Diss. ETH No. 14585 2002

IVORY - An Object-Oriented Framework for Physics-Based Information Visualization

A dissertation submitted to the
Swiss Federal Institute of Technology (ETH) Zurich

for the degree of Doctor of Technical Sciences

presented by

Thomas Carl Sprenger
Dipl. Informatik-Ingenieur ETH Zürich

born July 5, 1970 in Basel

accepted on the recommendation of
Prof. M. H. Gross, Examiner
Prof. H.-J. Schek, Co-Examiner

To my parents Silvia and Robert

“Jeder weiss, dass Wahrscheinlichkeitsrechnung und Statistik dasselbe sind, und dass Statistik weiter nichts ist als Korrelation. Nun ist aber die Korrelation nichts anderes als der Cosinus eines Winkels. Also ist das Ganze trivial”

(Emil Artin, 1947)

ACKNOWLEDGMENTS

At this place, I would like to take the opportunity to thank all the colleagues and friends who supported me in realizing this work. Without their help this project would never have succeeded.

Especially, I would like to express my gratitude my theses supervisor Prof. Markus Gross for his spontaneous commitment to this research project. I want to thank Markus for his guidance and his uncompromising support for the project, as well as for the confidence and freedom he gave me in pursuing my research interests. With his remarkable combination of expertise and social competence, Markus established an excellent working environment in his research group, in which not only the professional but also personal development takes place.

I am very grateful to Prof. Hans-Jörg Schek to be my co-examiner, and that he offered me the chance to use the data from one of his research projects for an additional illustrative example of use.

I thank all my fellow colleagues at the Computer Graphics Group who have helped me along the way, especially Lars Lippert, Martin Roth, Oliver Staadt, Reto Lütolf, Rolf Koch, Daniel Bielser, Andreas Hubeli, Matthias Zwicker, Ronny Peikert, and all the others not explicitly named here. Working with them has always been a great pleasure for me.

Many thanks also to all the former students who contributed in parts to the implementation of this work: Andreas Eggenberger, Michael Kaufmann, Dominik auf der Maur, Felix Jost, Jörg Finger, Daniel Bielser, Tobias Strasser, Patrick Pfister, Stavros Antifakos, Thomas Ammann, Reto Schorno and Renato Brunella.

Many thanks to my brother Christoph, Stefan Wengi, Harry Walther and Bruno Kaiser for careful proof-reading of parts of this thesis.

Further on, my special thanks goes to all the members of our cooperation partner the former IT Camp of UBS AG, Basel. Especially, I appreciate the critical but always constructive feedback from Martin Huber. Also, many thanks to Ming Hao of HP Laboratories in Palo Alto. She offered me an internship at HP Labs during 1998, which can - from today's point of view - not only be denoted as a worthwhile, but also as a personally trend-setting experience.

I am much obliged to all my friends from beach volleyball, particularly to my team partner Andi Lagler, giving me the chance to take a break once in a while and clear my mind between the long hours of the final writing. Special thanks and a warm hug go to Rea for her virtually infinite patience with me and my work.

Last but not least, I want to thank my family for generously providing me with the greatest support all the way through the last years. Without their help, I could not have achieved the goal to finish this thesis.

CONTENTS

Abstract

Zusammenfassung

1 Introduction . . . 1
1.1 Information Visualization . . . 2
1.1.1 Basic Principles . . . 3
1.1.2 Origins . . . 3
1.1.3 Visualization Techniques . . . 4
1.1.4 Frameworks and Systems . . . 7
1.2 Contribution . . . 8
1.3 Chapter Outline . . . 9

2 Operator Framework Architecture . . . 11
2.1 Functionality . . . 11
2.2 Fundamental Concepts . . . 12
2.3 Abstraction Model . . . 15
2.4 Classification of Operators . . . 16
2.5 Implications for Framework Designs . . . 18

3 Metric Spaces . . . 19
3.1 Terminology and Basic Definitions . . . 19
3.2 Mappings . . . 20
3.3 Constructions . . . 21
3.4 Quantifying Similarity in Information Space . . . 23
3.4.1 Booleans . . . 25
3.4.2 Scalars . . . 26
3.4.3 N-dimensional Scalar Vectors . . . 26
3.4.4 Dynamic Numerical Data Series . . . 28
3.4.5 Non-Numerical Entities . . . 30

4 Physics-based Models . . . 35
4.1 Basic Mass-Spring Systems . . . 36
4.2 Advanced Force Models . . . 37
4.3 Differential Equations . . . 40
4.4 Initial Particle Positioning . . . 42
4.4.1 Determination of Initial Positions . . . 42
4.4.2 Assign particles to initial positions . . . 44
4.5 Force Calculation Schemes . . . 47
4.5.1 The Particle-Particle Method (PP) . . . 48
4.5.2 The Particle-Mesh Method (PM) . . . 50
4.5.3 The Top Down Tree Code Method (TDTC) . . . 52
4.5.4 The Fast Multipole Method (FMM) . . . 54
4.6 Numerical Time Integration Methods . . . 55
4.6.1 Runge-Kutta Methods . . . 57
4.6.2 Leap Frog Method . . . 61
4.7 Stiff Problems and numerical Stability . . . 62
4.7.1 Stability Analysis for Runge-Kutta Methods . . . 63
4.8 Implications for Information Visualization . . . 65

5 Graph Visualization 67
5.1 Graph Theory 67
5.1.1 Basic Definitions 68
5.1.2 Weighted Graphs 69
5.2 Graph Layout 70
5.2.1 Conventions 70
5.2.2 Key Issues 71
5.2.3 Force-Directed Layout Methods 72
5.3 Graph Rendering 73
5.4 Implementation Issues 75
5.4.1 Graph Methods 75
5.4.2 Data Structures 76
5.4.3 Complexity Considerations 79

6 The IVORY Framework 81
6.1 Requirements 81
6.2 The Fundamental Visualization COncept 82
6.3 Implementation of the Abstraction Model 83
6.4 Software Architectural Issues 86
6.4.1 Framework Kernel 87
6.4.2 Plug-in Mechanism 88
6.5 Framework Configuration 91
6.6 Implementation Issues in Java 92

7 Clustering 95
7.1 Principles 95
7.2 Analytical Clustering Algorithms 96
7.2.1 Partitioning Methods 97
7.2.2 Hierarchical Methods 99
7.3 Visual Clustering Methods 101
7.3.1 Ellipsoid Clustering 102
7.3.2 BLOB Clustering 106
7.3.3 H-BLOB Clustering 108
7.4 Implementation Issues 115

8 Information Visualization Modeling Language (IVML) 117
8.1 Scope 117
8.2 Requirements 118
8.3 Design Concepts 119
8.3.1 Deviations from the VRML standard 119
8.4 Specification 120
8.4.1 Language Basics 120
8.4.2 Coordinate System 125
8.4.3 Fields 125
8.4.4 Field Containers 126
8.5 Implementation Issues 133
8.5.1 Interpreted script language 133
8.5.2 Extensibility through Java's Reflection API 133
8.5.3 Dirty-Tag Propagation 133
8.5.4 Constant Folding 133
8.6 Example 134

9 Results 139
- 9.1 Usability Study 139
  - *9.1.1 Objectives and Limitations 139*
  - *9.1.2 The UBS Usability Lab 140*
  - *9.1.3 Scenario 140*
  - *9.1.4 Hypothesis 140*
  - *9.1.5 Observations and Results 140*
  - *9.1.6 Conclusions 141*
- 9.2 Applications and Coverage Examples 141
  - *9.2.1 Prediction of Long Term Interest Rates 142*
  - *9.2.2 Shopping-Basket Analysis 143*
  - *9.2.3 Document Retrieval Visualization 146*
  - *9.2.4 Image Retrieval Example 147*

10 Conclusions and Outlook 151

A Package Layout 153

B The Grammar of IVML 161

C Usability Test Scenario 165

D References 173

E Color Plates 183

F Copyrights 187

G Curriculum 189

FIGURES

Fig. 1.1: Classification of established visualization frameworks 7
Fig. 2.1: Chuah and Roth's Basic Visualization Interaction Taxonomy. 13
Fig. 2.2: Adapted standard visualization pipeline. 5
Fig. 2.3: Summarizing list of recognized operator classes 17
Fig. 2.4: Schematic view on the OFA concepts 17
Fig. 3.1: Similarity Mapping Scheme A 24
Fig. 3.2: Similarity Mapping Scheme B 25
Fig. 4.1: A basic mass-spring system 38
Fig. 4.2: Particle with spring and friction forces 39
Fig. 4.3: Particle with spring, friction, electrical and gravitational forces. 42
Fig. 4.4: A simple mass-spring system converging to its equilibrium state 43
Fig. 4.5: Initialization of particles on a virtual sphere. 45
Fig. 4.6: Poisson disc distribution 46
Fig. 4.7: Unfavourable initialization with crossing connections 46
Fig. 4.8: Assignment order of breadth-first-traversal assignment 47
Fig. 4.9: Assignment order of depth-first-traversal assignment 48
Fig. 4.10: Flowchart of particle-particle code (PP) 51
Fig. 4.11: Flowchart of particle-mesh code (PM) 53
Fig. 4.12: Illustartion of the Euler's method 59
Fig. 4.13: Illustrating inaccurate and instable behavior for Euler's method. 60
Fig. 4.14: Leap Frog integration scheme. 64
Fig. 4.15: Domains of absolute stability 67
Fig. 5.1: A simple undirected graph 70
Fig. 5.2: Different types of graph renderings 75
Fig. 5.3: Basic structure of an *Edge List*. 79
Fig. 5.4: Basic structure of an *Adjacency List*. 79
Fig. 5.5: Basic structure of an *Adjacency Matrix*. 80
Fig. 5.6: Basic structure of an *Implicit Adjacency Matrix*. 81
Fig. 6.1: Typical result computed by the IVORY framework. 85
Fig. 6.2: IVORY's consequent implementation of the OFA. 85
Fig. 6.3: Schematic overview of the system-components in IVORY 88
Fig. 6.4: Screenshot showing the main GUI elements of the IVORY frontend. . . . 89
Fig. 6.5: IVORY's plug-in hierarchy: The base classes. 91
Fig. 6.6: A example scene containing several cluster object instances 93
Fig. 6.7: User abstraction model and system architecture 94
Fig. 7.1: Clustering of a subset of objects performed with BLOBS 98
Fig. 7.2: Partitioning using C-means method 99
Fig. 7.3: Partitioning using K-means algorithm. 100
Fig. 7.4: Probable object arrangement with 8 objects 101
Fig. 7.5: Generation of chains applying the single linkage method 102
Fig. 7.6: Different techniques to visualize clusters of data objects. 103
Fig. 7.7: Adjustment of the scaling using a transformed point set 106
Fig. 7.8: Illustration of the clustering method: 108
Fig. 7.9: Illustration of implicit surfaces obtained from two field function 109
Fig. 7.10: Clustering of a subset of objects scattered in 3D space 109
Fig. 7.11: Undesired intersection of object and hull. 110

Fig. 7.12: Progressive edge-collapse algorithm 112
Fig. 7.13: Cluster tree with three levels 113
Fig. 7.14: Isolines of a spherical, symmetric and a new ellipsoidal primitive 114
Fig. 7.15: Three objects for which an enclosing tight BLOB surface 116
Fig. 7.16: Isovalue considerations for BLOB surfaces. 116
Fig. 7.17: Small example showing the H-BLOB clustering process. 118
Fig. 8.1: Schematic structure of an IVML script 123
Fig. 8.2: Small scene consisting of 4 objects 125
Fig. 8.3: Illustration of the scene resulting from the script above 126
Fig. 8.4: Example showing the basic structure of a field container. 128
Fig. 8.5: Plug-in hierarchy of generic IVML specific classes 129
Fig. 8.6: Dirty-Tag propagation within an expression tree 136
Fig. 8.7: Example of an expression tree reduced through constant folding 136
Fig. 8.8: Layout computed from IVML script presented below 137
Fig. 9.1: Floor plan of the UBS usability lab. 141
Fig. 9.2: Video snapshots of a typical usability session 142
Fig. 9.3: Two typical IVORY scenes from the usability test 143
Fig. 9.4: Condensing multivariate relationships. 144
Fig. 9.5: Two views showing the influence of economic indicators 144
Fig. 9.6: Clustering the results of econimic example 145
Fig. 9.7: Snapshot of a *VisMine* [HP©] scene. 146
Fig. 9.8: Different views on the results of the shopping basket analysis 147
Fig. 9.9: Document retrieval visualization using H-BLOB clustering 148
Fig. 9.10: Two different images with their corresponding feature vectors 149
Fig. 9.11: Screen shots from the image retrieval example 150
Fig. A.1: Layout of package ivory.backend 156
Fig. A.2: Layout of package ivory.backend.cluster 157
Fig. A.3: Layout of package ivory.backend.layoutsystem 158
Fig. A.4: Layout of packages ivory.ivml.system 159
Fig. A.5: Layout of package ivory.plugin 160
Fig. A.6: Layout of package ivory.ivml 161

TABLES

Table 1.1: Classification of visualization techniques 5
Table 2.1: Categorization of operators 14
Table 3.1: Summary of metric categories 21
Table 3.2: Elements of a typical document description vector 31
Table 4.1: Pseudocode listing breadth-first-traversal position assigment 48
Table 4.2: Pseudocode listing depth-first-traversal position assigment. 49
Table 4.3: List of force calculation methods 50
Table 4.4: Time step loop of particle-particle method (PP). 51
Table 4.5: Time step loop of particle-mesh method (PM). 53
Table 4.6: Time step loop of top down tree-code method (TDTC/BHM) 55
Table 4.7: Time step loop of fast multipole method (FMM). 57
Table 4.8: The Euler's update algorithm 59
Table 4.9: A reduced second-order Runge-Kutta update algorithm 61
Table 4.10: The implicit Euler's update algorithm. 63
Table 4.11: The Leap-Frog update algorithm 63
Table 5.1: List of widely used layout conventions. 72
Table 5.2: List of commonly adopted aesthetic issues 73
Table 5.3: Compilation of visual parameters showing meaningful results. 76
Table 5.4: Complexity considerations of graph data structures 81
Table 7.1: Pseudocode listing of the C-mans algorithm 100
Table 7.2: Pseudocode listing of the ellipsoid scaling adjustment 107
Table 7.3: Pseudocode segment of the volume-based clustering algorithm 110
Table 8.1: List of valid IVML plain fields 128
Table 8.2: Connectivity matrix of the example shown below. 137
Table 9.1: Assignment of feature vector values 149

ABSTRACT

The present thesis investigates a new variant for physics-based information visualization. The fundamental idea bases on a quantification of the similarity of related objects, expressed by the parameters of a mass-spring system. Since the spring stiffnesses correspond to the computed similarity measures, the system converges into an energy minimum, which reveals multidimensional relations and adjacencies in terms of spatial neighborhoods.

As a part of this work we develop a platform-independent framework, called IVORY, for physics-based visualization and analysis of multidimensional data relations. The software design follows strictly the theory of operator frameworks. It is fully implemented in JAVA and its architecture features client-server setup, which allows to run the visualization even on thin clients. In addition, VRML 2.0 exports can be viewed by any VRML enabled web browser. Individual visual metaphors and interaction paradigms are invoked into IVORY via an advanced plug-in mechanism. The configuration of IVORY is accomplished using a specialized high-level script language, the Information Visualization Modeling Language (IVML).

In order to simplify complex setups we suggest visual clustering algorithms for postprocessing. We present a set of three new visual clustering algorithms – called Ellipsoidal, BLOB and H-BLOB clustering – which group and visualize cluster hierarchies at multiple levels-of-detail. We propose that the algorithms are especially suited for the visualization of very large data sets and for visual decision making in information visualization.

The versatility of the framework and its algorithms is demonstrated by means of experimental results. They show the framework's capability to visualize large and complex volumes of different types of abstract data. Furthermore, a usability test including daily-business cases in the areas of our cooperation partner validates IVORY's performance in practice.

This project was sponsored by the Advanced Engineering Center (ACE) of UBS Basel (Switzerland). For that reason the framework is mainly employed in the context of risk analysis, stock prediction, document retrieval and other tasks being important in the context of banking.

ZUSAMMENFASSUNG

Diese Dissertation stellt eine neue Methode der physikalisch-basierten Informationsvisualisierung vor. Die grundsätzliche Idee basiert dabei auf der Quantifizierung der Ähnlichkeit von in Beziehung stehender Objekte, welche wiederum die Parameter eines Masse-Feder-Systems konfigurieren. Weil die Federhärte und die berechneten Ähnlichkeitswerte korrespondieren, konvergiert das System in ein energetisches Minimum, das die inhärent in den Daten enthaltenen Beziehungen und Zusammenhänge als räumliche Nachbarschaften widerspiegelt.

Als Teil dieser Arbeit entstand ebenfalls das plattform-unabhängige Framework IVORY zur physikalisch-basierten Visualisierung multidimensionaler Datenrelationen. Das Softwaredesign orientiert sich streng an der Theorie der Operator-Frameworks. Die Implementation ist vollständig in JAVA realisiert worden und unterstützt Client/Server-Konfigurationen, womit die Visualisierung ebenfalls auf Thin-Clients genutzt werden kann. Zusätzlich können Szenen in VRML 2.0 exportiert werden, womit sie mittels eines Web-Browsers mit installiertem VRML-Plugin betrachtet werden. Individuelle visuelle Metaphern und Interaktionsparadigmen (Information Drill-down) werden modular durch einen sogenannten Plugin-Mechanismus ins System integriert. Die Konfiguration von IVORY erfolgt mit einer eigens dafür entwickelten Sprache, die Information Visualization Modeling Language (IVML).

Zusätzlich erlaubt das Framework komplexe Szenen mittels verschiedener Verfahren visuell und analytisch zu clustern. Wir präsentieren in diesem Kontext drei neue Clusteringansätze: Ellipsoid-, BLOB und H-BLOB-Clustering. Sie berechen mehrstufige Cluster-Hierarchien und stellen diese anschliessend dar. Diese Verfahren eignen sich besonders zur Visualisierung sehr grosser und komplexer Datensätze und für Decision-Making-Systeme.

Die Vielseitigkeit des Frameworks und dessen Algorithmen wird anhand verschiedener experimenteller Beispiele aus diversen Anwendungsgebieten gezeigt. Sie demonstrieren die Leistungsfähigkeit des Frameworks im Bereich grosser, komplexer und abstrakter Datenvolumen verschiedenen Typs. Abschliessend wird mittels einer Anwenderstudie die Leistungsfähigkeit von IVORY in alltäglichen Arbeitsprozessen beurteilt.

