This makes such fully connected mass-spring systems non-scalable for interactive usage.

To overcome this scaling problem, we define an alternative mapping from the distance matrix to the graph. Instead of generating a fully connected graph, we generate a graph in which only *highly similar* vertices are connected. For this, we introduce a threshold distance T. When the graph is constructed only edges corresponding to a distance that is less than T are taken into account, i.e., vertices i and j have an edge if and only if $d_{ij} < T$.

The governing equations of the interactive mass-spring layout model are captured as follows: denote the position of vertex i in visualization space as p_i and the position of point i in multidimensional space as P_i. The force applied by a vertex j onto a vertex i depends on the discrepancy between the $\|p_j - p_i\|$ and $\|P_j - P_i\|$. If the discrepancy of these two distances is large than the force will be large. The resulting total force applied to a vertex is the sum of all forces on the vertex. The mass spring algorithm will minimize the total discrepancy of the distances.

With $\mathbf{v}_{ij} = \frac{p_i - p_j}{\|p_i - p_j\|}$, the unit vector in the direction from p_i to p_j, we define the force between vertices i and j as

$$\mathbf{F}_{ij} = \begin{cases} w_{ij}(d_{ij} - \|p_i - p_j\|) \cdot \mathbf{v}_{ij} & \text{if } \|p_i - p_j\| < R \text{ or if } d_{ij} < T \\ 0 & \text{otherwise} \end{cases}$$

The visualization space parameter R can be set by the user.

This formulation allows an efficient algorithm to be implemented. The computation of F_i is sped up in two ways. First, a uniform grid of radius R around vertex i is used to quickly test and select only those vertices in the neighborhood of i. Second, edges are used to select only those vertices of a distance smaller than d_{ij} to i. In this way, instead of testing all vertices, only a limited number of vertices have to be tested (see [13]).

9.3.3 GraphSplatting

GraphSplatting is a technique to transform the graph into a continuous field. It is based on the observation that the density of vertices is an important characteristic of the graph. Splatting projects each vertex of the graph onto a two-dimensional scalar field. Instead of showing the individual vertices and edges, the variations in density are shown. Vertices of the graph are represented in the field by a splatting function. Each vertex contributes to the field with a two-dimensional Gaussian shaped basis function. The resulting field is constructed by adding all the contributions. This field is called the *splat field*.

Figure 9.1 illustrates the mapping primitive. The figure shows a cross section of the Gaussian splatting function. The width of the Gaussian (σ in Figure 9.1) determines the 'smoothness' of the splat field. A large value of σ will result in smoothing out the details of the graph. Using a small value for σ will present more detail of the graph. In the limited case $\sigma = 0$ the original vertices will be represented as points. The user can interactively control the width of the splats with a global parameter.

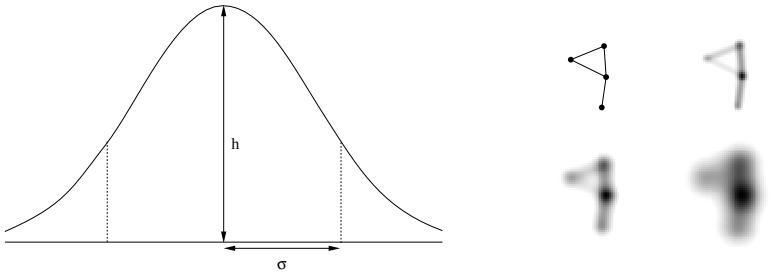


Figure 9.1: Visualization of a splat field. The left panel shows the mapping parameters for the base function; The right panel shows three splat fields of a graph with 4 vertices and 4 edges (upper left). The splat fields have different splat widths.

The height (h in Figure 9.1) of each splat can be used for mapping an attribute of the vertices. In this way, different properties of the graph can be highlighted. Vertices with a large attribute values will contribute more to the splat field that vertices with low attribute values.

GraphSplatting is designed to be used in combination with other graph rendering methods. A continuous representation is often useful for obtaining an overview of the data associated with the vertices of the graph. After zooming into a detail, it can be combined with other graph visualization methods.

9.4 Results

Test Image Set. We applied our methods to a synthetic test set of 3276 images. The test set consisted of 36 groups of images with distinct hue values. Each group had 91 textures of varying frequency and orientation. For each image, 6 feature vectors were computed: a 1 four-dimensional gabor feature vector for texture analysis and 5 distinct color-based features vectors. The color-based features vectors including a hue histogram, a hue histogram of the center region of the image, and 3 hue transition histograms. For transition histograms, the hue is first dithered to 16 bins, then the histogram of the 256 resulting combinations is recorded. As a pre-processing step, the images were segmented into 32, 128, and 256 tiles, and each tile was replaced by its dominant hue. The dimensionality of the feature space spanned by the 6 features vectors is 804.

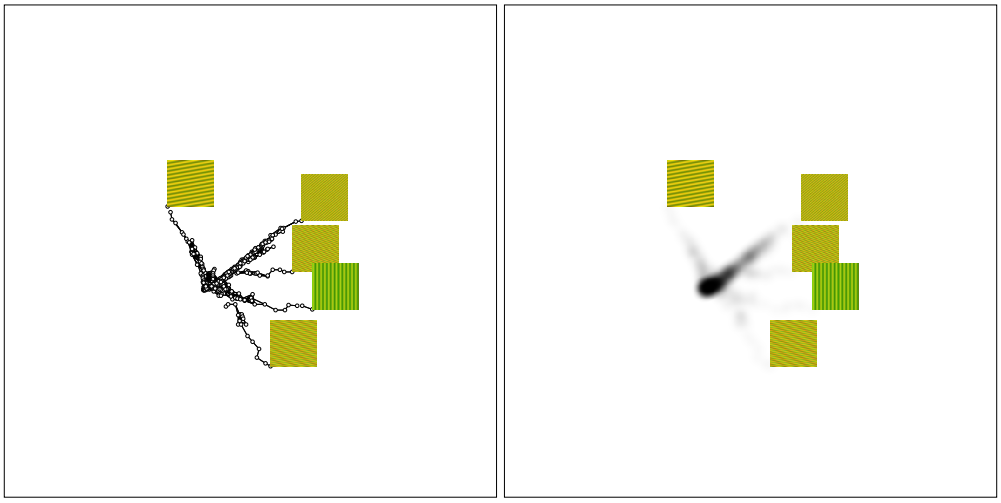


Figure 9.2: Two views of a graph arrangement for the test set. The left panel shows graph view with vertices and edges. The right panel shows the splat field. Some vertices are annotated with a thumbnail image. (see color plate on page 157)

Figure 9.4 shows a snapshot of the user interface. The left panel shows the graph view: an arrangement of the graph in the visualization space. Small dots are used to represent vertices. Grey lines represent edges between points with distances below the threshold distance T. Edges also provide additional feedback on the state and progression of the layout algorithm. For example, very long edges will indicate that the layout algorithm has not reached an equilibrium. Some selected vertices are annotated with a thumbnail image. The right panel shows the splat field, which is color encoded from white (low density values) to black (high density values). In Figure 9.4, the mass spring algorithm has reached an equilibrium. Users can drag vertices to other positions, after which the mass spring algorithm will compute a new equilibrium. Animation is used to display each step of the mass spring system evolving to an equilibrium. In this way, the user can study how an arrangement evolves towards another.

The graph provides a 2D view in which the images are displayed according to their mutual dissimilarities and similar images are clustered. A problem with the graph view is the potential cluttering, making it difficult to estimate density of vertices in dense regions. The splat field provides a 2D view of a continuous density field. Colors are used to show which areas have a high density of vertices. In this way, the user can see in a glance which images are similar.