Das Projekt wurde in Kooperation mit dem Advanced Engineering Center (ACE) der UBS Basel durchgeführt. Aus diesem Grund liegt das Hauptanwendungsgebiet für das System im Bereich der Finanzdienstleistungen.

1

INTRODUCTION

Visual communication has always been of fundamental importance to mediate information and to understand complex relationships. With the advent of the computer, scientific visualization was born as a discipline [MDB87] and has found many applications in various fields of science and engineering. However, whereas in the past research was mostly focused on the visualization of spatial data sets and metric spaces, the design of new visual metaphors for abstract, complex information spaces has recently emerged as a challenging research topic. Applications are manifold and range from web/network visualization and document retrieval, via customer data to risk and portfolio management in the financial services [CEG96]. They are developed as a crucial support of today's demands on the knowledge crystallization process [RSP+93], which essentially comprises the following steps:

1. Extracting information from a huge amount of arbitrary data
2. Gaining knowledge on the basis of the extracted information
3. Making decisions based on the extracted knowledge and the user's experience

Nowadays, high computing performance, global broadband networks and distributed data bases – including the World Wide Web – provide platforms for new dimensions of retrieval systems. The problem of spotting the relevant information has changed profoundly. Since *data warehouses* have been established as a well-prove technology gathering, storing and accessing the data no longer denotes a real challenge. Today, many of our daily activities, such as shopping and using a credit card, making phone calls or simply surfing the internet are routinely registered in some way or other. As a result, many companies hoard a vast quantity of customer data out of which they expect valuable information about the clients and their specific behavior which again may offer the potential for a competitive advantage. However, extracting useful information inherently contained in this tremendous amount data is a challenge of its own. This problem is known as *data mining*.

For most of the previously established data analysis methods the abundance of data is a serious problem. Unable to view the data as a whole, they suffer from a strongly partitioned view whereas each fragment misleadingly seems to contain *noise* only preventing the extraction of the desired information. A lot of research effort went into developing more advanced data mining methods to automate data filtering processes and information extraction. As a result, sophisticated statistical and heuristic approaches have been designed to find the implicit but potentially useful information buried in large data volumes.

These approaches, however, suffer from two substantial shortcomings. First, they operate in batch mode, which inherently prevents user interactions during the analysis process. Since the input data may vary significantly, the correct parametrization of the involved algorithms is virtually impossible. As a consequence, several runs are required to finally obtain the desired result – the possibility of interaction would doubtlessly alleviate this problem. Secondly, results are presented in a textual form or simple graphics only. Although a lot of the analysts are used to working with texts and tables, this form of representation is rather unsuitable for today's data volumes. On the one hand, only a limited number of result items can be displayed without overloading the presentation. On the other hand, text or even simple charts are unable to reflect the complexity of today's information. Thus, potentially important information could remain undiscovered.

As a consequence, the visualization and computer graphics communities have been challenged to develop advanced methods and tools for understanding, navigating through and interactively analyzing abstract information spaces. To achieve this goal, there is not only the problem of how to represent object attributes visually to be solved, but also the more fundamental one of effectively mapping abstract data features to visual properties– the research field of *information visualization* was born.

1.1 INFORMATION VISUALIZATION

Information visualization describes the process of transforming information into a visual representation enabling the user to interactively explore large and complex data spaces at a more abstract level. Its main goal is to provide the viewer with a qualitative understanding of the information content, and thus amplify cognition. It efficiently supports the user in the task of *knowledge crystallization* described previously in this chapter. To this end, information visualization takes heavily advantage of the enormous storage capabilities, the computational power and the high performance graphics subsystems of today's computers. According to [CMS99] information visualization can be defined as follows:

Information Visualization

The use of computer-supported, interactive, visual representation
of abstract data to amplify cognition.

In contrast to the data used in scientific visualization, the one applied in the context of information visualization is usually non-spatial and abstract, which implies that no obvious spatial mapping exists.Thus, defining a meaningful projection from data space into the physical space is one of the most crucial tasks in the visualization of abstract information.

1.1.1 Basic Principles

Since effective data mining depends on an expert's knowledge – and in contrast to the previously mentioned data mining methods – the theory of information visualization integrates the user with its abilities as an important element into the analysis process. Thus, such systems need to take human cognitive, perceptual and intuitive capabilities into account. Graphical representations, using appropriate visual paradigms, do not only provide an efficient way of *information condensing*, but also make consequent use of powerful *human-computer interfaces*. The information can be ascertained by the user in a multi-parallel manner, while it is examined concurrently by the strongest available pattern recognition processor–the human visual system. Thus, presenting the data in a visual manner may reduce the amount of extra cognitive work.

In order to support the user in the dynamic process of information exploration, which may involve detection, measurement and comparison, data specific interaction methaphors provide the possibility for direct data manipulations. They promise to enhance our understanding of complex multidimensional data as well as assist us in navigating and exploiting today's information collections. By including interaction into the process of knowledge crystallization will enhance to an interactive analysis cycle.

1.1.2 Origins

This subsection presents the main parts of the historical evolution of this admittedly rather young research field. According to [CMS99] the first time use of the term *information visualization* was in [RCM89].

Playfair (1786) was one of the earliest to use abstract shapes, such as lines in order to represent data visually. At this time the development of the classical data plotting methods were started. In his work, Tufte [Tuf83] mentions the famous graphical picture of Napoleon's terrible defeat in Russia as one of the earliest multi-dimensional illustrations. It shows a combination of data map and time series, documenting Napoleon's disastrous losses in 1812. The graphic was drawn in 1861 by the French engineer Charles Joseph Minar (1781-1870).

At a later date in 1967, the French cartographer Bertin published his theory, which identifies the basic elements of diagrams and mentions a framework for their design. Other pioneering work was done by E. R. Tufte, a political scientist from Yale. He focuses on the visualization of data with inherent geometrical semantics. In 1983, Tufte published a theory that emphasized maximizing the density of useful information, which includes layout rules, the mapping of attributes and governs color compositions.

In 1977 when statistical experiments were designed as well organized and defined processes and the suggestion of exploratory analysis seemed like heresy, John Tukey a long-term Bell Labs and Princeton companion, came up with the idea of introducing interaction to experiments. Thereby, the intention was to dynamically find patterns one has not thought of before in the data. His work *Exploratory Data Analysis* initiated and established the use of interactive graphics to give rapid statistical insight into data.

At a time when the scientific visualization community was already able to present mainstream rendering techniques for data types with an underlying physical model – such as flow, voxel or geographical data – the visual representations for arbitrary multidimensional data just began to emerge. A book of particular interest in this context, is the one titled *Dynamic Graphics for Statistics* written by Cleveland and McGill back in 1988. Inselberg's parallel coordinates [ID90] and Mihalisin's technique of cycling through variables at different rates [MTS91] were other important contributions.

In the late eighties the computer graphics and artificial intelligence communities investigated in automatic design of visual data presentation. The visual quality of the results was of little interest compared to the aspects of automating the graphical representation of the data. Not until around 1990, when the development of graphics hardware took a step forward and got generally available at affordable prices, a new generation of user interfaces emerged. They focused on user interaction with large and high-dimensional data sets, such as telecommunication traffic or document collections.

After these initial steps in the field of information visualization a large variety of modified and new approaches followed. The perspective wall [MRC91], cone trees [RMC91] and their hyperbolic projections [MHC+96] are only the most prominent examples.

More expatiated surveys of contemporary information visualization methods can be found in [CEG96, Kei97 or CMS99].

1.1.3 Visualization Techniques

This section provides an overall view of a variety of information visualization techniques which find their application in many systems. These range from the familiar line plots, pie charts or histograms to the technique of projecting high-dimensional abstract data spaces into a visual representation rendered in a physical space. Since the first-mentioned methods are limited to a relatively small amount of not too complex data, they will be discussed no further.

All the techniques presented below take data-specific properties into account in order to produce a meaningful mapping to objects within the visualization. [Wis+95, Cha96] for instance, visualized text documents and clusters as galaxies and themescapes, whereas [CK95] proposed cone trees which specifically address hierarchical organization. Another promising method is [BF95] or [Woo95], who essentially used self-organizing schemes and neural networks to arrange information objects of the WWW. In a more general understanding, multidimensional visualization problems have been stressed in [YR91 and CK95]. Here, mathematical projection algorithms were introduced to map data into subspaces while preserving their most important features. Interestingly, many current methods use physically-based paradigms, such as [Ben96] or [HD95], where information units are taken as nodes of some generalized mass spring system revealing the structure of relations upon relaxation. These types of multidimensional visualization methods have been studied extensively in graph theory, and efficient algorithms had been introduced for fast graph relaxation, such as [FLM94, BF95].

According to the main techniques they employ these methods can be categorized roughly into the classes [Kei97b] shown in Table 1.1.

Table 1.1 Classification of visualization techniques

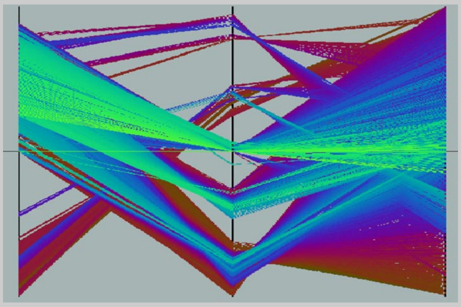
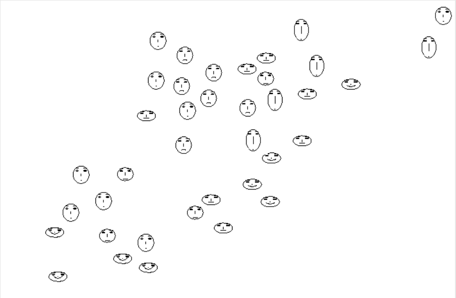
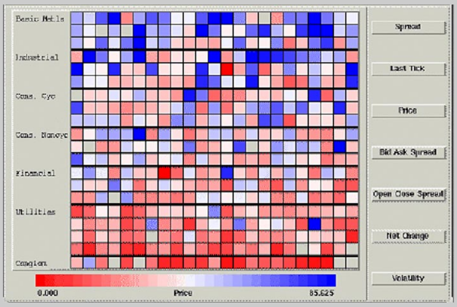
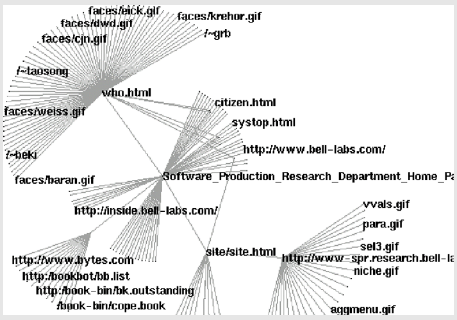
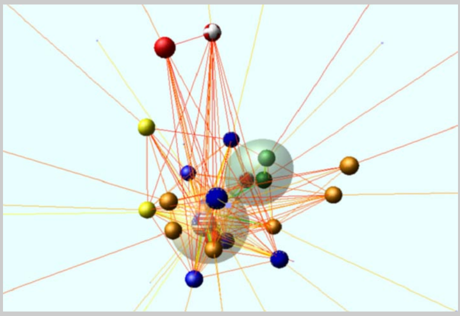
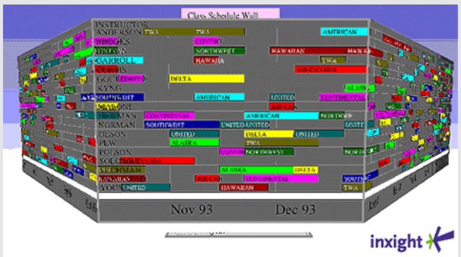
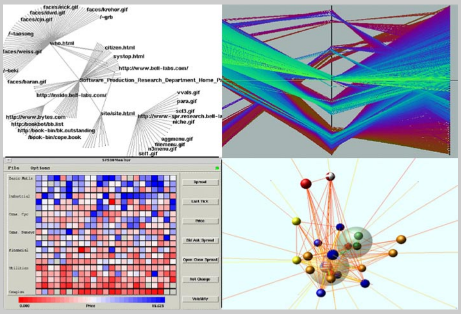
Technique / Description	Example
Geometric Techniques **Basic Idea:** Visualization of geometric transformations and projections of the data **Examples:** Scatterplot, Projection Views, Landscapes, Hyperslice, Parallel Coordinates	
Icon-based Techniques **Basic Idea:** Visualization of data properties as features of icons **Examples:** Chernoff Faces, Stick Figures, Shape-Coding, Color Icons, Tile Bars	
Pixel-oriented Techniques **Basic Idea:** Each attribute value is represented by one single pixel **Examples:** Recursive Patterns, Circle Segments, Spiral- & Axes-Techniques	 Basic Metls Industrial Cons. Cyc Cons. Noncyc Financial Utilities Conglom. 0.000 Price 85.625 Spread Last Tick Price Bid Ask Spread Open Close Spread Net Change Volatility
Hierarchical Techniques **Basic Idea:** Visualization of the data using a hierarchical partitioning into subspaces **Examples:** Dimensional Stacking, Worlds-within-Worlds, Treemap, Cone Trees, Info-Cube	 faces/eick.gif faces/krehor.gif faces/dwd.gif /~grb faces/cjn.gif /~taosong who.html citizen.html faces/weiss.gif systop.html http://www.bell-labs.com/ /~beki Software_Production_Research_Department_Home_Pa faces/baran.gif http://inside.bell-labs.com/ vvals.gif para.gif sel3.gif site/site.html http://www.bytes.com http://www-spr.research.bell-la niche.gif http:/bookbot/bb.list http:/book-bin/bk.outstanding /book-bin/cope.book aggmenu.gif
Graph-based Techniques **Basic Idea:** Visualization of large graphs (2D/3D) using techniques to convey the meaning of the graph efficiently **Examples:** Straight-Lines Graph, Curved-Lines Graph, Orthogonal Graph, Cluster-Optimized Graph, Symmetry-Optimized Graph	

Table 1.1 Classification of visualization techniques

Technique / Description	Example
Distortion Techniques **Basic Idea:** Distortion of the image to allow a visualization of larger amounts of data **Examples:** Perspective Wall, Bifocal Displays, TebleLens, Hyperbolic Representation	
Hybrid Techniques **Basic Idea:** Integrated use of multiple techniques in one or multiple windows to enhance expressiveness **Examples:** IVEE, XmDv	

Which technique would be the best choice for a certain problem depends on the one hand on the type of data and on the other hand on the particular data aspects of interest. Virtually any data may be processed by several visualization techniques. However, each one will consider different facets in the resulting visual representation. Therefore, we should also take a short survey on properties of analysis tasks. The following classification has been derived from a proposition presented in [Kei97b]

Explorative Analysis (with or without any previous knowledge)

Start: data without hypothesis about it
Process: interactive, usually undirected search for patterns, trends, etc.
Result: visualization of the data which might provide a hypotheses

Confirmative Analysis

Start: hypothesis about the data
Process: goal-oriented examination of the hypothesis
Result: visualization, which allows the user to confirm or reject the hypothesis

Presentation

Start: facts to be presented are well known and approved
Process: choice of an appropriate presentation technology
Result: high-quality visualization presenting the facts

For the task of explorative analysis, where apriori no hypothesis about the data exists, the technique of choice must be able to handle a large volume of data at once while respecting its complexity and/or dynamics. In contrast, we may select a less complex technique when dealing with *confirmative analysis* problems. Since there already exists an understanding of the data, basic or geometrical techniques may be adequate for the job. Greater parts of the information presentation task is covered by the distorting techniques, which concentrate on the appearance, accessibility and usability of the data. These assignments are only rough and there is some degree of overlap, for example high quality rendered graph-based techniques may also apply for presentation purposes perfectly.

1.1.4 Frameworks and Systems

Due to the tremendous importance of visualization methods various commercial systems and frameworks have successfully been designed in the past, part of which are available as commercial products. One of the pioneers is AVS/Express [AVS97] that uses a data flow paradigm and allows the user to compose a visualization application interactively by definition of data flow paths between individual modules. Similar paradigms have been implemented in the IRIS Explorer [SGI91] or in IBM's DataExplorer [IBM91]. Another elegant visualization library is provided by General Electric's VTK [SML96] and can be customized in TCL/TK. However, most of the general purpose toolkits and libraries target at classical scientific visualization of spatial data.

For information visualization and visual data mining sophisticated algorithms and metaphors had been devised in recent years to visually inspect abstract and multidimensional information spaces (see also Section 1.1.3). However, generic frameworks and systems are rare and mostly limited in their adaptabilities, such as statistics packages, like Bell Lab's XGobi [ATT97], IVEE [AW95] or the hierarchical algorithms in SGI's SiteManager. Conversely, visual data mining tools, like SGI's MineSet [SGI97] often provide visualization functionality, but, have limited flexibility regarding the integration of new data types and metaphors. The latest representatives of frameworks developed by Visual Insight [VI97] and Visual Decision [VD98] appear to be open and more flexible in their core design.

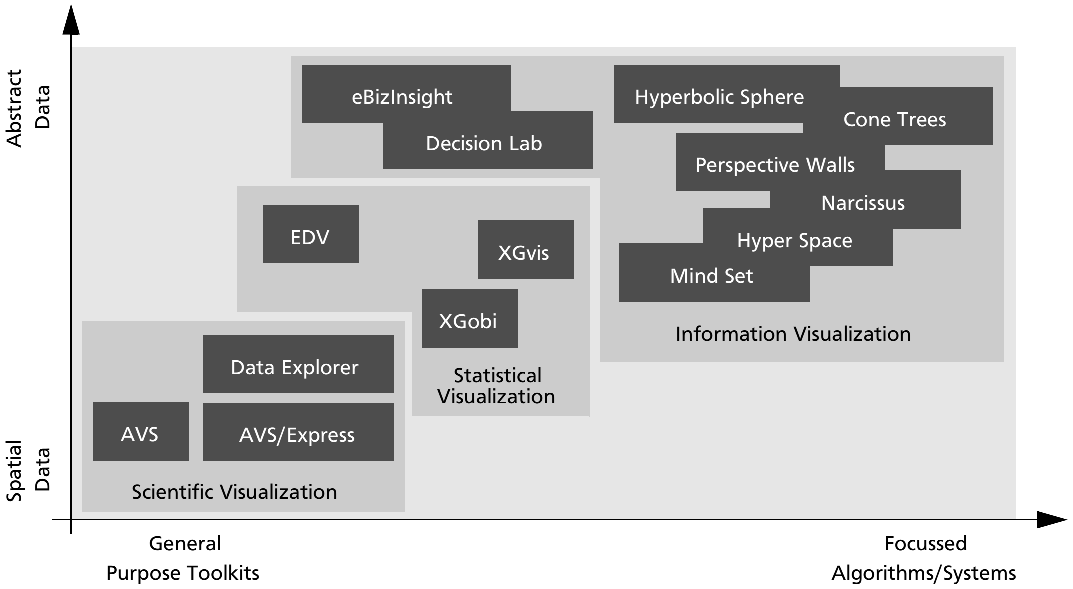


Figure 1.1 Classification of established visualization frameworks, systems and techniques. Drawing their flexibility against their visualization domain.

1.2 CONTRIBUTION

The main contribution of this thesis is the presentation of a sound and versatile framework for physics-based information visualization. The key concept is an efficient multi-resolution setup, breaking down the structural and visual complexity of scenes.

- *Visualization Techniques*: We present a generalized variant for physically-based visualization of multi-dimensional relations in arbitrary and large data sets. The approach is based on the quantification of the similarity of *abstract* data objects, which then governs the parameters of a mass-spring system. Relaxation of the model figures out the structural relations in information space. Much effort has gone into on finding appropriate particle simulation methods which allow stable and reproducible simulation results, as they are needed for the layout algorithm. However, not much attention was given to the development of new numerical methods on our own.

- *Similarity Measurement*: Information visualization targets data sets, which inherently lack a quantifiable similarity measurement. This work presents a systematic approach to quantify similarity for the most common data types encountered in this research field.

- *Framework Design*: Although there already exist a number of systems, most lack general usability. We developed a generic framework for physics-based visualization and the analysis of multidimensional data relations. The system is open and expandable and follows a consequent separation of data-dependent and data-independent components. It is capable to handle static as well as dynamic data likewise. Individual visual metaphors and interaction paradigms are invoked via an advanced plug-in mechanism. Alternative layout and clustering methods are also plugable at runtime.

- *Configuration Language*: We introduce a fully object-oriented script language named *Information Visualization Modeling Language* (IVML), which is specially designed to describe information visualization problems. The language accomplishes rapid prototyping and dynamic configuration for the framework.

- *Complexity Handling*: The complexity and amount of data forces the use of an efficient level-of-detail strategy. In order to efficiently cope with large data sets we propose the ellipsoidal clustering, which wraps similar objects with an ellipsoidal surface. For complex data volumes two visually superior clustering mechanisms–BLOB and H-BLOB–enable a hierarchical encapsulation of similar objects by implicit shapes.

- *Usability Testing*: We show the framework's versatility by experimental results, demonstrating IVORY's capability to simplify and enhance the feasibility of cluster visualization. In addition, a usability test including daily-business cases in the areas of our cooperation partner UBS has validated IVORY's performance in practice.

We have published research results from our physically-based information visualization method in [SGE97, GSF97]. The concepts of the IVORY framework design and the information visualization modeling language have been published in [SGB98]. The BLOB clustering and its hierarchical successor the H-BLOB methods have first been presented in [SGB98, SBG00]. Further on, three additional publications [HDH+00, HDH+00 and HHD+01] with HP Research Laboratories (Palo Alto, CA) demonstrate the practical value of many of IVORY's core ideas.

1.3 CHAPTER OUTLINE

The remainder of the thesis is outlined as follows:

- Chapter 2 places the fundamentals for a proper framework design presented later on in Chapter 6. An operation model is discussed, in order to systematically examine the interaction between framework components. The proposed operator classification enables us to design new components while keeping their general reusability in mind.

- Chapter 3 introduces a mathematical framework for the quantification of the similarity of related objects located in arbitrary information spaces. The framework depends on the well-known theory of metric spaces. Furthermore, a selection of commonly used similarity functions is given.

- Chapter 4 describes physics-based models and their key issues more detailed. This includes the presentation of an advanced force model, a spherical initialization approach for the model and a study of numerical integration methods.

- *Chapter 5* contains a introduction to graph theory with respect to information visualization. We define the conventions used when drawing as well as a set of key issues that a layout may be required to satisfy. In particular, we take a closer look at force-directed graphs and show the interplay with the methods introduced in Chapter 4.

- *Chapter 6* contains the description of IVORY, a physics-based framework for information visualization, which reveals multidimensional relations and adjacencies in terms of spatial neighborhoods. The framework bases principally on algorithms and methods introduced in the previous chapters. In addition, different conceptual aspects of the framework architecture as well as implementation issues are described.

- *Chapter 7* examines established clustering algorithms concerning the demands of today's information visualization systems emphasizing specific advantages and disadvantages. Upon the analysis' insights, we introduce three new visual clustering algorithms called Ellipsoidal, BLOB and H-BLOB clustering. They provide effective approaches for efficient level-of-detail strategies and fit seamlessly into the IVORY framework presented in Chapter 6.

- *Chapter 8* introduces the information visualization modeling language (IVML), which is distinctively proposed to describe information visualization problems. After determining the requirements, we present the language's design concepts. The section closes with the language specification and the implementation issues.

- *Chapter 9* shows the versatility of the framework by experimental results, demonstrating IVORY's capability to efficiently represent real-world configurations in a visual format. These examples cover the areas of financial analysis, e-commerce, document retrieval and image retrieval. Additionally, we present the results of the usability study realized in cooperation with our partner UBS, which focuses on the general applicability of IVORY's new visualization paradigm.

- *Chapter 10* summarizes the conclusions of this thesis and points out future directions.

- The *appendix* includes the package layout of the IVORY framework components as well as the grammar of the information visualization modeling language (IVML). Furthermore, the original questionary of the usability test introduced in Chapter 8 is appended.

2

OPERATOR FRAMEWORK ARCHITECTURE

The aim to develop a new *information visualization framework*[1] (IVF) calls for some thorough preparational work. One of the key factors for a successful framework design is a well-devised and proper definition of its structure and components. There are many considerations to be addressed by the design. In fact, the demands on IVF or on frameworks in general are manifold, which usually concludes in a trade-off design approach. On the one hand, a framework should be as generic as possible in order to be applicable to a large variety of problem classes. On the other hand, it has to handle very domain specific problems, encapsulate them in components and thus hide them from the framework user.

In addition, another important issue is the question about the reusability of components. Since, often we already have a solution for one problem, we would like to reuse it – whenever possible – in similar cases. However, the large variety of information visualization operators and their manifold applications make them difficult to implement in a reusable manner. Therefore, a powerful framework design should consider this aspect in its core structures.

This chapter places the fundamentals for a proper framework design presented later on in Chapter 6. An operation model is introduced, in order to systematically examine the interaction between framework components. The proposed operator classification enables us to design new components while keeping their general reusability in mind.

2.1 FUNCTIONALITY

Information visualization has made great strides in the development of a semiology of graphical representation methods [CM97, Mac86]. Yet, it lacks a framework for studying

---

[1] In the context of this work we do not explicitly distinguish between the expressions *framework* and *toolkit*. We will call the definition of a modular concept a framework, as well as its implementation.

visualization operations. This stands in clear contrast to the area of scientific visualization with its much longer background, where quite large a number of general purpose toolkits is available [AVS97, IBM91, SML96]. There, past research in data-flow networks and the visualization pipeline [CM97] helps to understand the implied visualization processes, the operators and their interaction.

With our framework we explicitly propose an operator-centric design approach, which will lay the systematic foundations for an effective and consistent IVF design. By *operator* we generally mean all functions applied directly to the data or to one of its possible representations, producing a transformation or generating a new representation or view of the original data. In opposite to data-centric approaches that support overloading of operators – commonly used in the object oriented world – we split a single operator into several different operators, if it appears to be able to work on multiple data types. On one hand this assumption leads to a sound theory, on the other hand it actually corresponds to the reality of information visualization systems.

In fact, we introduce an entire *operator framework* (OF) for information visualization systems, which again serves as a skeleton to build upon the overlaying IVF (see Chapter 6). Primarily, the OF will assist us in the following three tasks:

1. Understanding the interactions between different operators
2. Examining and recording operator properties
3. Developing a classification for the space of operators.

Previous work concerning this topic has been published in [CR98]. The authors present a powerful and flexible approach based on the semiology of graphic representations. They describe the properties of such a framework and use it to characterize typical operations relevant to their field of application and the focus of their impact. Their work addresses problems specific to framework designers as well as users-specific issues.

Although our work pursues similar goals it differs in some important aspects. First, we extend the considerations by a finer-grained abstraction model. Second, we exclude all aspects concerning the unification of the interaction and the visualization. In our opinion, this topic has its very own challenges which should be discussed separately. Along with it, we also eliminate the arbitrarily merging of end-user and designer aspects. Hence, we concentrate our reflections on problems arising in the domain of framework designers only. Finally, for the sake of an improved presentation our approach follows a reversed argumentation chain compared to [CR98].

2.2 FUNDAMENTAL CONCEPTS

The presented concept is founded by systematically determining fundamental properties of operators from the information visualization point of view. These will determine the criteria for the operator classification presented later on in this section.

Chuah and Roth [CR96] previously presented a model that categorizes *basic visualization interactions* (BVI). The classification taxonomy shown in Figure 2.1 is based on Foley's user interaction framework [FVF+90] and extends it for the purpose of a more detailed handling of information visualization specific operations, such as data filters.

Starting from the root node on the left side, the first hierarchy step subdivides the interactions according to the corresponding argument type they work on. On the upper end,

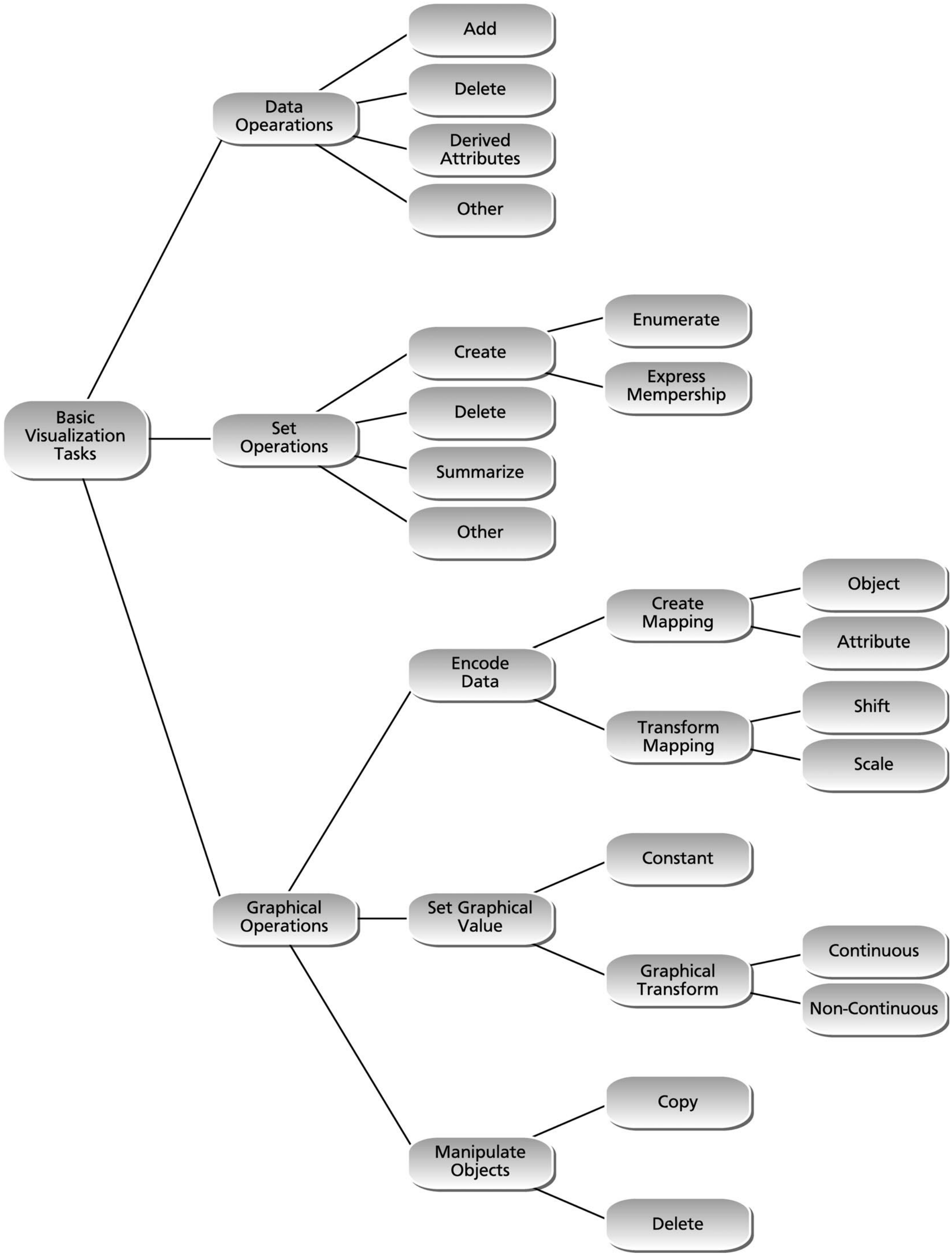


Figure 2.1 Chuah and Roth's Basic Visualization Interaction Taxonomy [CR96].

we find a set of data interactions. They run directly on the raw data and may change the state of the data by such processes as adding, deleting or modifying subsets of the data. In the middle section operations run on sets. They mainly hold meta information derived from the raw data by some extraction processes. Operations on this level may modify a logical intermediate state of the data. Finally, on the lower end the graphical operations are located. They change the visualization content only – typically no underlying data set will be changed. Examples of such operators include view transformations (rotation, translation, zooming, etc.), the manipulation of graphical objects (copy, delete, etc.) and the encoding of data properties to visual attributes.

Inspired by this work we will ameliorate the principal idea as far as operator reusability is concerned. In particular, the aspect of data flow in such systems deserves some further consideration. In addition, Chuah and Roth define a taxonomy tree. This classification scheme works hierarchically, meaning that the classification criteria only make sense with respect to the corresponding parent operation. This kind of scheme is well suited to categorize a fixed set of frequent operators. However, our experience has shown that for the current problem a flat partitioning of the operator space, according to a small set of global criteria, proves to be more useful.

Nevertheless, Chuah and Roth's work provides important indicators regarding a classification with respect to operator reusability. Especially, the first level of hierarchy – the division in data, set and graphic operations – seems to be an interesting starting point. A quite similar partition is exhibited by the visualization pipeline [Gro94] defined in Figure 2.2.

On the upper end, the pipeline starts with the raw data. Over two intermediate stages the pipeline ends in the visualization. Fundamentally, it describes a transition from raw data values to a visual representation (view) of the same data. On each stage of this pipeline various operators may be applied. Thereby, we may identify two different types. First, there are the operators, which act on a single stage within the pipeline. This means, the input arguments and the result are members of the same stage. Second, we have the operators taking input arguments from one stage, but producing a result that belongs to the next stage. Thus, we may consider stage transitions as a result of corresponding operators.

As a first result we can categorize operators according to the stage the operator is involved in and whether the operator works within a single stage or not.

Table 2.1 Categorization of operators according to its input arguments and the result type

Operator Category	Input	Result
Data Stage Operators	RD	RD
Data Transformation Operators	RD	AA
Analytical Abstraction Stage Operators	AA	AA
Visual Mapping Operators	AA	VA
Visualization Abstraction Stage Operators	VA	VA
Visual Transformation Operators	VA	V
View Stage Operators	V	V

RD: Raw Data — **AA:** Analytical Abstraction — **VA:** Visualization Abstraction — **V:** View

Basing on this first categorization the next section develops an abstraction model in order to examine the aspect of operator reusability in more detail. This will then lead to a set of characteristic properties and finally to a reusability classification scheme.

2.3 ABSTRACTION MODEL

Frequently, published concepts dealing with the current topic suffer from limited power of distinction. Because these models run on the restricted differentiation between the categories of value and view operators, they essentially consider two different levels of abstraction only. As the previously introduced BVI model shows, these approaches provide a too polarized view. Furthermore, the binary distinction between a value and a view operator is not always clear.

Our abstraction model is based on the visualization pipeline with its different stages (see Figure 2.2), where each stage holds a specific representation of the original data. At the outset, we have the *raw data* which may be of any type and format (HTML-Documents, DB-Records, images, complex geometry, etc.). Examples for operators working on this stage are data parsers and data specific filters. On the next stage, we find the *analytical abstraction* of the underlying data. It embodies the data in a generalized and abstract manner (weighted graph, feature vectors, etc.). This representation is also called metadata or information. This analytical abstraction is further reduced into some form of *visual abstraction*, which is a visualizable representation. The data here consists of attributed geometrical objects (boxes, spheres, polygons, etc.). Finally, a visual transformation produces a *view* that may be displayed on a screen. This completes the pipeline.

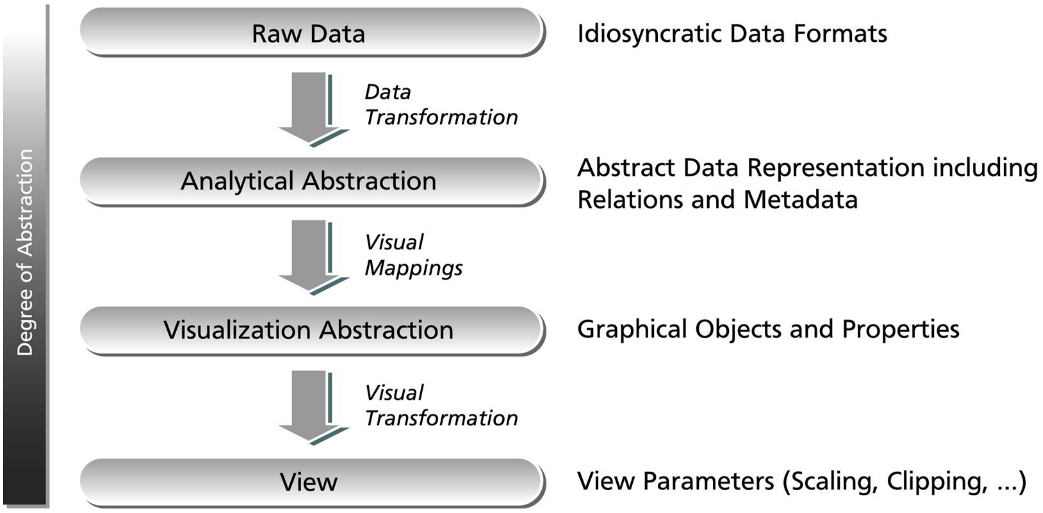

Degree of Abstraction
Raw Data
Idiosyncratic Data Formats
Data Transformation
Analytical Abstraction
Abstract Data Representation including Relations and Metadata
Visual Mappings
Visualization Abstraction
Graphical Objects and Properties
Visual Transformation
View
View Parameters (Scaling, Clipping, ...)