Scaling is illustrated in Figure 9.3. Each panel of the 3x3 matrix show the splat field of the graph layout in a equilibrium. Each row has scaled the hue histogram feature vector by incrementing the corresponding λ by 0.5. Similarly, each column has scaled the gabor feature vector with increasing λ settings.

The influence of scaling the hue histogram feature vector in combination with scaling the gabor feature vector can be analyzed from the matrix. For example, the arrange-

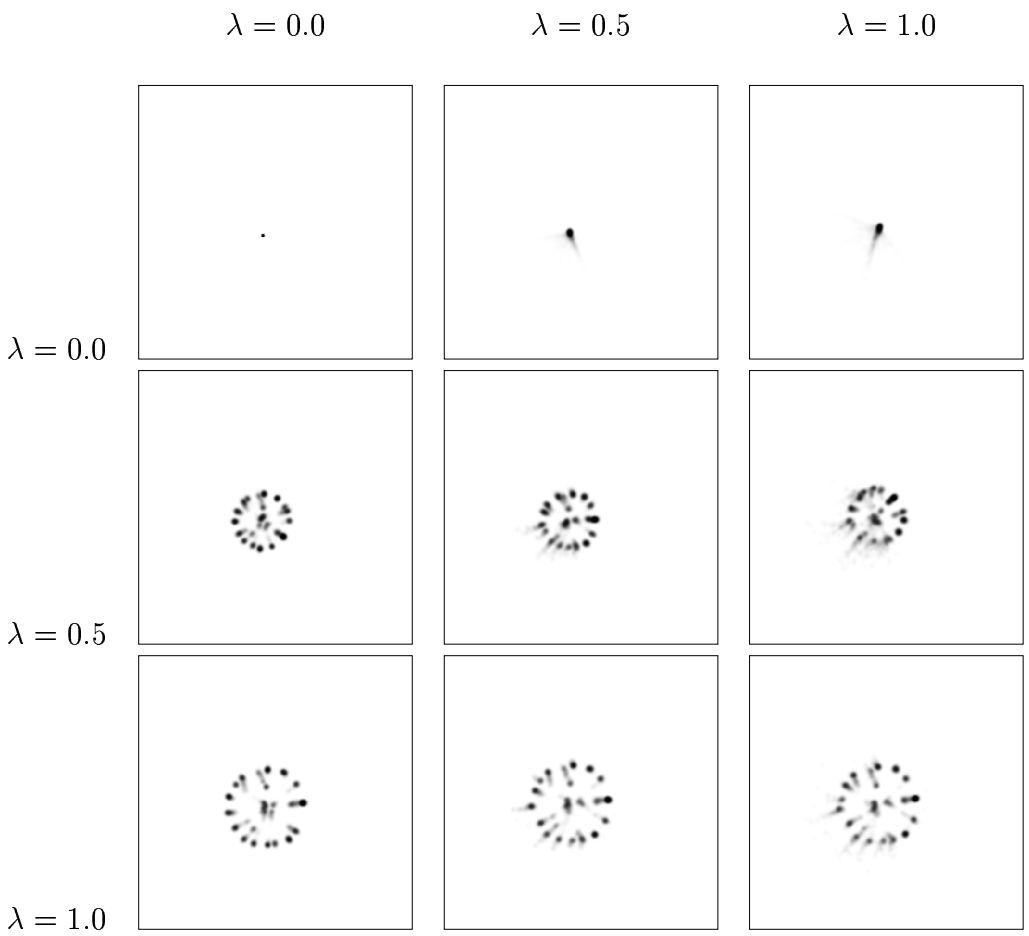


Figure 9.3: Nine splat fields with different scaling factors. Rows have increasing λ values for the hue histogram feature vector. Columns have increasing λ values for the gabor feature vector. (see color plate on page 158)

ments shown in the first row are very different from the arrangements in the last row. In addition, the circular pattern of small clusterings seen in the last row, can already be discerned in the second row. This observation indicates that the hue histogram feature vector is dominating the gabor feature vector for this test set.

Corel Image Collection. We also experimented with images taken from the Corel Image Collection [14]. A set of 200 images were selected across different genres, yet, at the same time care has been taken that there is a small fraction of images per genre that would be commonly regarded as "similar". For example, images of similar objects like sailing boats, or image of objects which differ in lighting characteristics or camera positions only. The 6 feature vectors mentioned above were computed for each image.

The four panels in Figure 9.4 demonstrate the effects of different λ settings. These panels shows that different weightings of features lead to distinctly different clusterings of the graph. Scaled features are meaningful in their own way but provide significantly different separation of different regions or images.

In this case the gabor and hue histogram feature vector were scaled. The upper left panel shows the graph with λ setting for the gabor feature at 1.0 and the λ setting for the hue histogram feature at 0.0. In the upper right panel, the λ settings were 0.8 and 0.2. In the lower left panel, the λ settings were 0.2 and 0.8. In the lower right pane, the λ settings were 0.0 and 1.0. The threshold value was set so that each graph contained approximately 1100 edges.

Each panel show a very different structure of the underlying graph. The upper left panel shows a structure with 4 clusters of vertices. Clusters are connected with relatively few links. The lower right panel shows a structure with dense cluster.

To illustrate the scalability of the splat field, we have applied our methods to larger image sets. Figure 9.4 shows the splat fields of four image sets taken from the Corel Image Collection. The sets have 1000, 4788 and 10000 images respectively. The same set of features as above was computed for both sets. To generate the snapshots, the mass spring algorithm was used with the λ of the gabor feature vector set to 1.0. All other λ factors were set to 0.0.

The top row shows the graph view for each layout. Graph views with 1182, 33524 and 86521 edges are very cluttered and it is very difficult to determine which areas contain images that are similar. The bottom shows the splat field for each layout. Here, the structure of the graph is clearly shown. This is useful particularly in areas of high vertex density, i.e., those areas in which images have high similarity. Also, the structure of the graph is very similar for the four image sets.

9.5 Discussion

The framework discussed in this chapter allows developers concerned with the design and tuning of features to experiment with the precision of particular features, feature