Figure 2.2 The standard visualization pipeline adapted for the purpose of information visualization. The pipeline is annotated with the corresponding data types for each stage and their degree of abstraction

While examining the different data representations associated with each stage, we may come to the very intuitive conclusion that the degree of abstraction increases with the level of a stage along the pipeline. Starting with the raw data the generalization of its representation increases with each transformation and eventually ends in a view regardless of the initial type of data. Thus, the raw data is a concrete instance of a data representation. It may occur in an arbitrary number of variations. By contrast, the view embodies the most general stage. This grounds on the fundamental assumption that once we reach the stage of a view we are dealing with graphic primitives such as points, lines and polygons that a view can operate on in exactly the same ways. One and the same type of view may suffice to handle a large amount of peculiarity raw data may exhibit.

One should be aware that the process of data transformation down the pipeline is typically not lossless. Thus, in order to minimize the loss of information optimization aspects play an important role when determining the concrete set of data mappings and transformations.

As a matter of fact, it may be an essential necessity to have a larger set of views occasionally. However, this requirement does not establish on the inability of a view to manage the amount of underlying data. It is rather based on the demand for multiple views on the same data, where each view may be parameterized individually.

Summarizing, the information visualization pipeline comprises four different degrees of abstraction. The degree of abstraction increases along the direction of the pipeline, where the view represents the stage with the highest degree (see Figure 2.2). Thus the challenge of operator reusability is no longer a question of "value" or "view", but rather a question of the degree of abstraction the operator acts on. Value and view represent thereby only the two extrema.

2.4 CLASSIFICATION OF OPERATORS

The previous observations together with the operator categorization from Section 2.2 are supposed to be used for extracting a generalized classification scheme for operators. In the following, unless otherwise mentioned, the reusability of an operator will implicitly be assumed as classification criterion.

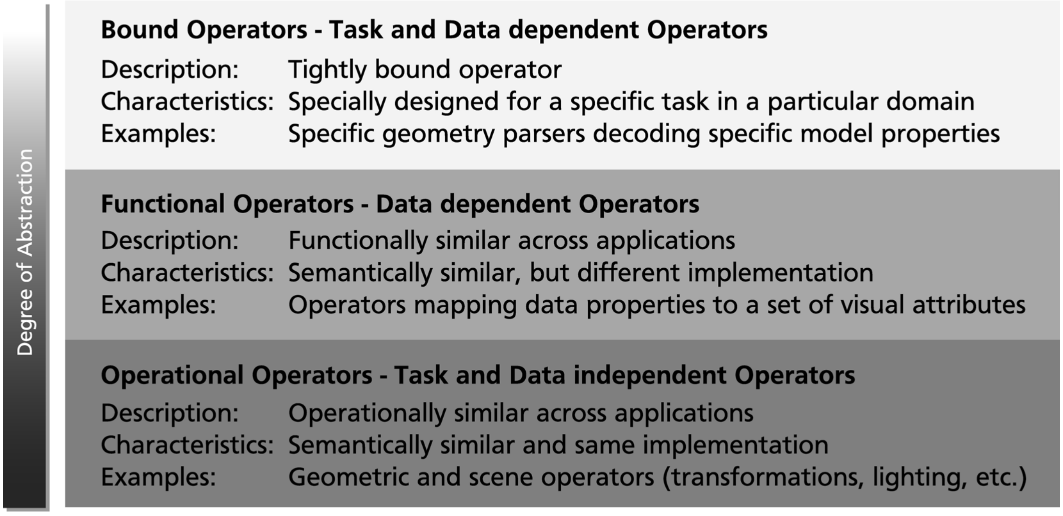
Inspecting operator properties while following the information pipeline from the raw data to the view, we discovered that operators can be classified according to their reusability based on a very small set of criteria. In fact, we may distinguish between operators that are:

1. Specially designed for a specific task
2. Similar according to their functionality
3. Similar according to their implementation

On the lowest level we have the operators that are specially designed for a specific task in a particular domain. These operators are application task and data dependent and thus tightly bound – we call this class the *bound operators*. An example for operators belonging to this class are highly specialized geometry parsers decoding specific model properties in order to control a particular feature detection algorithm.

Next are the operators that have similar semantic, but the underlying implementations differ in a decisive manner. These operators are functionally similar across different applications. However they are still data dependent and consequently they require adaption efforts for different types of data – they are termed *functional operators*. Large parts of the group of database operations, such as adding or deleting data records belong to this operator class. Another example are visual mappers, which are operators that map a set of data properties to a set of visual attributes. In both cases the semantic of operators are the same across applications, however the implementation heavily depends on the underlying data type.

Finally, there is the class of operators that have a similar semantic as well as an exactly identical implementation independent of a given application context. Therefore, these operators perform data independent tasks. They are named *operational operators*, because


Degree of Abstraction

Bound Operators - Task and Data dependent Operators

Description:	Tightly bound operator
Characteristics:	Specially designed for a specific task in a particular domain
Examples:	Specific geometry parsers decoding specific model properties

Functional Operators - Data dependent Operators

Description:	Functionally similar across applications
Characteristics:	Semantically similar, but different implementation
Examples:	Operators mapping data properties to a set of visual attributes

Operational Operators - Task and Data independent Operators

Description:	Operationally similar across applications
Characteristics:	Semantically similar and same implementation
Examples:	Geometric and scene operators (transformations, lighting, etc.)

Figure 2.3 Summarizing list of recognized operator classes

they are operationally similar across applications. The perfect examples here are the classes of geometric and scene operators we know from computer graphics. Concretely, geometry transformations as well as lighting and view operations belong to this class.

Obviously, this classification correlates heavily with the previously developed categorization of pipeline operators and the abstraction model. In a next step we join these three concepts in a multi-layered classification scheme, which is shown in Figure 2.3. By these means, we can deduce essential design issues for a powerful and sound framework design.

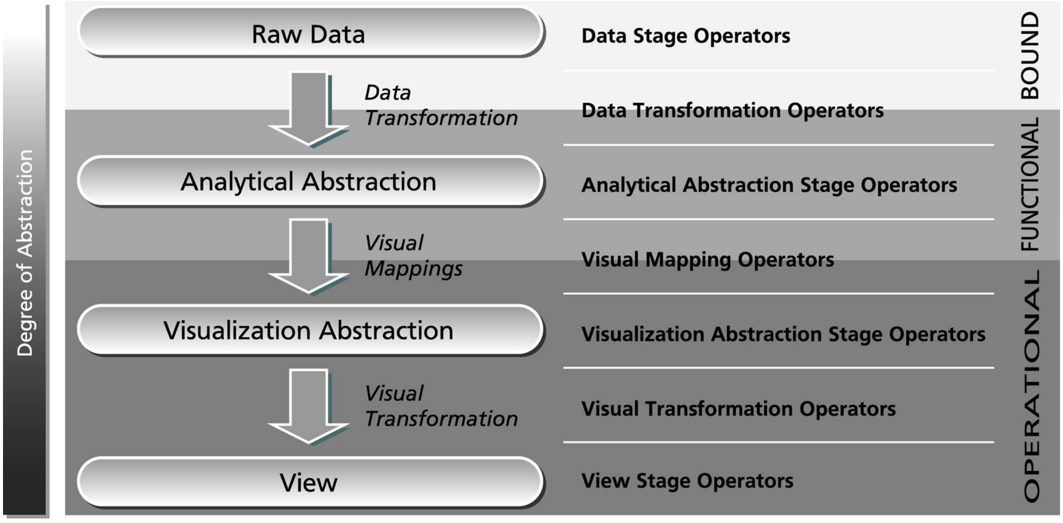


Figure 2.4 Schematic view on the joined concepts of the information visualization pipeline, the operator categorization, the degree of abstraction and finally the operator classification.

2.5 IMPLICATIONS FOR FRAMEWORK DESIGNS

This subsection summarizes the consequences of the theory presented above. Furthermore, it describes how this OF affects the design process of information visualization systems.

A first quite simple realization is the fact that, if we intend to introduce a new operator in a reusable manner it has to be situated in the class of operational operators. From the visualization pipeline point of view this condition is equal to a placement after the visual mapping operators – in other words, the operator should be designed to work on data with the highest possible degree of abstraction. On the other hand, if we know on which stage preexisting operators work, we may identify those that are not application domain specific and thus are easily reusable.

Operations with the same functionality may be invoked on several levels. Typical examples are data filters or clustering techniques. Since, the breadth of an operator depends on how far down it appears in the visualization pipeline, we are tempted to constantly choose a position as close to the view stage of the pipeline as possible. However, this strategy hase also some disadvantages. Especially, in the case of filter-like operators this approach may lead to a considerable and unnecessary increase of data traffic down the visualization pipeline. Since, typical data volumes of serious information visualization problems tend to be large, this strategy may result in a permanently overloaded system. In such a case we prefer to drop the reusability aspect. Data specific filters on an early stage may significantly ease the problem while saving resources.

This knowledge can be used to design an extensible and well-structured IVF. According to the different operator classes and their corresponding property profiles, operators are implemented within the system's kernel or, alternatively, in plugable modules. The framework supports the modularization of the system kernel. Furthermore, it helps with the definition of a sound interface architecture connecting such modules.

And finally, by forcing operation designers to think about where a given operator may exist in the pipeline, the operator's semantics are made explicit. Hence, a potential end-user can interact more accurately with such a system because he understands how operators at different stages of the pipeline fit together.

3

METRIC SPACES

Metrics occur in many disciplines of computing science (e.g. combinational logic, semantic analysis or fractal image generation) and their meaning varies within a broad spectrum. In the more applied fields, we often consider a metric in the sense of a geometrical distance.

In our case, metrics serve as a base concept for quantifying the similarity of related objects, which will govern the parameters of a spring-embedding system. These in turn, determine the resulting object arrangement in space thereby revealing multidimensional object relations in terms of spatial neighborhood. Therefore, metrics play a major role in unveiling multidimensional relations and adjacencies in terms of spatial neighborhood. Thus, special observation must be given to the design of a metrics. For every specific problem and in particular for the pertinent raw data a corresponding metric must be defined. This chapter is supposed to provide the basics for this purpose.

This chapter introduces a mathematical framework for the quantification of the similarity of related objects located in arbitrary information spaces. The framework depends on the well-known theory of metric spaces. Furthermore, a selection of commonly used similarity functions is developed.

It should be admitted that our presentation of metrics in this section is somewhat focused on the scope of our application domain. The presentation in this chapter follows largely the two references [AM75] and [GA92], to which we refer the reader for a broader and more in depth treatment of the subject.

3.1 TERMINOLOGY AND BASIC DEFINITIONS

Definition. A *metric* (or *distance function*) on a set X is a function $d : X \times X \rightarrow \boldsymbol{R}^{0+}$ mapping a pair of elements to a non-negative real number, satisfying the following four axioms:

1. $d(x, x) = 0 \qquad ; \forall\, x \in X$
2. $d(x, z) \le d(x, y) + d(y, z) \qquad ; \forall\, x, y, z \in X$

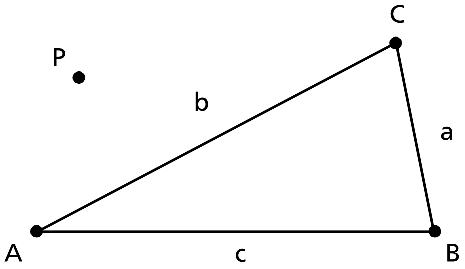
3. $d(x, y) = d(y, x) \qquad ; \forall\, x, y \in X$
4. $d(x, y) = 0 \Rightarrow x = y \qquad ; \forall\, x, y \in X$

Each condition is to be understood as universally quantified with respect to x, y, z. Axiom 1 may be termed *reflexivity*: the distance from an element to itself is always 0. Axiom 2 – called the *triangle inequality* – asserts that one cannot shrink the total distance traveled by making detours. Axiom 3 is *symmetry*, meaning that the distance from x to y is the same as the distance from y to x. Finally, Axiom 4 is the inverse of Axiom 1 – called *identity*.

Example: One of the most familiar distance functions is the Euclidean metric $d : \boldsymbol{R}^2 \times \boldsymbol{R}^2 \to \boldsymbol{R}^{0+}$ defined by

$$d((x_1, x_2), (y_1, y_2)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

In this case the metric axioms can easily be checked by visual inspection. Let A, B, and C be three points in $\boldsymbol{R}^2$, that span a triangle *ABC* with sides *a*, *b*, *c* as shown here.



Axioms 1, 3 and 4 are immediate. Clearly $d(A, A) = 0$, $d(A, B) = d(B, A) = c$ and $d(A, P) = 0 \Rightarrow \overline{AP} = 0 \Rightarrow A = P$, where P is an arbitrary Point in $\boldsymbol{R}^2$. For Axiom 2 (triangle inequality) $d(A, C) \le d(A, B) + d(B, C)$ which is equal to $b \le c + a$ and obviously true for any triangle *ABC*.

Definition. A set X together with a distance function d defines a *metric space*, in the following notated as (X, d).

Moreover, we distinguish between different sub-categories of metric spaces. They meet varying combinations of the axioms listed previously. The most familiar sequence of these axioms is [1, 2, 3, 4], which says that (X, d) is a *proper metric space*. If we weaken this definition, namely by dropping Axiom 3, we speak of a *quasi-metric space*. Dropping the symmetry axiom makes sense, when we think of the distance $d(x, y)$ as a measure of the effort involved going from x to y; then think of hilly terrain. We should point out that for quasi-metric spaces the notation of Axiom 4 has to be logically expanded to the formulation $((d(x, y) = 0) \wedge (d(y, x) = 0)) \Rightarrow x = y$ for all $x, y \in X$. Further, if Axiom 4 is dropped from one of the formulations considered so far, the corresponding metric name will be joined by the prefix “pseudo-”; thus, a *pseudo-metric space* is defined by [1, 2, 3], a *pseudo-quasi-metric space* by [1, 2], etc.

3.2 MAPPINGS

Following the definitions, this subsection discusses *mapping functions* (or *maps*) between metric spaces. Applied to two metric spaces, they allow – from a theoretical point of view – the projection of features from one space into the other. That is, supposed we have two metric spaces and we know the features of one of them. If we manage to define a mapping function connecting the two spaces, we may assume corresponding features for the second

Table 3.1 Summary of metric categories

Name	Met Axioms	Dropped Property
Metric	1, 2, 3, 4	
Quasi-metric	1, 2, 4	Symmetry
Pseudo-metric	1, 2, 3	Identity
Pseudo-quasi-metric	1, 2	Identity and symmetry

metric space. In addition, maps present a powerful tool to derive new metric spaces based on already existing ones.

For the reason of simplicity we will state mapping definitions only for *metric spaces* even when they apply without adjustment to all the subclasses of metric spaces mentioned in the previous subsection. However, comment will be reserved for those situations in which, the dropping of one of the axioms does make a significant difference.

Definition. Let (X, d) and (Y, e) be metric spaces. A function $f: X \to Y$ is an *isometry* from (X, d) to (Y, e) if for all $x_1, x_2 \in X$, $e(f(x_1), f(x_2)) = d(x_1, x_2)$, that is, if f preserves distances; in this case, the function f is designated also as *isometric*.

Except for pseudo-metric spaces, an isometry defines one-to-one relationship; for if $x_1 \neq x_2$ then $e(f(x_1), f(x_2)) = d(x_1, x_2) \neq 0$, which implies by Axiom 1 $f(x_1) \neq f(x_2)$. Because axiom 4 has been dropped for pseudo-metric spaces the first implication could obviously be invalidated. Consider $f: (X, d) \to (Y, e)$. Then f is termed *non-expansive* if, for all $x_1, x_2 \in X$, $e(f(x_1), f(x_2)) \leq d(x_2, x_2)$, while f is *uniformly continuous* provided

$$\forall \varepsilon > 0\ \exists \delta > 0\ \forall x_1, x_2 \in X : d(x_1, x_2) \leq \delta \Rightarrow e(f(x_1), f(x_2)) \leq \varepsilon . \tag{3.1}$$

The idea of uniform continuity is that if two points are close in X then their images are to be close in Y, uniformly across the space X. Note that, isometry is the strictest notion of a structure-preserving map between metric spaces. That gives us the following implication chain:

$$f \text{ is isometric} \Rightarrow f \text{ is non-expansive} \Rightarrow f \text{ is uniformly continuous.} \tag{3.2}$$

Definition. $f: (X, d) \to (Y, e)$ is an *isomorphism*, if f is an isometry and f is onto. A function $f: X \to Y$ is onto (or a surjection) if $f(X) = Y$. I.e. f can return any value in Y. This means that its image is equal to its codomain. In this case f^{-1} is also an isomorphism, since $d(f^{-1}(y_1), f^{-1}(y_2)) = e(ff^{-1}(y_1), ff^{-1}(y_2)) = e(y_1, y_2)$.

Definition. If $f: (X, d) \to (Y, e)$ is a *Lipschitz map* if there exists $\lambda > 0$ such that $e(f(x_1), f(x_2)) \leq \lambda d(x_1, x_2)$ for all $x_1, x_2 \in X$. Lipschitz maps are much more general than isometries, but of course, every isometry and every non-expansive map is a Lipschitz map with $\lambda = 1$.

3.3 CONSTRUCTIONS

We now point out some of the principal ways of constructing new metric spaces from old ones. Particularly, we consider techniques to build sub- or supersets of metric spaces, symmetrize quasi-metrics or lift a pseudo-metric space to a metric space.

Weighted Sum Construction. Let (X, d_i), where $i \in \boldsymbol{N}$ be a collection of metric spaces; all defined on the same set of objects X, but with different metrics d_i. The *weighted sum construction* defines an appropriate overall distance function as a weighted linear combination defined by

$$d(x, y) = \sum_i w_i d_i(x, y), \text{ with } \sum_i w_i = 1 \text{ and } w_i > 0 \text{ for all } i. \tag{3.3}$$

The weights w_i allow to control the influence of each metric function d_i. It can easily be shown that (X, d) still defines a metric space, by meeting all 4 Axioms. However, if one of the metric spaces of (X, d_i) belongs to the class of pseudo- or quasi-metric spaces, needs a closer examination.