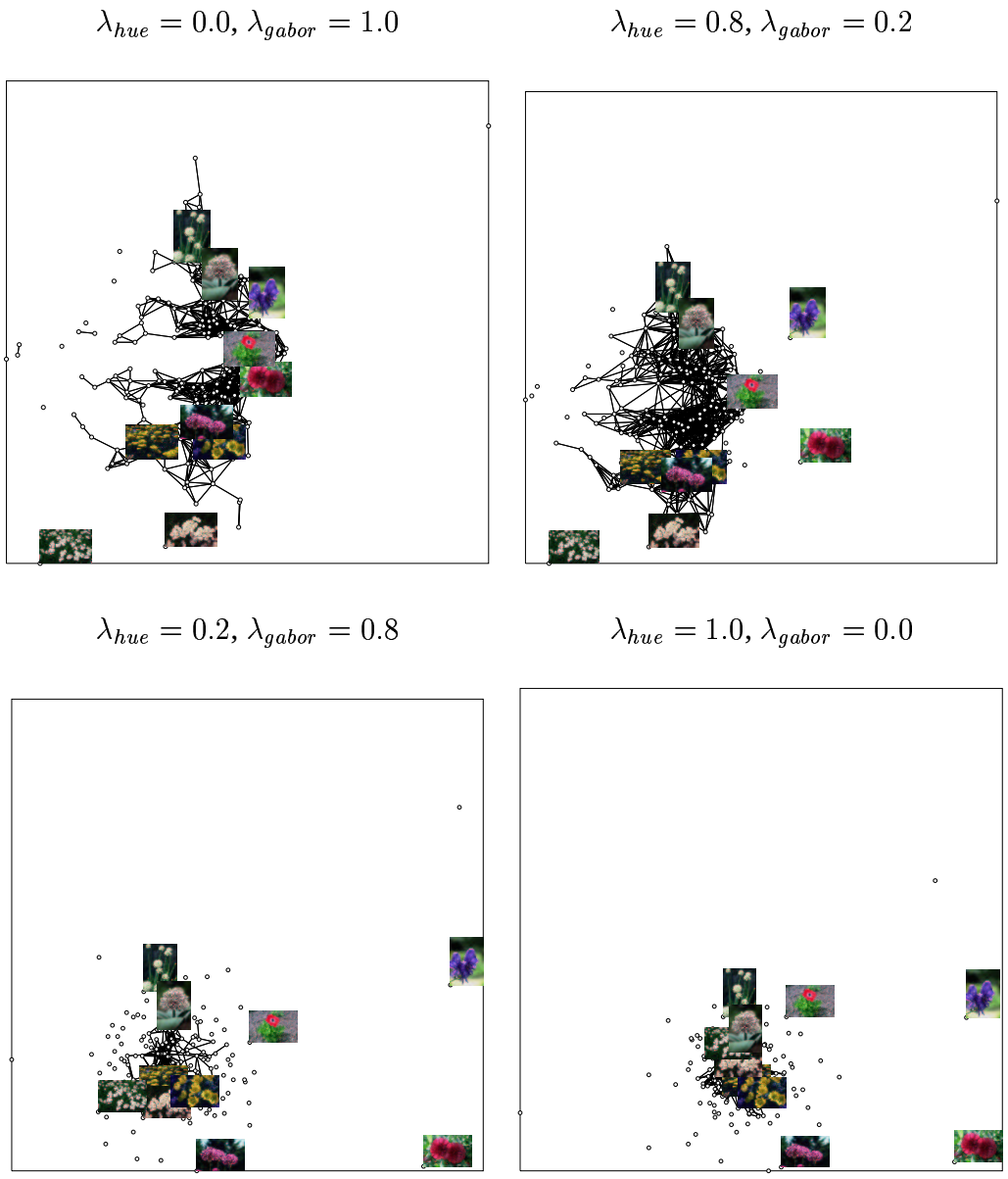


Figure 9.4: Four panels showing the graph with different scaling factors for the hue histogram and gabor feature vectors. Ten vertices are annotated with a thumbnail image. (see color plate on page 159)

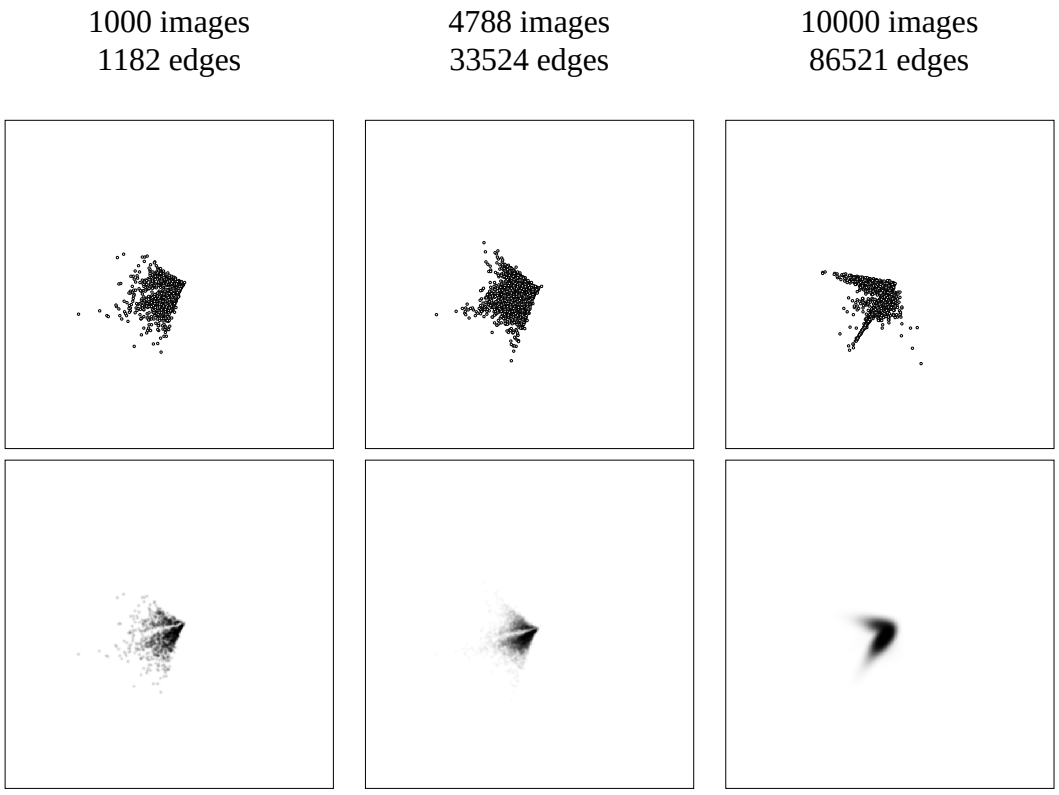


Figure 9.5: Scalability of splat fields. The top three panels show the graph views of three image sets. The bottom three panels show the splat fields. The gabor feature vector was scaled to 1.0.

distributions, similarity models, and the visualization of similar images. The interactive visualization tools are tailored towards the exploration and presentation of the underlying multidimensional spaces.

9.5.1 Visualization

The layout algorithm detailed in Section 9.3.2 generates arrangements in which similar images are attracted and dissimilar images are repelled. Since related images are grouped together in order of increasing dissimilarity, the density of the images can be interpreted as a measure of image similarity. Images are displayed either as points (useful for density distributions), thumb-nails (useful for visual similarity comparisons), or the complete image.

However, the interactive nature of the interface provides additional advantages regarding both the layout and modification of the points in the space by different feature weightings.

- Layout

Besides the actual layout algorithm, the tool comes with a whole array of interactive elements including zooming, drag-and-drop of vertices, inspecting a vertex properties, highlighting of neighbored vertices and connecting edges etc. This enables users to disentangle dense areas and study neighborhoods of individual vertices.

The animation of the layout algorithm gives an immediate impression of the strength of the links and components. Additionally, the user can also adjust the velocity of the convergence of the system to study these effects also in slow-motion.

- Scaling

  It is well known that there is no *universal* concept of similarity but similarity always depends on both the properties of the query image and the images stored in the repository. For different sets of images, features differ in their effectiveness. In addition, the effectiveness may vary even from one region of the feature space to another [15]. Thus, weighting and scaling of features is a necessity.

  Studying these effects with statistical methods like cluster analysis etc. is often not satisfactory. Capturing the structure of the points in space in order to describe the effects of scaling is computational expensive and results are difficult to grasp. In contrast, scaling the influence of features interactively, helps to grasp these effects in an immediate and evident way.

Our experience has been that for the bulk of images the weighting is of little influence—these images appear very similar under many different features, particularly, since features often subsume other features. However, there are also areas where the influence of a few, sometimes even one single feature, is crucial. Interactive and animated scaling of single features makes it easy to explore and analyze the impact of individual features. Our layout provides for the visualization of areas which is facilitated by the threshold parameter. Modifying this parameter enables pruning of the similarity relationships in the graph to blot out regions that are of little interest. It does not matter exactly how distant dissimilar images are from a given image, as long as they are far in relation to similar ones.