Axiom 1,2 ✓

Axiom 3 $d(x, y) = \sum_i w_i d_i(x, y) = \sum_i w_i d_i(y, x) = d(y, x)$

$\Rightarrow$ all (X, d_i) must satisfy Axiom 3

Axiom 4 $d(x, y) = \sum_i w_i d_i(x, y) = 0$

$\Rightarrow d_i(x, y) = 0$ for all i

$\Rightarrow x = y$

$\Rightarrow$ some (X, d_i) must satisfy Axiom 4

Note that (X, d) is a pseudo-metric space provided all (X, d_i) are pseudo-metric spaces, while for (X, d) to be a quasi-metric space it is sufficient that at least one of the d_i is a quasi-metric.

- all (X, d_i) satisfy Axiom 3 $\Rightarrow (X, d)$ satisfies Axiom 3
- some (X, d_i) satisfy Axiom 4 $\Rightarrow (X, d)$ satisfies Axiom 4

Metric Subspace. This is one of the simplest construction schemes and produces lots of new metric spaces. If $A \subseteq X$, (X, d) is a metric space, the subspace metric d_A is just the restriction of d to A, defined by $d_A(x, y) = d(x, y)$ for all $x, y \in A$. Then (A, d_A) is a *metric subspace* as is trivial to verify. In fact, d_A is the unique metric on A rendering the inclusion function $i : (A, d_A) \to (X, d)$ an isometry.

Example: Consider $f : \boldsymbol{R} \to \boldsymbol{R}$ defined by $f(x) = x^2$. Then $|f(x_1) - f(x_2)| = |x_1^2 - x_2^2|$ $= |(x_1 + x_2)(x_1 - x_2)| = |x_1 + x_2||x_1 - x_2|$. As $x + y$ can be made arbitrary large, $f : (\boldsymbol{R}, d) \to (\boldsymbol{R}, d)$ is not Lipschitz if $d(x_1, x_2) = |x_1 - x_2|$. On the other hand, let $A = \{x \in \boldsymbol{R} | (|x| \le 5)\}$. Then $|x_1 + x_2| \le 10$ for $x_1, x_2 \in A$ so that $f : (A, d_A) \to (\boldsymbol{R}, d)$ is a Lipschitz map since $|f(x_1) - f(x_2)| \le 10|x_1 - x_2|$.

Symmetrizing a quasi-metric space. There are cases, where symmetric metric spaces are required. For example the similarity space defined on a set of objects (see Section 3.4). However, in a straight forward approach, it is sometimes hard to find a symmetric metric formulation. With less an effort, one could define a quasi-metric on the same set of objects, setting up a quasi-metric space. For this case we consider a useful process for *symmetrizing a quasi-metric space*. Let (X, d) be any quasi-metric space. Define the *conjugate* of d, written d^{-1}, by

$$d^{-1}(x, y) = d(y, x). \tag{3.4}$$

One easily verifies that d^{-1} is a quasi-metric on X. Then we obtain a (symmetric) metric space (X, d^{*}) by taking

$$d^{*}(x, y) = max(d(x, y), d^{-1}(x, y)). \tag{3.5}$$

As we can easily prove, d^{*} meets all metric axioms and (X, d^{*}) specifies a metric space.

Example: Consider the quasi-metric space (X, d), where X is the set of real numbers $\boldsymbol{R}$ and d is defined by $d(x, y) = x - y$. Then $d^{-1}(x, y) = y - x$ and d^{*} becomes

$$d^{*}(x, y) = max(x - y, y - x) = |x - y|.$$

Lifting a pseudo-metric. Finally, we remark that any pseudo-metric evolves in a rather trivial way to a metric, by identifying points having 0 distance. If (X, d) is a pseudo-metric space, we define (X', d'), where $X' = X/\equiv$ is the set of equivalence classes of elements of X under the equivalence given by $x \equiv y \Leftrightarrow d(x, y) = 0$ and $d'(x, y) = d(x, y)$. That means, all the objects of X with $d(x, y) = 0$ are projected onto one single element of X, thus $X' \subseteq X$.

Metric-stretcher. Given is a function $f : \boldsymbol{R}^{0+} \to \boldsymbol{R}^{0+}$. If f satisfies the conditions $f(0) = 0$, f is one-to-one and $f(x) + f(y) \geq f(z)$ if $x + y \geq z$, then f is called a *metric-stretcher*. It can easily be shown that, if $d : X \times X \to \boldsymbol{R}^{0+}$ is a metric and $f : \boldsymbol{R}^{0+} \to \boldsymbol{R}^{0+}$ is a metric-stretcher then $d_f : X \times X \to \boldsymbol{R}^{0+}$ defined by $d_f(x, y) = f(d(x, y))$ is again a metric.

3.4 QUANTIFYING SIMILARITY IN INFORMATION SPACE

One of the very challenging problems of information visualization is the definition of a mathematical framework for the quantification of similarity of entities (= items) in information space. One should be aware that because of its manifoldness this field is still a hot topic in ongoing research. In general, similarity quantification may be described as a function that associates a numeric value with a pair of sequences, with the idea that a higher value indicates greater similarity. A similarity measure is a type of scoring function.

Definition. The *similarity* s on a set of objects O is defined as a 1-normalized, inverted metric[1] $s : O \times O \to [0 \ldots 1]$. The value of $s_{ij} = s(O_i, O_j)$ is computed using a weighted sum construction over the attribute sequences of O_i and O_j. A value of $s_{ij} = 0$ means that the two objects O_i and O_j have minimal and in the case of $s_{ij} = 1$ maximal similarity, whereas the value of the *self-similarity* of an object O_i is always bounded to $s_{ii} = 1$.

Remark, that much like a pseudo-metric a similarity value of $s_{ij} = 1$ does not necessarily imply the identity of the two involved items O_i and O_j – meaning that, O_i and O_j represent one and the same item instance. Nevertheless, in such a case the item attributes are identical in pairs regarding to the criteria defined in the similarity computation. Therefore, we may regard the similarity still as a metric function.

---

[1] For the special purpose of physics-based information visualization we demand a metric fulfilling least Axioms 1, 2 and 3 (see Section 3.1). Therefore, a pseudo-metric is the weakest valid representant at this point.

The definition of an applicable similarity, which by definition is a function operating on the attribute sets of two items, is intrinsically task and data dependent. According to the operator classification given in Section 2.4 on page 16 it belongs to the group of bounded operators. Based on the same items one could define a whole set of distinct similarities; each dedicated to a different task. Therefore, when constructing a similarity measurement we have to be aware what it supposed to express. The general question we have to ask, is:

"What does *similarity* mean in the specific context?"

Without doubt, this process requires a lot of experience value and heuristics. Thus, the support of an expert with domain specific experience and knowledge is essential.

However, the previously presented theory of metrics and metric spaces provides a solid foundation for measuring distances in arbitrary object spaces, thereby making it feasible as an appropriate instrument for similarity measurement. Analogous to the metric theory, we define that a set of entities together with the similarity function span a *similarity space*. Entities which are close in object space, this is with a small distance are "more similar", than those items with a larger distance, which are designated to be "less similar".

In general, we find two different approaches to establish a formal association between a metric space and a similarity space. Starting with the former (and stricter) case, the following section describes two different mapping schemes allowing a well defined construction of a similarity based on an arbitrary but already defined metric function.

Mapping Scheme A. We define a linear mapping function from metric distances to the normalized similarity domain, which reflects this behavior and further takes the similarity specific co-domain into account:

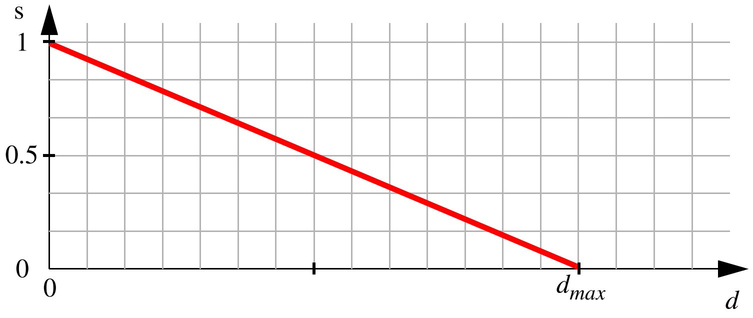


Figure 3.1 Chart showing the interdependence of the similarity value s and the metric value d when applying *Mapping Scheme A* (in case of a known d_{max})

$$s_{ij} = 1 - f(d(O_i, O_j)) . \tag{3.6}$$

The function f is defined as a metric-stretcher (see Section 3.2) that scales the metric function d to a normalized co-domain of $[0 \ldots 1]$

$$f(x) = \frac{x}{d_{max}} , \tag{3.7}$$

where d_{max} designates the maximal possible value of $d(O_i, O_j)$ for a given set of items.

Mapping Scheme B. Less common is an alternative definition of s_{ij} that avoids the involvement of a scaling function. Thereby, s_{ij} will be defined as

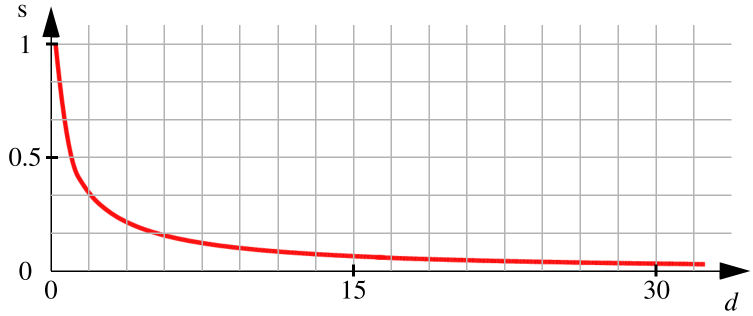


Figure 3.2 Chart showing the interdependence of the similarity value s and the metric value d when applying *Mapping Scheme B* (in case of an unknown d_{max})

$$s_{ij} = \frac{1}{1 + d(O_i, O_j)} . \tag{3.8}$$

Considering the pros and cons of this formulation, we can establish that, compared to the previous definition of s_{ij}, shown in Equation 3.6, this formulation does not require a value for the upper boundary d_{max}. This could be an advantage, particularly if an estimation of d_{max} is very expensive or even impossible to compute for a given problem. However, there are disadvantages as well: Equation 3.8 performs decently only, if the image of the metric function $d(O_i, O_j)$ starts near zero and reaches sufficiently large values. Further, the alternative definition introduces additional non-linear behavior, which could be difficult in the sense of reasonable predictions and control.

In the following subsections we give a selection of different commonly used similarity functions listed by the type of the input data. We do not claim completeness, rather we focus on a basic set of similarities applied in the examples presented later on in Chapter 9.

3.4.1 Booleans

Starting with the simplest form of input data, this function computes a value expressing the similarity between two boolean values – obviously a trivial case.

Input Type: . . . . . . Boolean

Input Values: . . . . . True/False

Metric: . . . . $d(O_i, O_j) = \begin{cases} 0, \text{ if } O_i = O_j \\ 1, \text{otherwise} \end{cases}$

Axioms satisfied:. . . 1, 2, 3

Metric category: . . . Metric

Similarity:. . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f = identity$

The underlying metric function is considered of pseudo-metric type, because it can return a distance value of zero even if $i \neq j$ (see Section 3.1). The projection into the similarity space is done by the a linear mapping function combined with the identity function.

3.4.2 Scalars

In the case of scalar values, we generally construct a valid metric function by applying the absolute value function to the difference of the two scalar input values $d(O_i, O_j) = |O_i - O_j|$. As one can easily verify, this metric evidently satisfies all metric axioms (1 to 4).

A sound mapping to a similarity function can be achieved in two different ways. If the upper bound of $d(O_i, O_j)$ is known, we may apply *Mapping Scheme A* – presented previously – to get *Similarity A* shown below. Otherwise, if the maximum value is unknown, we must use the non-linear *Mapping Scheme B* ending up in *Similarity B*.

Input Type: . . . . . . Scalar

Input Values: . . . . . ***R***

Metric: . . . . $d(O_i, O_j) = |O_i - O_j|$

Axioms satisfied: . . 1, 2, 3, 4

Metric category: . . . Metric

Similarity A: . . . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f(x) = \frac{x}{d_{max}}$

Similarity B: . . . . . . $s_{ij} = \frac{1}{1 + d(O_i, O_j)}$

3.4.3 N-dimensional Scalar Vectors

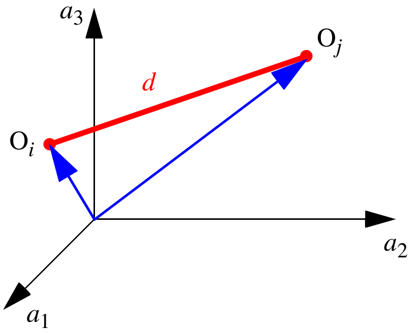
Similarity measurement of n-dimensional vectors calls for a closer inspection. Different metric definitions may be applied to measure the affinity of two n-dimensional vectors. Subsequently, the two most common, the *distance measurement* and the *angle measurement* are discussed in more detail.

Distance Measurement. A common approach to compute the distance between the end points of two vectors, is to apply a *norm* $d(O_i, O_j) = \|O_i - O_j\|$, where the most frequent norm definition is the L_2*-norm*, also known as the *Euclidean norm*. However, we may also think of other norm definitions, such as the maximum-norm L_{max} or the minimum-norm L_{min}; each holding its specific norm characteristics. It depends on the given problem, which norm fits best. However, all these norm variations have in common that a similarity value of 1 is reached only, if O_i and O_j are identical in all of their n dimensions. That is, when the two vectors point to exact the same location in the n-dimensional space.

In the light of an example (Section 3.1), we have already shown that the Euclidean norm fulfills the criteria of a metric function. Therefore, $d(O_i, O_j)$ denotes a true metric function.

Concerning the similarity mapping, analogous problems to the case of scalars occur. Again, depending on the criterion of an unknown d_{max} – that is, if vectors are of arbitrary length – we apply the appropriate scheme from Section 3.4.

Angle Measurement. In contrast to the distance-based similarity measurement, the angle measurement follows a less restrictive approach. The maximum similarity value is



Input Type: . . . . . . Vector

Input Values: . . . . . $\boldsymbol{R}^n$

Metric: . . . . $d(O_i, O_j) = \|O_i - O_j\|$

Axioms satisfied: . . . 1, 2, 3, 4

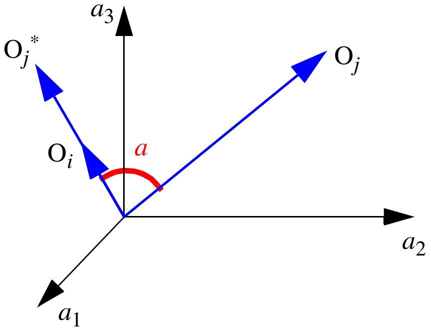
Metric category: . . . Metric

Similarity A: . . . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f(x) = \dfrac{x}{d_{max}}$

Similarity B: . . . . . . $s_{ij} = \dfrac{1}{1 + d(O_i, O_j)}$

already reached, when the two vectors referring to O_i and O_j point to the same direction, and thus the angle between O_i and O_j becomes zero (please compare with vectors O_i and O_j^* in the sketch below).

If two vectors point to the same direction there is a scalar $t > 0$ so that $O_i = t \cdot O_j$. This equation can only be satisfied if each separate component of O_i and O_j matches up. Thus, compared to the distance measurement the similarity condition has been weakened to the effect that it is now sufficient if there is a constant ratio between the components of the two vectors. That means, the unit of measurement now has a relative characteristic.



The metric, underlying the similarity, is based on the definition of the dot product

$$(O_i \bullet O_j) = |O_i| \cdot |O_j| \cdot \cos(\alpha).$$

We may easily extract the cosine of the enclosed angle α

$$\cos(\alpha) = \frac{(O_i \bullet O_j)}{|O_i| \cdot |O_j|},$$

which may already be applied as a metric, in some cases. In order to avoid additional non-linearity, the arc's cosine is computed. Consequently, we get a range of values between 0 and π for this metric. This metric obviously belongs to the category of pseudo-metrics, since – as shown in the previous subsection – the identity criterion is not met as sole exception of the four axioms. In order to get a valid similarity function a metric stretcher with a constant factor of $1/\pi$ is applied, consequently.

Input Type: . . . . . . Vector

Input Values: . . . . . $\boldsymbol{R}^n$

Metric: . . . . $d(O_i, O_j) = \operatorname{acos}\left(\frac{O_i \bullet O_j}{|O_i| \cdot |O_j|}\right)$

Axioms satisfied: . . 1, 2, 3

Metric category: . . . Pseudo-metric

Similarity: . . . . $s_{ij} 1 - f(d(O_i, O_j))$ with $f(x) = \frac{1}{\pi} \cdot x$

Besides the more conservative similarity assessment the angle measurement has the advantage, that it fundamentally works within a limited and well defined co-domain. Therefore, we neither have to consider any case distinction nor do we have to introduce any undesirable non-linearity.

Finally, choosing a similarity measurement for n-dimensional vectors highly depends on the concrete application. Even, if we work on the same data set, we may first choose the distance measurement, but prefer the angle measurement for an analysis with a slightly different focus. A concrete example may be given from the area of document retrieval analysis. Supposed a document is described by an n-dimensional feature vector, where each entry denotes for example the relative frequency of a given keyword within the document's text. If our analysis focuses on finding sets of preferably equal documents, we apply the distance measurement. However, if the search is confined to the finding of documents dealing with a similar topic, then the angle measurement is employed.

3.4.4 Dynamic Numerical Data Series

The class of dynamic numerical data series defines another set of problems that principally bases on n-dimensional data vectors too. However, the mathematical instrument to measure similarity in dynamic numerical data series – also called *timeseries* – is provided by the area of statistics and probability calculus. Since we may consider timeseries as a sequence of discrete random variables, we may apply the theory of descriptive statistics. It defines a measure of distance between two variables by quantifying their stochastic interdependence.

For our particular case, we are interested in quantifying the similarity between the courses of two different data sources. The correlation coefficient r is used to answer the question: Is a change in one of the independent variables linearly associated with a change in the other independent variable?

Linear Correlation. If the data distribution is *normal*, the correlation coefficient r may be defined directly on the data series. The measurement basically relies on a covariance computation. However, since the covariance may become any value depending on the underlying timeseries, this approach will require the use of the unfavorable *Mapping Schema B* in the similarity mapping step. Therefore, a normalized replacement – the *linear correlation coefficient* r_{lin} – is preferred. This coefficient is calculated as:

$$r_{lin} = \frac{\sum(O_i^k - \overline{O}_i)(O_j^k - \overline{O}_j)}{\sqrt{\sum(O_i^k - \overline{O}_i)^2 \sum(O_j^k - \overline{O}_j)^2}}, \tag{3.9}$$

where $\overline{O}_i$ and $\overline{O}_j$ denote the empirical mean value over the corresponding timeseries.

A value of $r_{lin} = 0.0$ indicates no linear relationship between the two timeseries, whereas a value of $r_{lin} = 1.0$ or $r_{lin} = -1.0$ suggests a strong linear relationship between the two timeseries. If r_{lin} holds a positive value, the behavior of the two timeseries shows a similar tendency in principle. This means, that if the values of one series increase over time, the other series will likely rise too. Correspondingly, if r_{lin} is negative the connection behaves reversely.

Input Type: . . . . . . Timeserie

Input Values: . . . . . $\boldsymbol{R}^n$

Metric A: . . . . $d(O_i, O_j) = 1 + r_{lin}$

Metric B: . . . . $d(O_i, O_j) = 1 - abs(r_{lin})$

Axioms satisfied:. . . 1, 2, 3

Metric category: . . . Pseudo-metric

Similarity A:. . . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f(x) = 0.5 \cdot x$

Similarity B:. . . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f(x) = abs(x)$

Depending on the concrete use case, two different variants of a similarity mapping are meaningful. The first variant *Metric A/Similarity A* applies a scaling with a factor of 0.5 . An alternative mapping definition *Metric B/Similarity B* uses the absolute value in order to project the values into a valid co-domain.

Both transformations finally map the values of the chosen metric (see Equation 3.9) to the normalized co-domain of a similarity function. However, they differ decisively in the manner they rate a reversal connection between the two timeseries. Where *Metric A/Similarity A* considers this behavior as completely non-similar, *Metric B/Similarity B* judges it – alike to the case of a similar behavior – as completely similar.

As one may have noticed, this definition seems to be quite similar to the previously presented angle management applied to n-dimensional scalar vectors. In fact, it is mainly a question of how we look at the data. From a statistical point of view we compute an empirical correlation value between two variables changing over time. Switching to a geometrical interpretation, the two data series may be equivalently viewed as n-dimensional vectors. Doing so, the correlation essentially corresponds to the cosine of the angle in between the two corresponding vectors, which have been translated preliminary by their respective mean value.

Again, the metric $d(O_i, O_j)$ belongs to the category of the pseudo-metrics, because the identity axiom is not fulfilled for this case.

Rank Correlation. When the distribution of variables is not normal, the degree of relationship between the variables can be calculated using *rank correlation*. Instead of using the precise values of the variables. The data series are ranked (R_i and S_i) in order of size, and calculations are based on the differences between the ranks of corresponding values. Let D be a data series containing the squared differences:

$$D_i = (R_i - S_i)^2$$

Then, using *Spearman's formula*, correlation is estimated according to:

$$r_{rank} = 1 - \frac{6 \cdot \sum D_i}{n(n^2 - 1)} \tag{3.10}$$

The rank correlation is a robust and resistive alternative to the linear correlation (See also [PFT+88]). Concerning the similarity mapping, the same approaches applied to the linear correlation may successfully suit for this case.

Input Type: . . . . . . Timeserie

Input Values: . . . . . $\boldsymbol{R}^n$

Metric A: . . . . $d(O_i, O_j) = 1 + r_{rank}$

Metric B: . . . . $d(O_i, O_j) = 1 - abs(r_{rank})$

Axioms satisfied: . . 1, 2, 3

Metric category: . . . Pseudo-metric

Similarity A: . . . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f(x) = 0.5 \cdot x$

Similarity B: . . . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f(x) = abs(x)$

3.4.5 Non-Numerical Entities

One may easily think of a large variety of similarity measure examples involving non-numerical entities. To give the reader an impression of the wide thematic range, the work presented in [LL00] may be mentioned as an exponent of a very pictorial example. They present a similarity measure motivated by cognitive considerations based on the method of discrete curve evolution and simplification of visual parts. This shape-matching technique is applied to retrieve similar objects in image databases of 2D objects. At the other end, we have the class of abstract problems. An appropriate example for that is represented by the work of Girardi and Ibrahim [GI94]. They introduce a similarity measure to retrieve affine software parts. The underlying metric bases on the automatic extraction of lexical, syntactic and semantic knowledge from natural language descriptions of software artifacts.

However, in order to stay focused, we will limit the further discussion on non-numerical entities to the class of document retrieval problems. Please be aware, that this restriction mainly applies to the selection of examples following in Chapter 9.

Measure of the similarity between documents. Documents can be characterized by a large number of attributes. Thus, a document may be seen as a n-dimensional attribute vector – a *document description vector (DDV)*. The set of all DDVs set up a *document space*. Each element of such a vector describes one specific property of a document. The meaningful determination of such property values still belongs to the hot topics of document retrieval research and shall not be discussed here. Our further reflections assume that a standard document retrieval engine is involved. Therefore, the DDVs result from a retrieval query by submitting a set of keywords. The involved retrieval engine then determines the value for each element in a DDV which will be sent back to the user.

Table 3.2 Elements of a typical document description vector

Name	Description	Type
`score`	Overall relevance of the document (within a result list)	scalar
`location`	The URL of the document	url
`language`	Language, automatically detected by the retrieval system	enum
`keyword-counts`	Relative frequency for each keyword (search argument)	vector
`length`	Absolute length measured in Bytes	scalar

The similarity of two documents can be measured as the distance between the two corresponding DDV in the document space. A comparable measurement for n-dimensional scalar vectors has already been presented in Section 3.4.3. However, as the example in Table 3.2 shows the elements of a DDV may be of different types.

In order to reduce this case to the one in Section 3.4.3, we must find a way to map the non scalar elements of a DDV to a single number. Again, this problem looks quite familiar to us, if we think back to the beginning of this chapter. In fact, defining an appropriate metric to each single DDV element solves the mapping problem. This gives us a two-stage measure process. In the first stage, we apply a metric on the level of the elements of each DDV. In the second stage, the similarity of the then scalar vectors is quantified based on the formulas for n-dimensional scalar vectors introduced in Section 3.4.3.

The following subsections present metrics for the non-trivial element types. Obviously, `score` and `length` will not appear because they are already scalars. Even the `keyword-counts` element is not of special interest, because its type is already covered by the metric for n-dimensional scalar vectors introduced in Section 3.4.3. Thus, the `language` and `location` remain for a closer examination.

Language. The language similarity can be mapped to a binary metric, in the simplest case. If the two languages to compare are identical s_{ij} will take the value 1, otherwise 0. Of course, one could think of more sophisticated approaches, that for instance take linguistic aspects into account. In that case, the underlying metric would be expanded to a scalar value.

Input Type: . . . . . . Language

Input Values: . . . . . Language description

Metric: . . . . $d(O_i, O_j) = \begin{cases} 0, \text{ if } O_i = O_j \\ 1, \text{ else} \end{cases}$

Axioms satisfied: . . 1, 2, 3, 4

Metric category: . . . Metric

Similarity: . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f = identity$

Uniform Resource Locator (URL). A URL [W3C97] can be logically subdivided into three parts: the protocol, the hostname and the path. For the purpose of document retrieval the protocol has no relevance. By contrast, the hostname and the path reveal information about the location of a document. Providing a URL similarity enables one to compare in quantity the location of two documents.

`http://`	`graphics.ethz.ch/`	`papers/spr00.pdf`
Protocol	Hostname	Path

The hostname (= hostaddress) contains information about the physical location of a document. It is structured in a hierarchical manner, where dots separate the single address parts. Starting from the end with the most important element, the document's top level domain name (in our example "ch"), the importance decreases to the left. Usually, the top level domain will be followed by a second level domain name ("ethz") and finally the name referring to the physical machine ("graphics").

Regarding this structure the obvious approach to compute a similarity value between two hostnames would be to split each one into a set of tokens. Those will be compared one-to-one right to left, with the token position being taken into consideration as a weight. This weight is supposed to be larger the closer the token's positions are located to the right. We propose the following metric for hostnames:

$$d^{hostname}(O_i, O_j) = \frac{\sum_{p=0}^{n} id(token_{n-p}^{hostname_i}, token_{n-p}^{hostname_j}) \cdot w(p)}{\sum_{p=0}^{n} w(p)} \tag{3.11}$$

where

$$id(u, v) = \begin{cases} 0, & \text{if } u = v \\ 1, & \text{otherwise} \end{cases}$$

$$w(p) = (hostname\_weightfactor)^p$$

and n is the number of hostname tokens.

The parameter $hostname\_weightfactor$ reflects how heavily the hierarchy is weighed. Valid values lie in the interval $[0\ldots1]$. Choosing a value of 0, we get the degenerated case that everything except the tail end of the hostname – the top level domain – would be omitted. On the other hand, if we set $hostname\_weightfactor$ to 1 all tokens will have an equal influence. With foresight to the use of the metric as a base for the similarity measurement the value range of Equation 3.11 is already normalized to $[0\ldots1]$ by dividing through the sum over all weight factors. For two identical hostnames we receive a metric distance of 0.

The path comes with its own metric definition, even if its general structure is very similar to the one for hostnames, it differs in the hierarchy weighting. For the path the hierarchical structure is weighted from the left to the right.

$$d^{path}(O_i, O_j) = \frac{\sum_{p=0}^{m} id(token_p^{path_i}, token_p^{path_j}) \cdot w(p)}{\sum_{p=0}^{m} w(p)} \tag{3.12}$$

where

$$w(p) = (path\_weightfactor)^p$$

and m is the number of path tokens. For the function id the definition from Equation 3.11 is still valid.

Analogous to the $hostname\_weightfactor$ we may influence the weighting of the path hierarchy with a parameter $path\_weightfactor$. The path metric function is normalized to the co-domain $[0\ldots1]$ as well. Note, the token comparison proceeds from left to right, which is exactly the reversed order compared to the hostname metric.

As a formula for the complete URL metric we apply the *Weighted Sum Construction* introduced in Section 3.3. Consequently,

$$d(O_i, O_j) = w^{hostname} \cdot d^{hostname}(O_i, O_j) + w^{path} \cdot d^{path}(O_i, O_j) \tag{3.13}$$

where

$$w^{hostname} + w^{path} = 1 \text{ and } w^{hostname}, w^{path} \in [0\ldots1].$$

The two additional weights $w^{hostname}$ and w^{path} allow to steer the influence of the hostname and the path component respectively. In practice, this has a certain relevance with respect to the technique of host mirroring.

Input Type: . . . . . . Document Location

Input Values: . . . . . URL

Metric: . . . . $d(O_i, O_j) = w^{hostname} \cdot d^{hostname}(O_i, O_j) \quad , \\ + w^{path} \cdot d^{path}(O_i, O_j)$

$w^{hostname}, w^{path} \in [0 \ldots 1]$ and $w^{hostname} + w^{path} = 1$

Axioms satisfied: . . 1, 2, 3, 4

Metric category: . . . Metric

Similarity: . . . . $s_{ij} = 1 - f(d(O_i, O_j))$ with $f = identity$

4

PHYSICS-BASED MODELS

The following chapter establishes the mathematics for the *physics-based models* which constitute the basics for the applied visualization paradigms. Particularly, it lays the foundations for the *force directed layout mechanisms* for weighted graph-structures which will be explained in more detail in Section 5.2.3 on page 72. Therefore, this chapter is laid out more generously in its circumference. However, since basic research in physics-based models is not a part of the core areas of this project – we consider ourselves more as users of those models – the following sections are limited to introduce the mathematical foundations required to understand the basics of the presented approaches. Specifically, we consider the different aspects of various numerical simulations for *mass-spring systems*, which from a general point of view belong to the more generic class of *N-body problems*. Additionally, this chapter includes the presentation of a new spherical initialization approach for this kind simulation models.

An extensive theoretical treatise of these algorithms is beyond the scope of this work. There exists a wide range of information sources available that descend from various application areas such as particle physics, chemistry and astronomy. Accordingly, for a comprehensive introduction concerning physics-based models we refer to the excellent works of [Wit96, WBK95 and GT88]. Furthermore, additional in-depth information can be found in [Gar94, HE88] and with emphasis on numerical methods we recommend [HNW91, HW96, Sch93, Dem96a, Dem96b, RG87, Gre90 and PFT+88].

In analogy to [BET+98], when examining physics-based models we can distinguish two main parts on principle:

1. The *model* itself: a force system primarily defined by mass points and a set of connecting springs. More complex systems may arise with supplementary type of forces (e.g. electrical or gravitational forces).

2. The *numerical method*: this is a technique for finding an equilibrium state of the force system, that is, a position for each mass object, such that the total force on every mass is zero.

The subsequent sections are divided according to the two main ingredients of this approach. The first half of the chapter presents the chosen *model* and covers the basics of the dynamics for mass-spring systems. The remainder discussess the problems of *numerical methods*, with an emphasis on their numerical stability and the requirements of interactive simulation.

4.1 BASIC MASS-SPRING SYSTEMS

We briefly review the principles of the dynamics of mass-spring and particle systems. Mass-spring systems belong to the group of linear finite element methods and have been widely used in computer graphics because they provide a simple means of generating physically realistic motion for a wide range of interests [WBK95]. In particular, they serve to simulate the motion of complex physical bodies approximately, by splitting the mass into a collection of single mass points, which are connected by a set of springs. Furthermore, mass-spring systems are also applied to abstract structures, such as graphs, equipping them with adjustable physical behaviors.

A basic mass-spring system consists of a number of *mass objects*, in the following referred as *particles*. In an idealized mass-spring system, a particle P_i is an object that only has mass m_i, position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$, and responds to external forces $\mathbf{F}^{ij}_{spring}$, but that exhibits no spatial extent. A spring is represented by an object that connects two particles P_i and P_j, and that is defined by a zero length l_{ij} and a spring constant k_{ij} (see Figure 4.1). Additionally, a spring is considered to have zero mass. Without distorting the results of non-degenerated problems, these assumptions are primarily made in order to simplify the simulation of such systems.

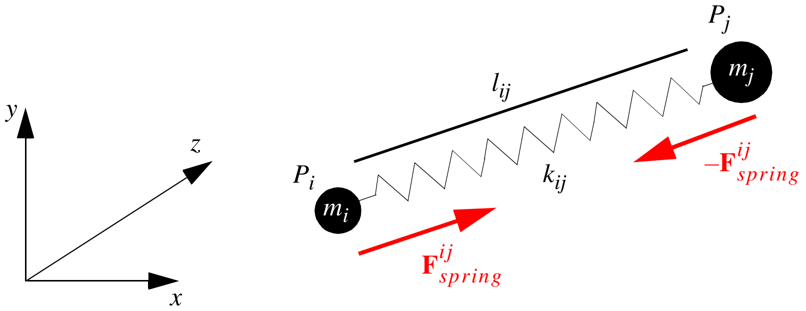


Figure 4.1 A basic mass-spring system, illustrating the forces between two mass objects m_i and m_j.

Applying Hook's law, the principal equation that governs the attractive force $\mathbf{F}^{ij}_{spring}$ between two mass points m_i and m_j is given by

$$\mathbf{F}^{ij}_{spring} = k_{ij} \cdot (\mathbf{x}_j - \mathbf{x}_i)\left(1 - \frac{l_{ij}}{|\mathbf{x}_j - \mathbf{x}_i|}\right) \tag{4.1}$$

where

k_{ij} . . . . spring stiffness

l_{ij} . . . . zero length of connecting spring

$\mathbf{x}_i$, $\mathbf{x}_j$ . . . . . . spatial position of particle P_i, respectively P_j

The total force $\mathbf{F}^{i}_{tot}$ on a particle P_i results from the superposition of all forces $\mathbf{F}^{ij}_{spring}$ due to all incident springs connected to P_i. We also add a velocity-dependent friction $\mathbf{F}^{i}_{frict}$ to the system in order to prevent undesirable harmonic oscillation [HE88]. Hence, the resulting equation for the total force is

$$\mathbf{F}^{i}_{tot} = \sum_{j \in A_i} \mathbf{F}^{ij}_{spring} + \mathbf{F}^{i}_{frict} = \sum_{j \in A_i} k_{ij} \cdot (\mathbf{x}_j - \mathbf{x}_i)\left(1 - \frac{l_{ij}}{|\mathbf{x}_j - \mathbf{x}_i|}\right) - f \cdot \mathbf{v}_i \tag{4.2}$$

where

$\mathbf{F}^{ij}_{spring}$ ..... force on P_i generated by the spring connection to P_j

$\mathbf{F}^{i}_{frict}$ ...... friction force acting on P_j

A_i .... set of indices j of particles P_j directly connected to P_i

f .... friction coefficient

$\mathbf{v}_i$ .... velocity of particle P_i

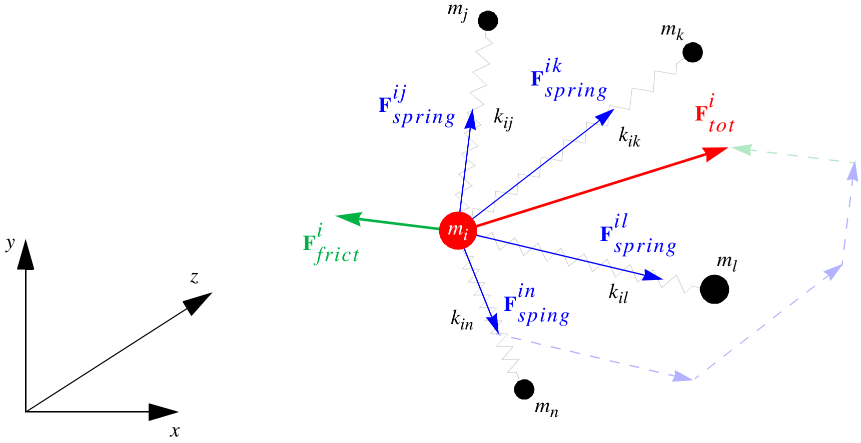


Figure 4.2 Total force $\mathbf{F}^{i}_{tot}$ on a particle P_i resulting from the set of superposed spring forces $\mathbf{F}^{ij}_{spring}$ and the cushioning friction force $\mathbf{F}^{i}_{frict}$.