A splat field is used to show the density of one single dimension of the feature space by mapping the feature value of a feature to a splat. Since splat fields are continuous representations of the discrete graph, they allow for the visualization of very large graphs. Individual vertices will not be discerned, but the continuous field will contain density information that can be used to determine clusters of similar images.

9.5.2 Comparison with other MDS based systems

The use of a mass-spring system for the layout of a multidimensional space is an approximation to MDS. Recall that MDS uses a least-square technique to define an

objective function that penalizes the overall disparity between distances and dissimilarities. The MDS objective function may be interpreted as the total energy of a fully connected mass-spring system, with a vertex for each image, and springs connecting each vertex. The relaxed length of a spring connecting two vertices is given be the dissimilarity between the corresponding pair of images. The actual length of the spring is the Euclidean distance between vertices. The equilibrium of this spring system corresponds to the minimum of the MDS objective function.

Our method differs from MDS when the graph is not fully connected. Constructing a graph from the distance matrix uses a distance threshold T. When the graph is constructed only edges corresponding to a distance that is less than T are taken into account, i.e., vertices i and j have an edge if and only if $d_{ij} < T$. For example, when $T = 0$, the graph will have no edges and if $T = \infty$, the graph will be fully connected.

In the case of a not completely connected graph, our mass-spring algorithm will result in groups of similar images. Mutual distances between images within a group can be interpreted as a measure of similarity. Distances between groups have no meaning, but the similarity between images in different groups is known to be larger than the distance threshold.

9.6 Conclusions

The framework discussed in this chapter allows image feature developers to experiment with the precision of particular features, feature distributions, similarity models, and the visualization of similar images. The interactive visualization tools are tailored towards the exploration and presentation of the underlying multidimensional spaces.

We have demonstrated that our tools can be used to gain insight into the strengths and weaknesses of features and how they perform in combination with other features. We believe that interactive interfaces, in which users continuously control one or more dimensions of the feature space, are very intuitive for understanding the effect that features have on the underlying similarity model.

Acknowledgements

Florian Waas provided the image sets and feature data discussed in Section 9.3.1. Wim de Leeuw implemented the solver discussed in Section 9.3.2.

References

[1] S.K. Card, J.D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization*. Morgan Kaufmann Publishers, 1999.

[2] A. del Bimbo. *Visual Information Retrieval*. Morgan Kaufmann Publishers, 1999.

[3] G. di Battista, P. Eades, R.A. Tamassia, and J.G. Tollis. *Graph Drawing*. Prentice Hall, 1999.

[4] R.J. Hendley, N.S. Drew, A.M. Wood, and R. Beale. Narcissus: Visualizing information. In S.K. Card, J.D. Mackinlay, and B. Shneiderman, editors, *Readings in Information Visualization*, pages 503–511. Morgan Kaufmann Publishers, 1999.

[5] M.H. Gross, T.C. Springer, and J. Finger. Visualizing information on a sphere. In *Proceedings Symposium on Information Visualization*, pages 11–16. IEEE Computer Science Press, 1997.

[6] T.F. Cox and M.A.A. Cox. *Multidimensional Scaling*. Chapman & Hall, London, 1994.

[7] J. Ashley, M. Flickner, J.L. Hafner, D. Lee, W. Niblack, and D. Petkovic. The Query By Image Content (QBIC) System. In *ACM SIGMOD Conference on Management of Data*, page 475, 1995.

[8] Y. Rubner, C. Tomasi, and L.J. Guibas. A Metric for Distributions with Applications to Image Databases. In *IEEE International Conference on Computer Vision*, pages 59–66, Bombay, India, January 1998.

[9] V. Ogle and M. Stonebraker. Chabot: Retrieval From a Relational Database of Images. *IEEE Computer*, 28(9):40–48, September 1995.

[10] A. Pentland, R. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, June 1996.

[11] J. Huang, S. R. Kumar, M. Mitra, W.-J. Zhu, and R. Zabih. Image Indexing Using Color Correlograms. In *IEEE Computer Vision and Pattern Recognition*, pages 762–768, Puerto Rico, 1997.

[12] B.S. Manjunath and W.Y. Ma. Texture features for browsing and retrieval of large image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, August 1996.

[13] R. van Liere and W. de Leeuw. Graphsplatting: visualizing graphs as continuous fields. Submitted for publication.

[14] Corel Stock Photos. http://www.corel.com/products/clipartandphotos/photos/, 1999.

[15] S. Santini and R. Jain. Similarity is a Geometer. *Multimedia Tools and Applications*, 5(3):377–306, November 1997.

Conclusions

The methods presented in this thesis provide a set of tools and open novel possibilities for interactive visualization. Tools have been provided for distributed data management in a computational steering environment, the development of portable virtual reality applications, and a new method for representing graphs as continuous fields. Two issues have been addressed in this thesis. These were:

- How can the application of interactive visualization result in a more effective modeling, computation and analysis cycle?
- How can the development costs of interactive visualization applications be lowered?

Interaction has been the driving force behind these visualization methods. The assumption is that direct control can improve the scientist's insight in the relations between input parameters and output results. This assumption has been studied by applying the interactive tools to three applications. For each application it was shown that a scientist could pose questions which would be inconceivable in a non-interactive computation:

- The smog application presented in Chapter 4 was done with a scientist that developed the numerical techniques for solving the underlying transport equations in a four layer model. In a non-interactive environment, the scientist could specify only the boundary conditions and run the computation for a number of time steps. In the interactive environment, the scientist could mutate the emission, meteorological and geographical conditions at any time step. This can not be achieved in a non-interactive environment since it is inconceivable to perceive these scenarios in advance.

  In addition, the interactive environment provided a visual debugger for the development of the numerical techniques. By analyzing the scenario, the developer was able to debug and improve the conditions for the adaptive grids and test these conditions in various scenarios.

- The Fekete application presented in Chapter 7 was done with two mathematicians that modeled and solved the underlying complex problem. In a non-interactive environment, they could specify only the initial conditions and run the computation for a number of time steps. The output could be used only to

analyze the convergence properties of the solution method. In the interactive environment, the mathematicians could analyze the dynamics of Fekete point configurations by monitoring the solution process and analyze the stability of a configuration by directly perturbing it. Moreover, special purpose visualization techniques (snap dragging and moonscapes) allowed global and local energy landscapes to be plotted. Virtual reality was experienced to be a natural interface for the visualization and precise interaction of point configurations.

- The feature space visualization in Chapter 9 was done with a developer of image features. In a non-interactive environment, multidimensional scaling can be used to project the high-dimensional space onto a lower dimensional space. The projection can be used to display a number of images in a way that allows the user to see at glance the commonalities of the image set. In an interactive environment, the developer can interactively scale a feature allowing the effectiveness of a feature in a particular context to be analyzed.

  This application illustrates situations in which the feature developer can do a better job of directing search than any pre-programmed algorithm.

A number of software engineering lessons have been learned when developing interactive visualization tools:

- Both the CSE and PVR are based on event-driven programming models. Event driven models can simplify the programmer's task because these models can decouple the coordination and management of multiple event streams from actual data processing. Programmers need only provide actions associated with an event and the run time system manages the control flow of events.

  The CSE and PVR environments use event driven models in different ways. Events in the CSE denote data mutations in the blackboard. Satellites can only write to a variable, and are not aware of other satellites that are subscribed to the variable mutation. The blackboard is responsible to dispatch the event to satellites. Events in PVR denote a state change in a process. Processes can only send an event to the PVR bus, and are not aware of other processes that are subscribed to the event. The bus is responsible to dispatch the event to processes.

- Tools should allow for rapid integration of existing scientific codes. Rapid integration is crucial particularly in the early development stages of the simulation and when user wishes to experiment with various aspects of the simulation. For example, when uncertainties exist if the problem is modeled detailed enough, when it is not yet known if the correct numerical methods were chosen, and when many parameters have still to be tuned.

  Integration has been addressed in the CSE by providing application programmer interfaces that are easy to use and flexible mappings between the data used and stored in the simulation and the data used and stored by the visualization

tools. Integration has been addressed in PVR in two ways. First, by providing the application programmer with simple mappings between the application workspace, standard workspace and device workspaces. Second, by providing an abstract device model which shields specific low-level device specific details.

Color Plates

Figure 2.6: Radon concentrations over the Indian ocean. Small colored circles show measured sites data.

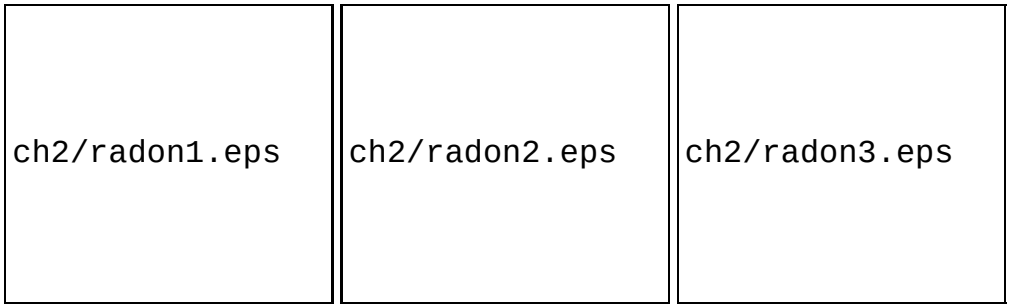


Figure 2.7: Comparative time sequence of converging point set.

See Section 2.4 for explanation.

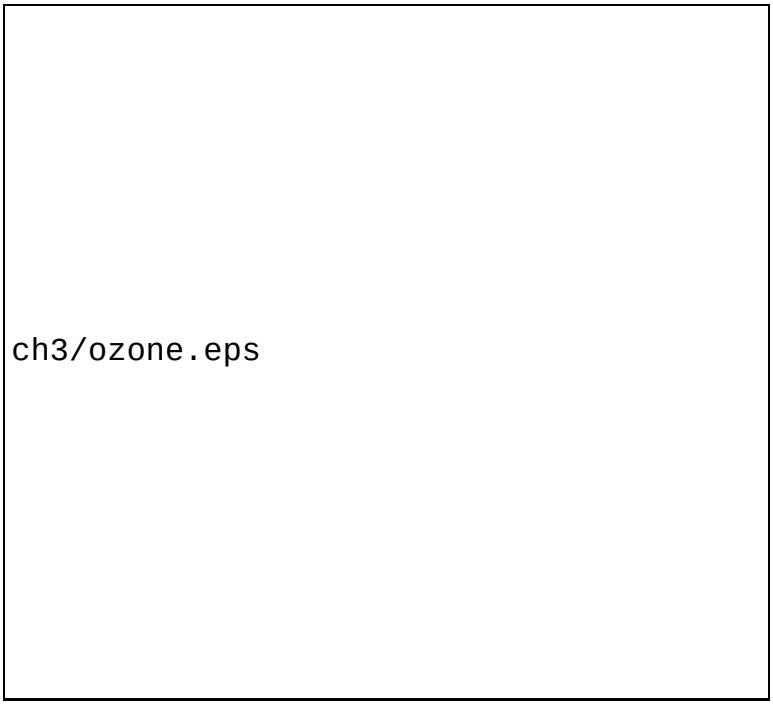


Figure 4.2: Four layers of ozone concentration over Europe.

Figure 4.3: Ozone with grid (surface layer) and wind fields (upper layer).

See Section 4.3 for explanation.

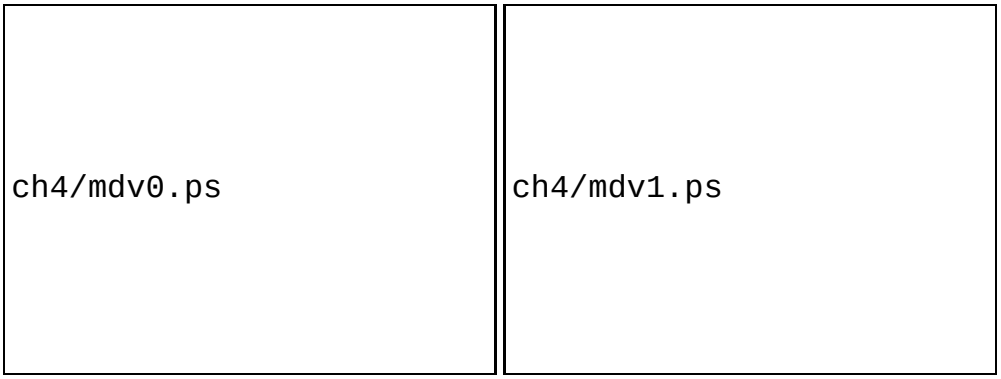


Figure 5.4: Molecular dynamics of a mixture of molten lithium and potassium carbonates. Left uses a convex hull to visualize a coordination of a selected Li^+ ion. Right uses a measuring tape to monitor distances between two selected ions.

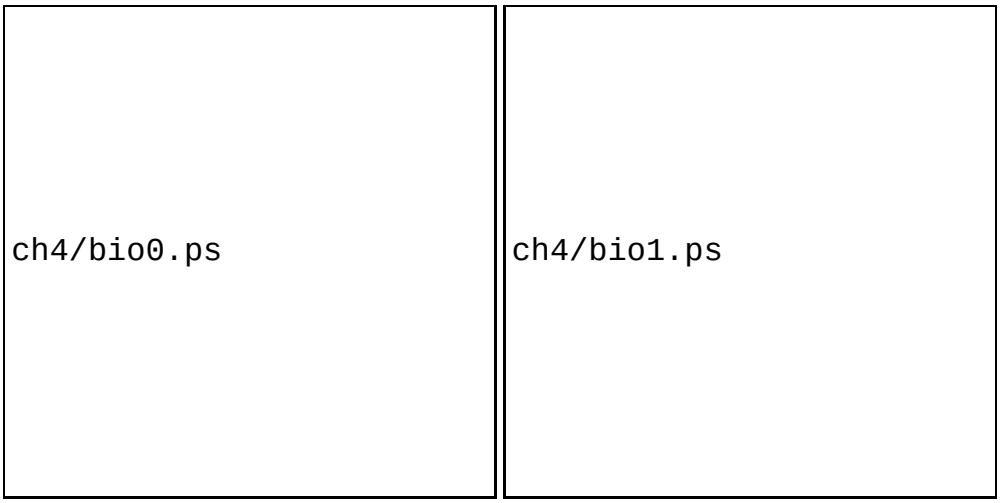


Figure 5.6: Microtubules in a spindle during nuclear division. Left uses a blue-yellow color map with high transparency values, right uses a red-yellow color map with lower transparency values. Head tracked stereo display allow researchers to effectively perceive the complex spatial relationships of the microtubules.

See Section 5.4.1 and 5.4.2 for explanation.