4.2 ADVANCED FORCE MODELS

With respect to our application domain we extend the concept of a basic mass-spring system by adding further force types. Naturally, we keep the *spring* and the *friction forces* as parts of the system. Additionally, we model repelling *electrical forces.* All particles are charged with equal polarity to prevent an accidental and as such unwanted coincidence of their positions. The electrical force on a particle P_i exerted from another particle P_j follows the inverse square law and can be described as

$$\mathbf{F}^{i}_{elec} = \kappa \cdot \frac{q_i \cdot q_j}{|\mathbf{x}_i - \mathbf{x}_j|^2} \cdot \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|} \tag{4.3}$$

where

κ .... constant defined by $\kappa = \dfrac{1}{4\pi\varepsilon_0}$, where ε_0 is the *dielectrical constant.*

q_i, q_j .... charge applied to particle P_i, respectively P_j

$\mathbf{x}_i$, $\mathbf{x}_j$ .... spatial position of particle P_i, respectively P_j

As a second extension we introduce *gravity* to the model. The gravity paradigm we apply in this case is special with respect to its interaction behavior. In contrast to the electrical forces gravity does not interact between individual particles, although they might have a mass larger than zero. We rather think of a large mass point M positioned at the origin of the spatial coordinate system. Each particle P_i is exposed to the gravitational force $\mathbf{F}^i_{grav}$ resulting from its own mass m_i and the mass object M only. Consequently, the gravity force always points towards the origin, which gives this force a desired centering characteristic. Newton's first law, the law of gravitation, defines the force according to the following equation

$$\mathbf{F}^i_{grav} = -\gamma \cdot \frac{M \cdot m_i}{|\mathbf{x}_i|^2} \cdot \frac{\mathbf{x}_i}{|\mathbf{x}_i|} \tag{4.4}$$

where

γ .... gravitational constant

m_i .... mass of particle P_i

M .... mass of fictive object located at the origin

$\mathbf{x}_i$ .... spatial position of particle P_i

Recapitulating, we propose an extended mass-spring system, which includes the following list of forces that have an effect on each individual particle P_i:

Force Type	**Equation**
Spring force	$\mathbf{F}^{ij}_{spring} = k_{ij} \cdot (\mathbf{x}_j - \mathbf{x}_i)\left(1 - \dfrac{l_{ij}}{\lvert\mathbf{x}_j - \mathbf{x}_i\rvert}\right)$
Friction	$\mathbf{F}^i_{frict} = -f \cdot \mathbf{v}_i$
Electrical force	$\mathbf{F}^{ij}_{elec} = \kappa \cdot \dfrac{q_i \cdot q_j}{\lvert\mathbf{x}_i - \mathbf{x}_j\rvert^2} \cdot \dfrac{\mathbf{x}_i - \mathbf{x}_j}{\lvert\mathbf{x}_i - \mathbf{x}_j\rvert}$
Gravitation	$\mathbf{F}^i_{grav} = -\gamma \cdot \dfrac{M \cdot m_i}{\lvert\mathbf{x}_i\rvert^2} \cdot \dfrac{\mathbf{x}_i}{\lvert\mathbf{x}_i\rvert}$

Considering this extended model we obtain the total force on a particle P_i

$$\mathbf{F}^i_{tot} = \sum_{j \in A_i} \mathbf{F}^{ij}_{spring} + \mathbf{F}^i_{frict} + \sum_{i \neq j} \mathbf{F}^{ij}_{elec} + \mathbf{F}^i_{grav} \tag{4.5}$$

or expanded to full length

$$\begin{aligned} \mathbf{F}^i_{tot} &= \sum_{j \in A_i} k_{ij} \cdot (\mathbf{x}_j - \mathbf{x}_i)\left(1 - \frac{l_{ij}}{|\mathbf{x}_j - \mathbf{x}_i|}\right) - f \cdot \mathbf{v}_i \\ &+ \sum_{i \neq j} \kappa \cdot \frac{q_i \cdot q_j}{|\mathbf{x}_i - \mathbf{x}_j|^2} \cdot \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|} - \gamma \cdot \frac{M \cdot m_i}{|\mathbf{x}_i|^2} \cdot \frac{\mathbf{x}_i}{|\mathbf{x}_i|} \end{aligned} \tag{4.6}$$

k_{ij}	spring stiffness
l_{ij}	zero length of connecting spring
A_i	set of indices j of particles P_j directly connected to P_i
f	friction coefficient
$\mathbf{v}_i$	current velocity of particle P_i
κ	constant defined by $\kappa = \frac{1}{4\pi\varepsilon_0}$, where ε_0 is the *dielectrical constant.*
q_i, q_j	charge of particle P_i, respectively P_j
γ	gravitational constant
m_i	mass of particle P_i
M	mass of fictive object located at the origin
$\mathbf{x}_i$, $\mathbf{x}_j$	spatial position of particle P_i, respectively P_j

In a more verbose form Equation 4.6 may be interpreted as follows:

- The first term, the spring force component, is responsible to push a particle into a certain distance to another one, where the pursued distance corresponds to the value of the spring zero length l_{ij}. The endeavour of the spring to reach l_{ij} is proportional to the spring constant k_{ij}.
- The value of the friction component is independent of the position and therefore it obviously does not have a direct influence on the particle placement. Friction essentially improves the system stability by suppressing oscillation.

- The term for the electrical force makes sure that two particles located coincidentally close will be pushed away from each other. Because this force follows the inverse square law, it's influence diminishes rapidly. Therefore it is restricted to the case where the distance between two particles is very small. The repulsion can be controlled by adjusting the values for the charges q_i and q_j.
- Finally, the gravity term tends to pull the particles towards the origin, which has the effect of a final particle arrangement centered around the origin. In most of the cases this force is very weak. To augment its impact the mass M has to be increased.

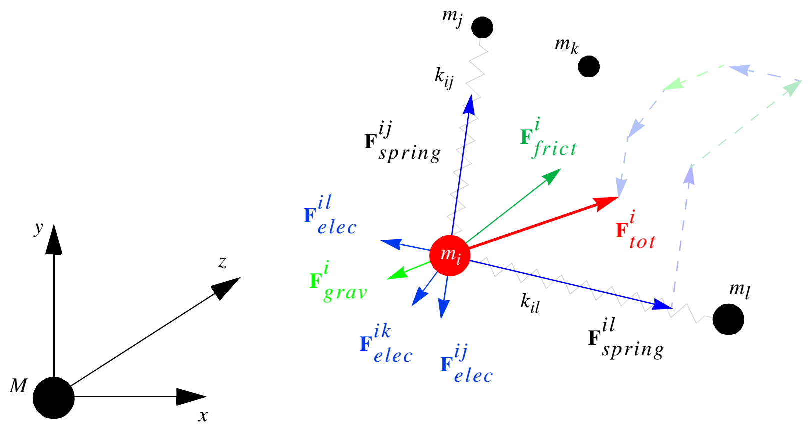


Figure 4.3 Total force $\mathbf{F}^{i}_{tot}$ on a particle P_i resulting from the superposition of spring, friction, electrical and gravitational forces.

4.3 DIFFERENTIAL EQUATIONS

If we consider the motion of a particle P_i by applying Newton's second law, Equation 4.6 converts into the familiar second-order linear *ordinary differential equation* (ODE) [Wit96] of the form

$$\mathbf{F}^{i}_{tot} = m_i \cdot \mathbf{a}_i = m_i \cdot \frac{d^2\mathbf{x}_i}{dt} = m_i \cdot \ddot{\mathbf{x}}_i \tag{4.7}$$

which could also be written as

$$\ddot{\mathbf{x}}_i = f(t, \mathbf{x}(t), \dot{\mathbf{x}}(t)) = \frac{\mathbf{F}^{i}_{tot}}{m_i}. \tag{4.8}$$

In order to determine the spatial motion – including velocity and the current position – of a particle P_i, a variety of numerical integration methods such as Euler or Runge-Kutta [Sch93, HNW91] can be used. Most of these methods are only applicable to first-order ODEs. But our problem, shown in Equation 4.8, is of second-order type. However, we can always convert a second-order equation to a system of first-order ODEs by introducing auxiliary variables. Here we introduce the additional variable $\mathbf{v}_i$ – representing the velocity – which results in the following system of coupled first-order ODEs.

$$\begin{bmatrix} \dot{\mathbf{x}}_i = \mathbf{v}_i \\ \dot{\mathbf{v}}_i = \dfrac{\mathbf{F}^i_{tot}}{m_i} \end{bmatrix} \tag{4.9}$$

This splitting technique entails a two step integration scheme. After we have computed the total forces $\mathbf{F}^i_{tot}$ and the resulting accelerations, we first integrate the accelerations over time to compute the velocities $\mathbf{v}_i$ and then integrate the velocities $\mathbf{v}_i$ to compute the positions $\mathbf{x}_i$ of the particles.

Because both equations in this system of Equation 4.9 can be integrated concurrently, we can assemble them to form a 6-dimensional vector. The space spanned by these vectors is also called the *phase space* [Wit96]. The component-wise formulation[1] of the phase space equation of motion is

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{v}_1 \\ \dot{v}_2 \\ \dot{v}_3 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ F_{tot_1}/m \\ F_{tot_2}/m \\ F_{tot_3}/m \end{bmatrix}. \tag{4.10}$$

Since the force $\mathbf{F}^i_{tot}$ is exclusively a function of $\mathbf{x}_i$ and the time t, Equation 4.10 now behaves like an ordinary scalar first-order ODE, with the only difference that the involved functions are of vector type. This circumstance implies that the standard numerical integration methods are directly applicable to Equation 4.10. For a further discussion of numerical integration methods we refer to Section 4.6 on page 55.

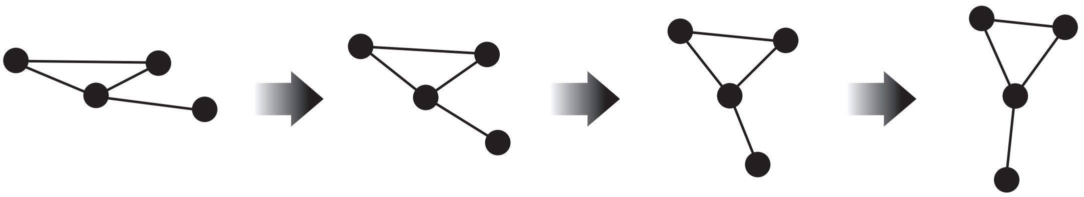


Figure 4.4 A simple mass-spring system converging to its equilibrium state. The relaxation sequence starts on the left and progresses to the right.

When we consider a whole system of n particles, it is described analogously by n copies of Equation 4.10, concatenated to an $6n$-long vector. Figure 4.4 shows a sequence of snapshots illustrating the calculated movements of a small mass-spring system consisting of 4 particles connected as shown. The initial arrangement is shown in the leftmost image. This defines the ODE's start values. From there the numerical simulation runs until the system converges into an equilibrium state, where all particle positions remain stable.

---

[1] Please note that for an improved presentation we have dropped index i for the notation of Equation 4.10.

4.4 INITIAL PARTICLE POSITIONING

In this section we discuss possible strategies concerning the initial arrangement of particles. Since, our simulation problem results in an ODE that belongs to the class of *initial value problems*, this procedure is crucial to the subsequent relaxation process and thus, to the resulting equilibrium state. In addition, we have to be aware that our model corresponds to a high dimensional optimization problem [YR91, Ben96] and is prone to all the known problems associated with subspace procedures [GK95].

Typically, physics-based models and their corresponding differential equations emerge from fields that attempt to model the physical behaviour of real systems. Such fields are particle physics, chemistry and astronomy. All those systems have in common, that their initial state is determined by their real world equivalents. However, since our field of application deals with abstract data this correspondence is nonexisting. Consequently and in order to prevent the system from convergence to a local minimum, a skilful preconditioning paradigm should be applied.

Therefore, we propose a two-step initialization strategy, where tightly connected particles should be positioned as close as possible to each other [SGE+97]. The first step comprises the determination of n positions in space. The second step assigns each particle P_i to one of the n positions determined previously. It is clear, that we won't succeed in the general case, but the method reported below avoids most initial ill-conditioning.

4.4.1 Determination of Initial Positions

Given is a set of particles $P_1, \ldots, P_n$. One may think of several different approaches to detremine n initial positions in practice. The following subsections categorize and desribe them by their individual strategy of chosing the positions.

Point positioning. The *point positioning* presumably marks the most simple strategy. Following this strategy, one single position in space (e.g. the origin of the current coordinate system) is determined only, which will be valid for all particles. This approach leads to a simulation process where all particles will diverge in a explosive manner from the point of initilization to find their equilibrium in a certain distance to each other. Actually, this idea will not work in practice! The proposed initialization denotes a singular point, where we get undefined expressions in the underlying ODEs. In order to avoid this undesirable effect, we may ease the initialization constraint by choosing the positions not at a single point but rather around a single point. However, such a simulation will still show very large forces which may result in numerical instability.

Random positioning. Another simple and still rather pragmatic approach is the *random positioning*. Following this strategy the n initial positions will be determined randomly in space, whereas an initialization parameter controls the maximal dilataion of the scattering. In many cases this method will turn out satisfactory. However, it suffers from the main disadvantage that the simulations initialized in such a way are not reliably reproducable. A fact, that makes this method unusable for our purposes.

Shape embedded positioning. The *shape embedded positioning* approach directly addresses the drawback of the random strategy by binding the initial positions to a well defined surface. Thereby, the arrangement of the initial positions, so called *patches*, follows the best possible evenly distribution within the given geometrical constraints. With this embedding approach the initialization process becomes well defined and reproducible.

A implementation of this strategy is the *paradigm of spherical embedding* also presented in [SGE+97]. Since this strategy emerges as a concrete result from our work and was for most of the examples it will be described in more detail hereafter.

The basic idea is to determine the n initial positions equally spaced on the surface of a sphere with radius R as illustrated in Figure 4.5a. The center of the sphere is located at the origin of the used coordinate system. Such an arrangement yields a satisfying approximation for an intrinsically aimed equidistant particle distribution in space, which would be optimal from a theoretical point of view. The major advantage of this arrangement is the degree of symmetry inherent to the geometry of a sphere. It fairly supports the attempt that non of the spacial dimensions is supposed to be favored.

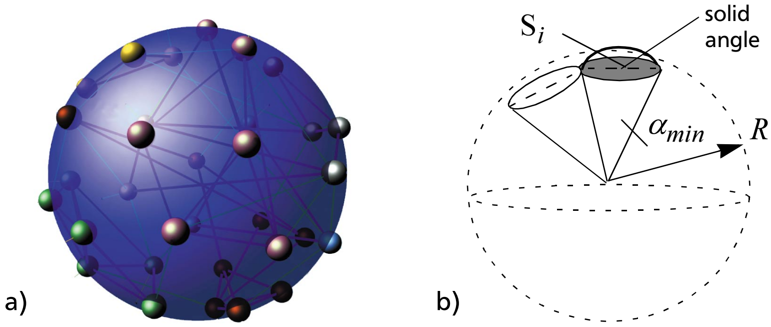


Figure 4.5 a) Initialization of particles on a virtual sphere. b) Poisson disc sampling for initial positioning

Computing equally spaced positions on a sphere results in a nontrivial optimization task, the so-called *Fekete-problem* [PSS97], which however is approximated efficiently by a *Poisson disc sampling* procedure in spherical coordinates [Gla95]. Thereby, the positions are determined interatively by successively allocating *solid angle* element S_i as depicted in Figure 4.5b. The center of such an element S_i denotes the position to initially place a particle. In doing so, the angle between the position vectors of two particles will be at least as large as a fixed anlge α_{min}. In other words, the distance of two particles must not undershoot a distance threshold d_{min}, computed by the following relation:

$$d_{min} = 2R \cdot f_d \cdot \sin\left(\frac{\alpha_{min}}{2}\right) = 4R \cdot \sqrt[4]{1 - \frac{f_d}{n}} \cdot \sqrt{\left(1 - \sqrt{1 - \frac{f_d}{n}}\right)} \tag{4.11}$$

where

d_{min}	distance threshold
α_{min}	minimal angle between two positions
f_d	distribution factor $\in [0 \ldots 1]$
R	radius of the spherical surface
n	number of particles

For the purpose of a simpler formulation Equation 4.11 was developed under the assumption, that the sum of solid angle elements complete exactly to a whole sphere. Since this is actually not true, we scale the element's surface with a factor $f_d \in [0 \ldots 1]$ to take this into account. As a concequence, this factor has a direct influence on the uniformity of the distribution of the initial positions is illustrated in Figure 4.6.

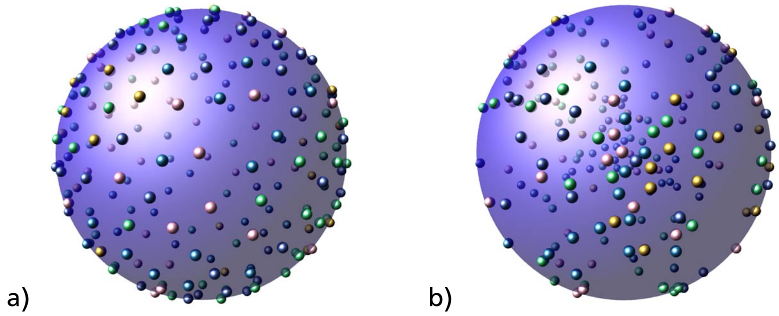


Figure 4.6 Poisson disc distribution for: a) $f_d = 1.0$, b) $f_d = 0.3$

4.4.2 Assign particles to initial positions

Now that the initial positions have been determined, a second step will cover the assignment of particles $P_1, \ldots, P_n$ to these n positions. Thereby, each particle will be assigned to exactely one position whereas each position does accept no more than one particle. This assignment may again occur by means of different strategies, discussed hereinafter.

Random assigment. Again at this level, a straightforward *random assignment* strategy could be employed. However, two grave disadvantages makes this assigment method in practice of no avail. Firstly the assignments found by using this strategy are not reproducible due to the random characteristic of the procedure itself. Secondly, the algorithm consequently ignores the given physics-founded connectivity between particles. Thus, it is potentially possible that this proceeding results in a disadvantageous particle assignment. This is for example the case if strongly connected particles are placed far from each other.