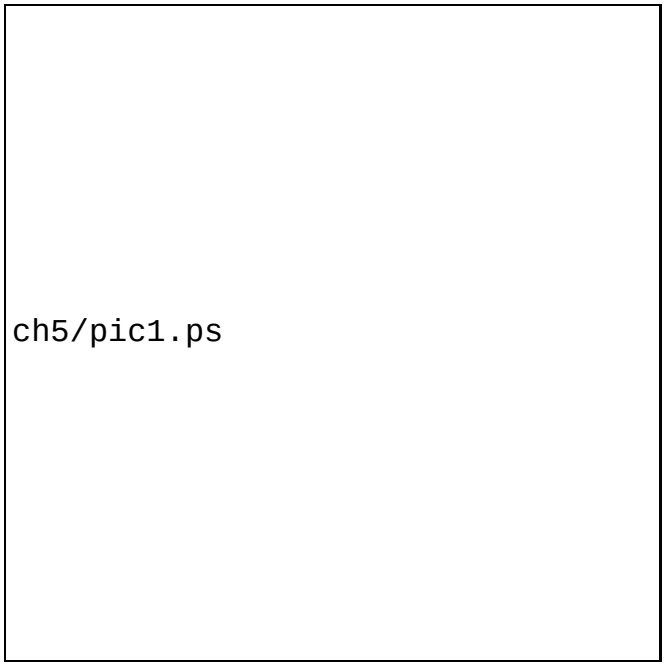


Figure 7.1: A 36 point configuration with convex hull and trajectories.

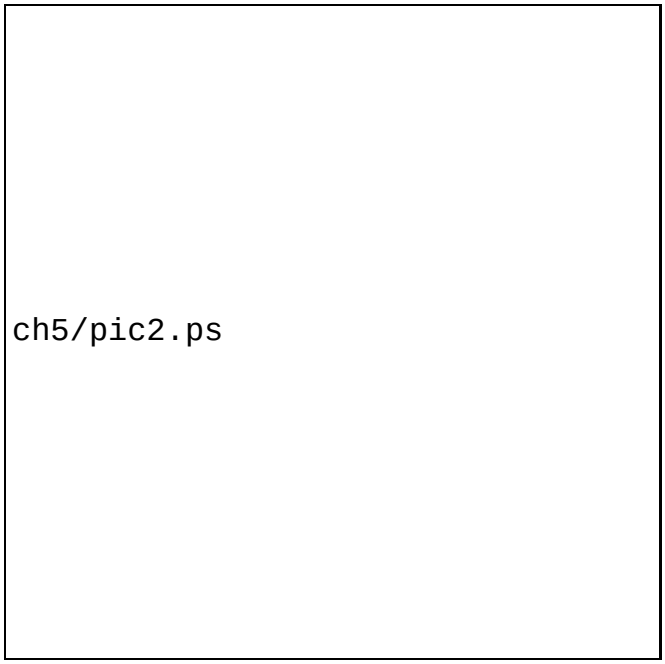


Figure 7.2: Local moonscape around a point. The moonscape visualizes the stability around the point.

See Section 7.4.1 for explanation.

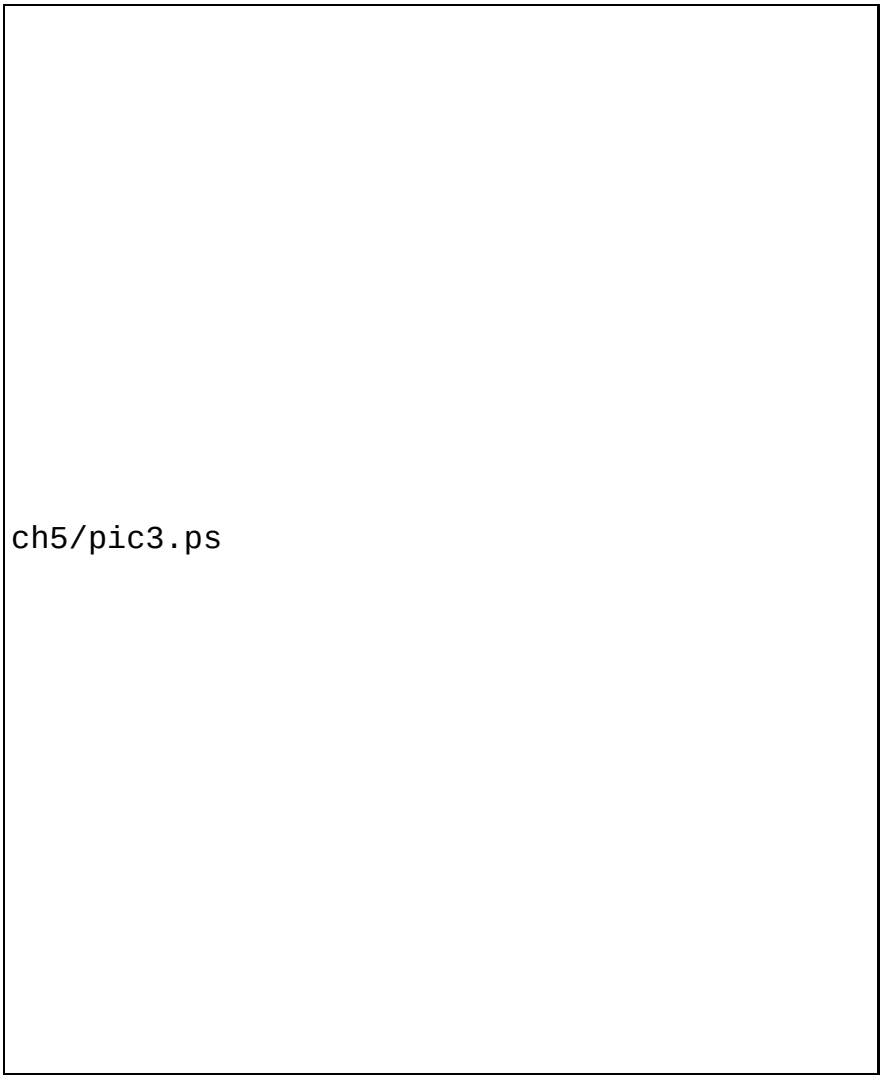


Figure 7.3: Snap convergence during perturbation of a configuration. The small red configurations on the upper left are a list of converged configurations. The yellow configuration is the configuration to which the current configuration will converge. Small yellow and red crosses are drawn on the path taken by the dragged point to indicate the stable configuration at the location of the cross.

See Section 7.4.1 for explanation.

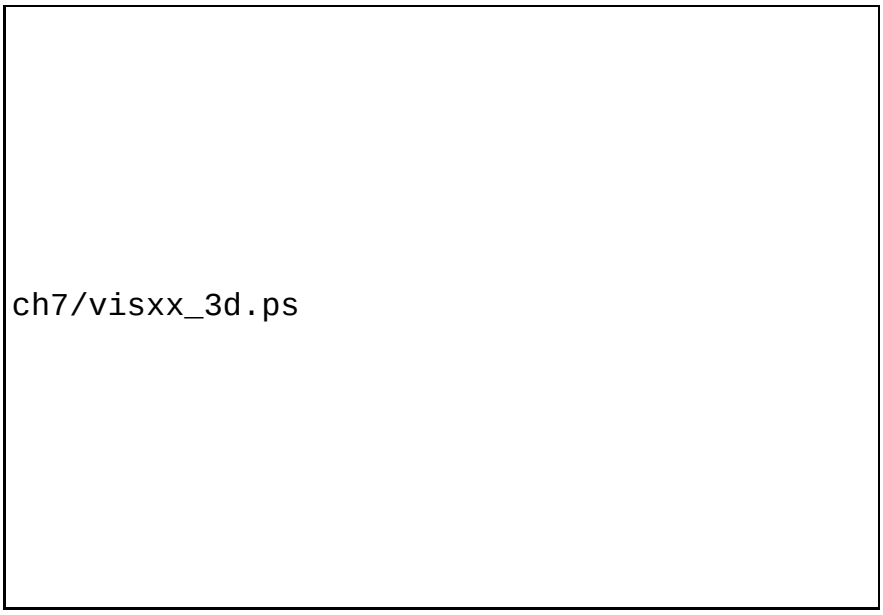


Figure 8.6: Height map representation of the graph showing the main topic clusters.

Figure 8.7: Interacting with the citation index. The influence of a group of papers is drawn with yellow (incoming) and blue (outgoing) references. Papers in the region selected by the highlighted contour on the right are shown as discs.

See Section 8.4.1 and 8.5.1 for explanation.

Figure 8.8: Overview of the call graph and associated splat field. The relative number of calls of a function from its calling function is mapped to the spring stiffness.

Figure 8.9: Dynamic profile of the Matrox G400 driver. The height of the splats is scaled with the amount of time spent in the corresponding function.

See Section 8.5.2 for explanation.

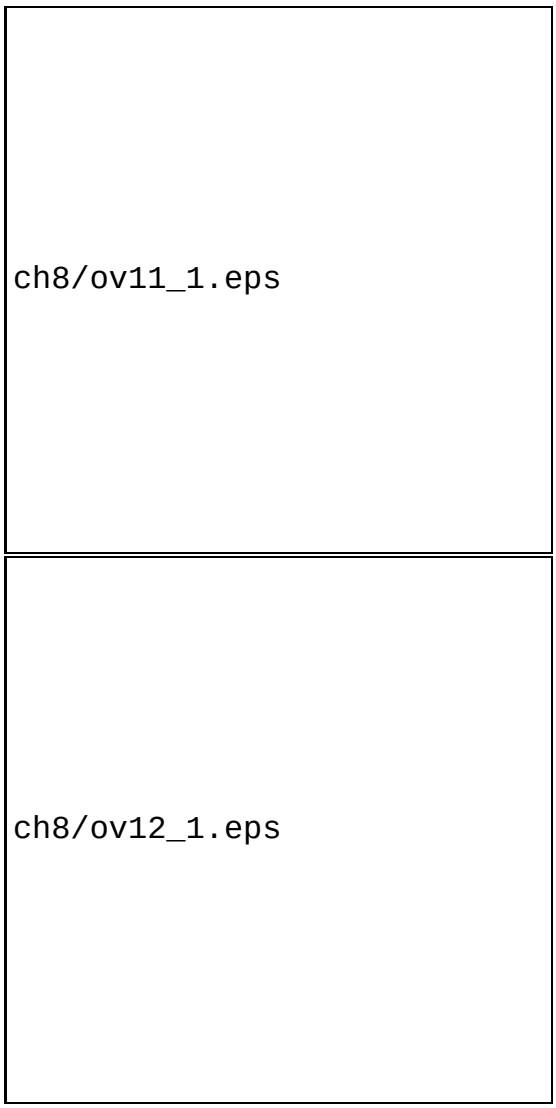


Figure 9.2: Two views of a graph arrangement for the test set. The upper panel shows graph view with vertices and edges. The lower panel shows the splat field. Some vertices are annotated with a thumbnail image.

See Section 9.4 for explanation.

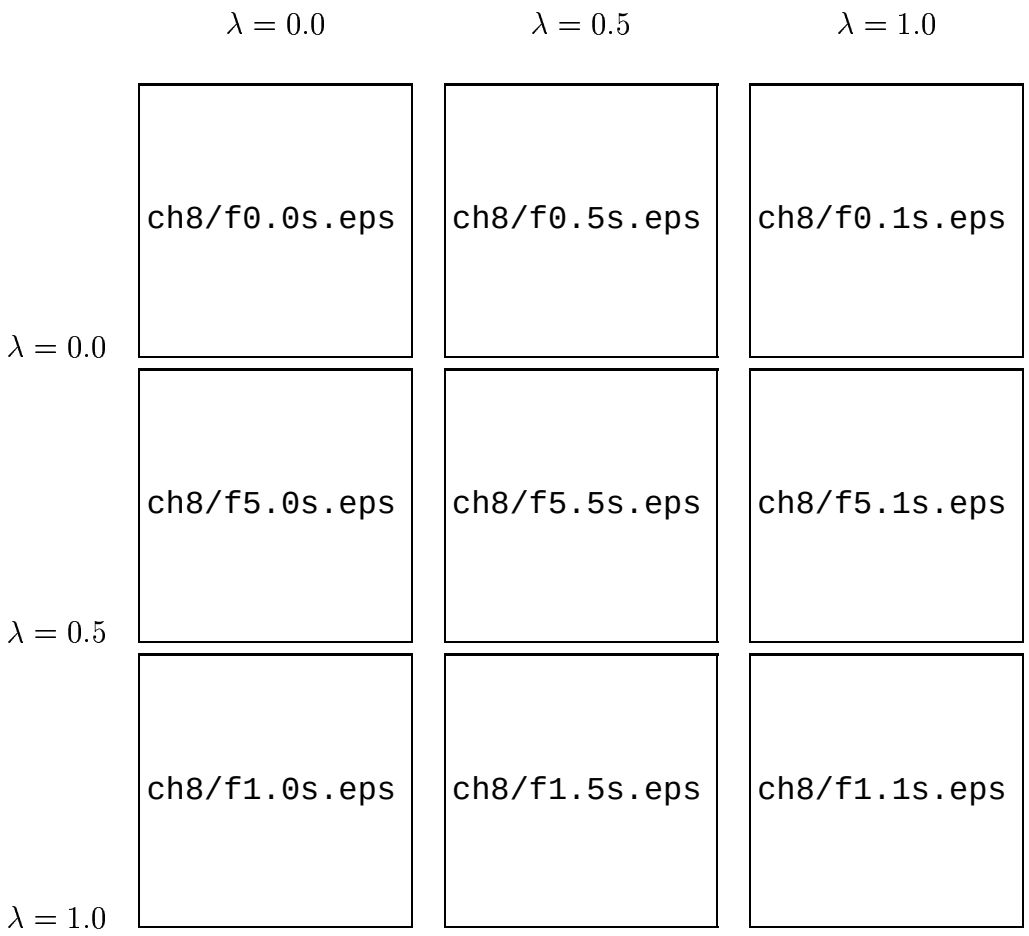
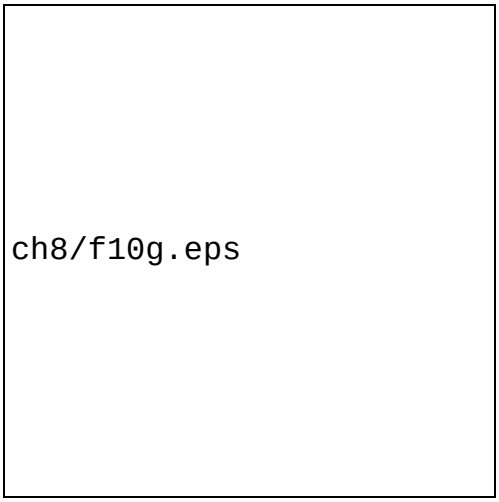


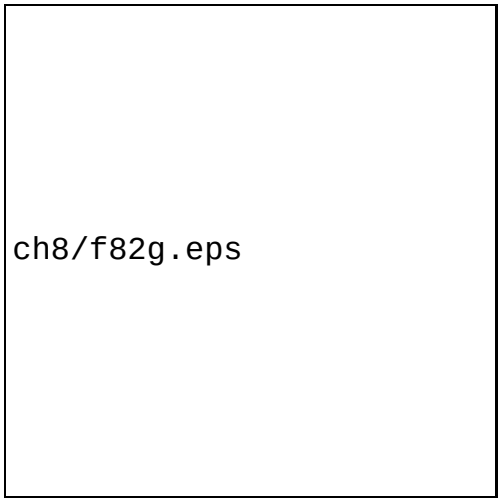
Figure 9.3: Nine splat fields with different scaling factors. Rows have increasing λ values for the hue histogram feature vector. Columns have increasing λ values for the gabor feature vector.

See Section 9.4 for explanation.

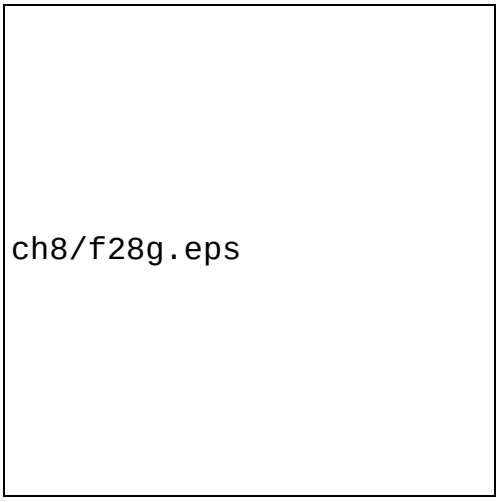
$\lambda_{hue} = 0.0, \lambda_{gabor} = 1.0$



$\lambda_{hue} = 0.8, \lambda_{gabor} = 0.2$



$\lambda_{hue} = 0.2, \lambda_{gabor} = 0.8$



$\lambda_{hue} = 1.0, \lambda_{gabor} = 0.0$

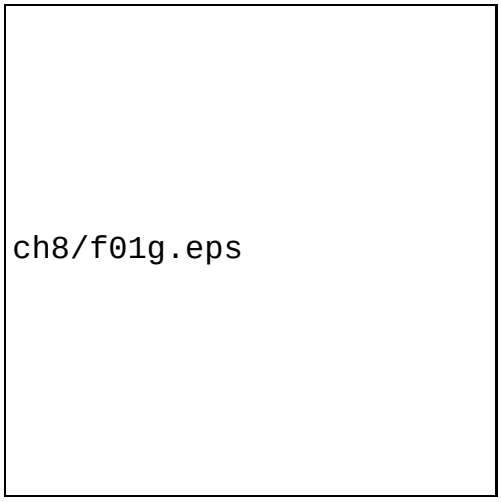


Figure 9.4: Four panels showing the graph with different scaling factors for the hue histogram and gabor feature vectors. Ten vertices are annotated with a thumbnail image.

See Section 9.4 for explanation.

Summary

The purpose of visualization is to enhance existing scientific methods by increasing a scientist's ability to understand the results of computations. Interactive visualization allows the scientist to control the computational processes generating the data for the visualization. Accordingly, the scientist is able to change properties of a mathematical model that is being computed with immediate and direct effect. The assumption is that interactive visualization can improve the scientist's insight in the relations between input parameters and output results.

Two topics are addressed in this thesis: how does the application of interactive visualization result in a more effective modeling, computation and analysis cycle, and secondly, how can the development costs of interactive visualization applications be lowered. These topics are addressed in the context of *computational steering*, *virtual reality* and *information visualization*.

Computational steering enables scientists to have direct control over the parameters of a simulation. Data management is an important aspect of steering large scale simulations. The design of a distributed data manager is described. The distributed data manager is applied to steer smog simulations, allowing a scientist to gain insight into alternative scenarios by changing various meteorological, geographical and emission conditions.

A virtual reality application is a computer synthesized virtual world with which a scientist can directly interact. However, since virtual reality involves new interface styles, a great deal of implementation effort is required to develop effective applications. The design of an architecture for portable virtual reality applications is described. This architecture is applied to the analysis of a classical problem in mathematics: that of approximating globally optimal Fekete point configurations. It is shown that a highly interactive virtual environment can provide valuable insight into the symmetry and stability of the point configurations.

Information visualization is a form of visualization used for the presentation of abstract data. In order to study issues related to the visualization of large graphs, the GraphSplatting technique was developed. The graph is displayed as a continuous field which preserves the main structure conveyed by the graph. GraphSplatting is applied to the interactive visualization of multidimensional feature spaces. By interactively changing the influence of a particular feature, a user can gain insight into how the feature behaves in relation to other features.

Samenvatting

Het doel van visualisatie is om bestaande wetenschappelijke methodes te verbeteren door onderzoekers een beter inzicht te geven in de resultaten die door deze methodes geproduceerd worden. Interactieve visualisatie stelt de onderzoeker in staat om parameters te controleren van de processen die de te visualiseren data genereren. Hierdoor kan de onderzoeker direct ingrijpen in de eigenschappen van een wiskundig model. De aanname is dat interactieve visualisatie het inzicht van de onderzoeker in de relaties tussen invoerparameters en uitvoerresultaten kan verhogen.

In dit proefschrift worden twee onderwerpen besproken: hoe draagt interactieve visualisatie bij aan een meer effectieve cyclus van modellering, berekening en analyse en, ten tweede, hoe kunnen de ontwikkelingskosten verkleind worden van toepassingen die interactieve visualisatie gebruiken. We bespreken deze onderwerpen in de context van *computationele sturing*, *virtuele werkelijkheid* en *informatievisualisatie*.

Computationele sturing stelt een onderzoeker in staat om directe controle uit te oefenen op de parameters van een simulatie. Databeheer is een belangrijk aspect bij het uitvoeren en sturen van grootschalige simulaties. We bespreken ontwerp en implementatie van een gedistribueerde databeheerder. Deze databeheerder wordt onder andere toegepast om smogsimulaties te sturen. Door het veranderen van de condities die te maken met meteorologie, geografie of emissies kan een onderzoeker inzicht krijgen in uiteenlopende scenarios.

Een op virtuele werkelijkheid (“virtual reality”) gebaseerde toepassing is een door de computer gegenereerde virtuele wereld waarmee de onderzoeker kan interacteren. Doordat deze toepassingen geheel nieuwe vormen van gebruikersinteractie vereisen, vergt hun implementatie tot nu toe een grote inspanning. We bespreken ontwerp en implementatie van een architectuur voor het bouwen van overdraagbare virtual reality toepassingen. We passen deze architectuur onder andere toe bij de analyse van een klassiek probleem in de wiskunde: de benadering van globaal optimale Fekete puntconfiguraties. We laten zien dat een interactieve virtuele omgeving het inzicht kan vergroten in de symmetrie en de stabiliteit van de puntconfiguraties.

Informatievisualisatie is een vorm van visualisatie die gebruikt wordt voor de presentatie van abstracte data. Om het visualiseren van zeer grote grafen te onderzoeken hebben we de techniek van “GraphSplatting” ontwikkeld: een graaf wordt voorgesteld als een continu veld waarin de struktuur van de graaf behouden is. We hebben GraphSplatting onder andere toegepast bij de visualisatie van kenmerkruimtes zoals in ge-

bruik bij het classificeren van multi-mediale informatie. Door de invloed van een bepaald kenmerk te veranderen kan een gebruiker inzicht krijgen in het gedrag van dat kenmerk ten opzichte van andere kenmerken.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-1

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-2

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-5

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-6

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-7

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-9

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division

of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06