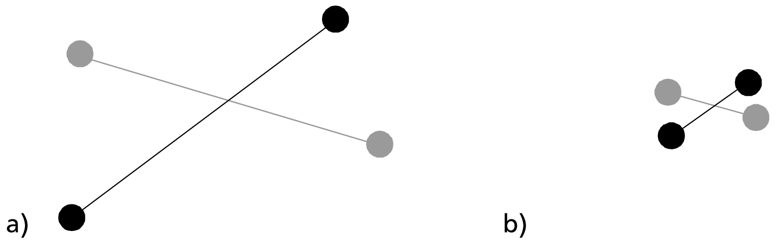


Figure 4.7 Unfavourable initialization with crossing connections. a) Initial layout b) Arrangement after relaxation, undesirably clustering of unconnected particles

Especially with regard to our concrete usecase the initialization strategy has to be aware of the spring forces in our system, which mainly determine the resulting particle layout. It should be avoided that weakly connected or even unconnected particles will be spatially neighboring after relaxation simulation. This may for example be the case if connections in short distance come to lie crosswise to each other initially (see Figure 4.7a). If this is the case, an undesirable cluster of particles will result near the connection's intersection point as shown in Figure 4.7b. In terms of the underlying spring connectivity, this cluster arises by chance and without a corresponding equivalence in the underlying connection topology. The presens of such “fake” clusters ihibit the desired interference of the resulting spacial arrangement onto the subjacent connectivity.

Breadth-first-traversal assigment. The *breadth-first-traversal assigment* startegy is a promising approach that takes the existing connections into consideration by placing strongly connected particles at neighboring positions. For that the particles and connections are considered as a special instance of a weighted graph, where not the edges but the vertices are weighted.

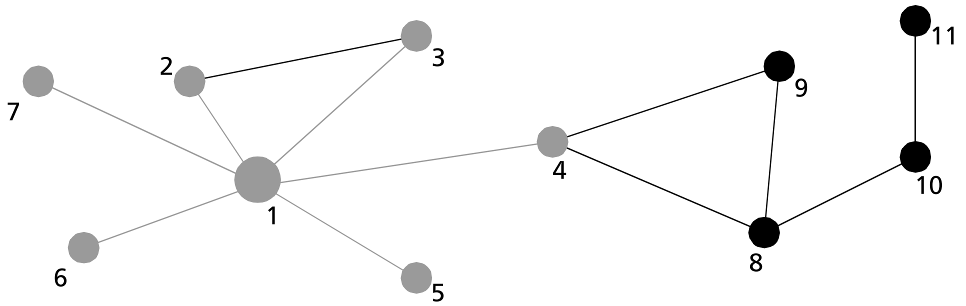


Figure 4.8 Assignment order of breadth-first-traversal assignment

We start with an initial weighting of all particles P_i, where the weight w_i is defined by the sum of spring constants k_{ij} of all adjacent springs.

$$w_i = \sum_{j \in A_i} k_{ij} \tag{4.12}$$

where

w_i .... weight of particle P_i

k_{ij} .... spring constant of spring connecting particle P_i and P_j

A_i .... set of indices j of particles P_j directly adjacent to P_i

This weighting emphasizes the importance of particles with strong connections. From there, a list $P = \{P_1, \ldots, P_n\}$ of all objects is built which is sorted with respect to the weights w_i. The assignment of particles to the computed surface positions is figured out by a *breadth-first-traversal* strategy. The algorithm starts from the particle with the largest w_i and assigns it to a first position. The positions are now ranked according to their Euclidian distance from the initial particle in 3D space. In the next step, all directly connected particles are visited and assigned in the order of their weights. From there, the procedure traverses the list recursively until all objects are assigned.

Figure 4.8 illustrates the procedure. The numbers shown indicate the sequence order in which the particles are traversed. The particles drawn in gray are those already visited by the time the recursion is applied the first time. A pseudocode fragment for the method is given below in Table 4.1.

Table 4.1 Pseudocode listing breadth-first-traversal position assigment

```
$P=\{P_1,...,P_n\}$   // initial particle list //
$S=\{S_1,...,S_n\}$   // initial positions //
$T=\{\}$              // list of sublists //

while (P not empty) do
  fetch particle $P_i$ from P | $w_i$ = $w_{max}$
  assign any position $S_k$ to $P_i$: $P_i$ -> $S_k$
  remove $S_k$ from S
  generate list $L_i$ of all directly
  connected particles $P_l \in P$ sorted with respect to $k_{il}$
  keep $w_i$ with $L_i$
  add $L_i$ into T

  while (T not empty) do
    fetch sublist $L_m$ from T | $w_m$ = $w_{max}$
    remove all particles $\in L_m$ from P

    while ($L_m$ not empty) do
      fetch next particle $P_j$ from $L_m$
      assign free position $S_k$ to $P_j$ | dist($S_k$,$S_m$)=min: $P_j$ -> $S_k$
      remove $S_k$ from S
      generate list $L_i$ of all directly
      connected particles $P_l \in P$ sorted with respect to $k_{lj}$
      keep $w_j$ with $L_j$
      add $L_j$ into T
    od
  od
od
```

Depth-first-traversal assignment. This strategy has been developed as an alernative to the *breadth-first-traversal assignment* procedure described above. It shares the basic concept, but applies a different type of traversal strategy.

The algorithm starts from the particle with the largest w_i and assigns it to a position. Subsequently, the directly connected particle holding the largest w_i is traversed, then a direct neighbor of that one and so forth. This process is continued recursive until a perticle is reached that does either not have any further connections or only such too already attended particles. Figure 4.9 illustrates the traversal algorithm. The numbers beside besides each particle mark the order in which the graph is traversed. Particles that have already been visited when the first recursion run is finished are shown in gray.

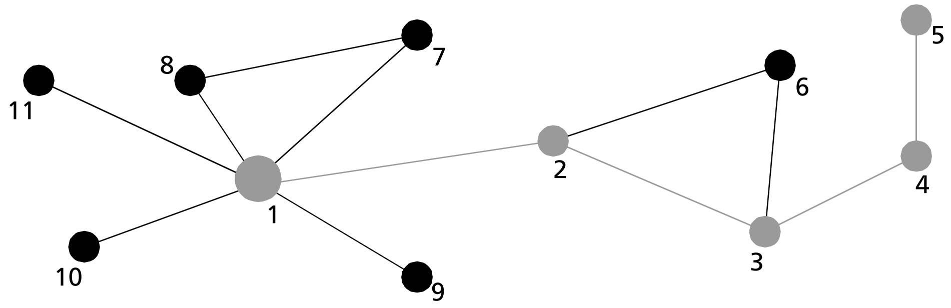


Figure 4.9 Assignment order of depth-first-traversal assignment

Table 4.2 describes the assignment algorithm in the form of a short sequence of pseudocode.

Table 4.2 Pseudocode listing depth-first-traversal position assigment

```
P={$P_1$,...,$P_n$}    // initial particle list //
S={$S_1$,...,$S_n$}    // initial positions //
T={}                   // list of sublists //

while (P not empty) do
  fetch particle $P_i$ from P | $w_i$ = $w_{max}$
  assign any position $S_k$ to $P_i$: $P_i$ -> $S_k$
  remove $S_k$ from S
  generate list $L_i$ of all directly
  connected particles $P_l \in$ P sorted with respect to $k_{lj}$
  insert elements of $L_i$ into T

  while (T not empty) do
      assign value of index k to m

      fetch first particle $P_j$ from T
      remove $P_j$ from P

      assign free position $S_k$ to $P_j$ | dist($S_k$,$S_m$)=min: $P_j$ -> $S_k$
      remove $S_k$ from S
      generate list $L_i$ of all directly
      connected particles $P_l \in$ P sorted with respect to $k_{lj}$
      insert elements of $L_j$ at the beginning of T
    od
  od
od
```

4.5 FORCE CALCULATION SCHEMES

Current literature describes many different particle simulation models, however most of them differ primarily in the type of the used force calculation scheme. Because of their actual $O(N^2)$ characteristic force calculations denote a crutial part for an efficient particle simulation. Especially for typical models – consisting of a large number of particles – these costs assume hardly controllable proportions. Therefore, literature knows a wide range of optimizing calculation schemes in order to handle this issue.

A comprehensive recapitulation can be found in [Gra96, HE88]. Table 4.3 shows a compilation of such methods – including references in case the reader desires further information. Because not all of them are relevant in the context of our work and some of them are just hybrids of two or more basic ones, we restrict ourselves to discuss only four selected methods: the particle-particle model (PP), the particle-mesh model (PM), the top down tree-code (TDTC) and the fast multipole method (FMM). The PP model uses a distance formulation of the force law, the PM model treats the force as a field quantity by approximating it on a mesh, the TDTC model optimizes force calculations by using hierarchical structures and a superior force decomposition and the FMM model can be regarded as a hybrid of the PM and the TDTC models, which is achieved by using a particle-mesh technique, or by using a tree code.

The PM approach will allow us to model a system in which the physics is determined by the interaction of a small number of particles only. PM techniques are most effective when the particle density distribution is relatively uniform. Because of its dynamic clustering technique, tree codes (e.g. TDTC) are favored in systems with large density contrast.

Table 4.3 List of force calculation methods (methods discussed in this section are shown in bold)

Force Calculation Method	References
Particle-Particle (PP)	**[GT88], [HE88]**
Particle-Mesh (PM)	**[HE88]**
Particle-Particle/Particle-Mesh (P3M)	[HE88]
Particle Multiple-Mesh (PM2)	[GCW97]
Nested Grid Particle-Mesh (NGPM)	[SR96]
Top Down Tree-Code (TDTC)	**[Dem96a, BH86]**
Bottom Up Tree-Code (BUTC)	[Pre86], [JP89]
Fast-Multipole-Method (FMM)	**[Dem96b, RG87, Gre90]**
Tree-Code Particle Mesh (TPM)	[Xu96]
Self-Consistent Field (SCF)	[HO92]
Symplectic Method	[CS90]

The following subsections will intorduce a set of calculation methods, which have a certain relevance with respect to our work. Each subsection closes with a short discussion about the main advantages and drawbacks of the presented method.

4.5.1 The Particle-Particle Method (PP)

The PP method (also known as *direct summation method*) is conceptually the simplest of all particle simulation methods. The state of the system at some time t is defined by the set of particle positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$. The main loop updates these values with respect to the present force contributions and equations of motion (See “Differential Equations” on page 40.) to obtain the new state of the system a discrete time step Δt later.

The temporal evolution of the particle system is achieved by a repeated application of the time step loop, as listed in Table 4.4.

The PP direct integration approach is trivial in its elements, but has high computational costs. Due to the straightforward force accumulation approach, $O(N^2)$ operations are required per time step to evaluate the resulting forces on all N particles. Typically, up to thousand time steps are needed to reach a stable particle distribution (equilibrium), so if forces are long-ranged the PP method is feasible only up to a few hundred particles. However, for close-range dynamics of particles the number of operations depends only on the sum of neighbours added over all particles P_j, which are close enough to a certain P_i in order to significantly contribute to the accumulated forces $\mathbf{F}_i$. The same consideration can be made for plain mass-spring systems, where the number of significant neighbours is equal to the number of directly spring-connected particles. Thus, the operations count corresponds to the sum of spring connections over all particles P_i:

$$Count_{ops} = \sum_{i = 1 \ldots N} Count_{neigbors}(P_i)$$

Table 4.4 Time step loop of particle-particle method (PP)

```
1. Accumulate forces by finding force $F_{ij}$ of particle $P_j$ on $P_i$

   // Reset all force accumulators //
   for i = 1 to N do
     $F_i$ = 0
   od

   // Accumulate forces //
   for i = 1 to N-1 do
     for j = i+1 to N do
       $F_i$ = $F_i$ + $F_{ij}$
       $F_j$ = $F_j$ - $F_{ij}$
     od
   od

2. Integrate equations of motion (e.g. Euler's method)

   for i = 1 to N-1 do
     $v_i^{new}$ = $v_i^{old}$ + $F_i/m_i$*dt
     $x_i$ = $x_i$ + $v_i^{old}$*dt
   od

3. Update time counter

   t = t+dt
```

It follows that, if we have a favorable particle distribution or a sparsely connected mass-spring system the $O(N^2)$ dependence of the operations count can be dropped and the PP method becomes applicable for larger systems with up to $N = 1000$ particles.

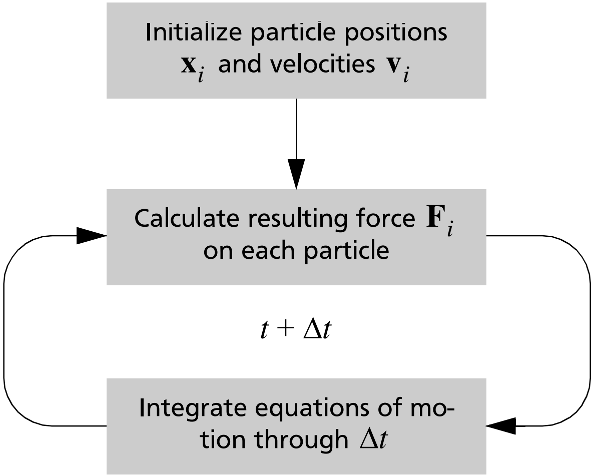

Initialize particle positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$
Calculate resulting force $\mathbf{F}_i$ on each particle
$t + \Delta t$
Integrate equations of motion through Δt

Figure 4.10 Flowchart of particle-particle code (PP)

Since this restriction in the number of particles is much too low for most simulation problems, more efficient techniques have been developed. However, these more elaborated techniques often tend to be less accurate than the PP method.

4.5.2 The Particle-Mesh Method (PM)

The PM method is one of the oldest improvements over PP, introduced somewhere around 1985 by R.W. Hockney & J.W. Eastwood [HE88]. It handles the force as a field quantity by approximating it on a regular array of mesh points. This could be either a 1D, 2D or 3D mesh, depending on the type of simulation. Thus, it not only uses a discretization of time, but also a discretization of space. Differential operators, such as the Laplacian ∇^2, are replaced by finite difference approximations on the mesh.

Each particle mass is assigned to one or more nearest mesh points. There are many different strategies how this assignment – an interpolation process – could be done [HE88]. So, basically mass is not considered to be located on points other than the mesh points. Based on this discrete mass distribution we could define a density function $\rho(\mathbf{x})$.

According to the *Poisson equation*

$$\nabla^2\phi = -\rho(\mathbf{x}) \tag{4.13}$$

where

ϕ .... potential

$\rho(\mathbf{x})$ .... density function

we can calculate the potential ϕ on every meshpoint only from the density function $\rho(\mathbf{x})$. This can be done using the *Fast Fourier Transforms* (FFT). If the potential ϕ is known at every meshpoint, we can easily calculate the force field $\mathbf{F}'_{uv}$ in every meshpoint (u, v) by taking the gradient of the potential (Equation 4.14).

$$\mathbf{F}'_{uv} = -\nabla\phi \tag{4.14}$$

where

$\mathbf{F}'_{uv}$ .... force field in a specific meshpoint (u, v)

ϕ .... potential

The force on a particle is obtained by using some sort of interpolating technique on the array of mesh-defined values. For several reasons this interpolation technique corresponds almost always to the one used for the mass assignment to the mesh.

So, the principle time step loop of the PM method differs from that of the PP only in the issue of the force calculation and looks as follows:

Steps (1a) and (1d) have operation counts proportional to the number of particles $O(N)$. The operation counts for step (1b) scales as $O(N_g \cdot \ln N_g)$ when using the fast Fourier transform (FFT) technique, where N_g is the number of mesh points. At last, the operations for step (1c) are proportional to $O(N_g)$. Hence, if we consolidate the cost over all these steps we can see that the number of operations for a complete time step scales as

$$O(\alpha N + \beta(N_g \cdot \ln N_g) + \gamma N_g).$$

where

N .... number of particles

N_g .... number of mesh points

α, β and γ ... constants

Table 4.5 Time step loop of particle-mesh method (PM)

```
1a. Assign density values ro(x,y) to the mesh

1b. Solve field potential equation (Poisson's) on the mesh

1c. Calculate forces $F'_{uv}$ from the mesh-defined potential
    involving Equation 4.14.

1d. Interpolate forces $F_i$ at particle positions $x_i$

 2. Integrate equations of motion (e.g. Euler's method)

      for i = 1 to N-1 do
        $v_i^{new} = v_i^{old} + F_i/m_i*dt$
        $x_i = x_i + v_i^{old}*dt$
      od

 3. Update time counter

      t = t+dt
```

The result is a much faster, but in general less accurate force calculation than we would obtain using the previously introduced PP method. More precisely, concerning the PM method the interaction between close particles – where close means being at a shorter distance than the meshspacing – is only a very rough approximation and so it is important that the total force on a particle is mainly determined by the most distant particles (so-called *uncorrelated systems*) if we want this method to be accurate.

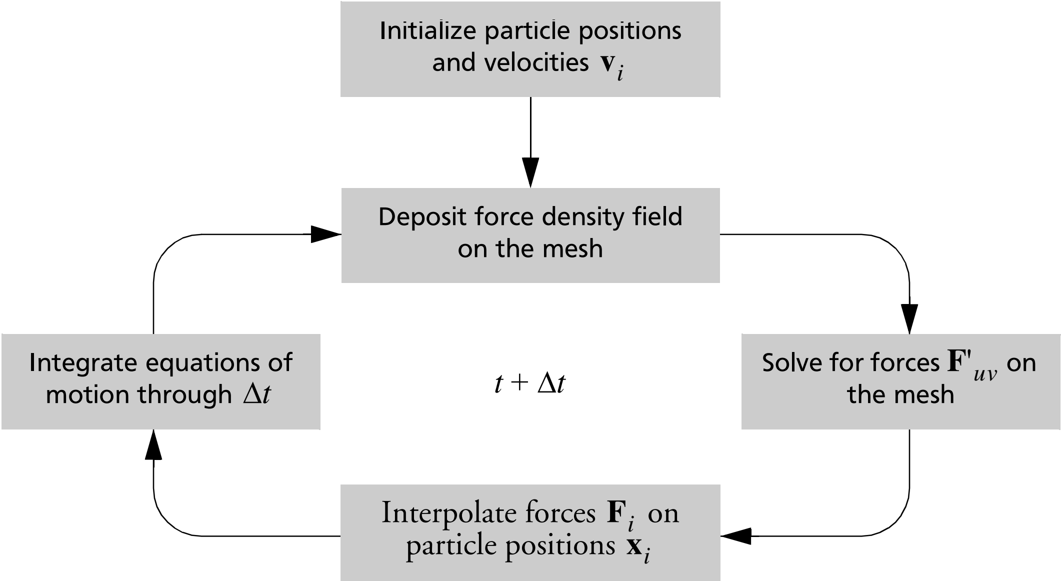

Initialize particle positions and velocities $\mathbf{v}_i$
Deposit force density field on the mesh
Solve for forces $\mathbf{F}'_{uv}$ on the mesh
$t + \Delta t$
Integrate equations of motion through Δt
Interpolate forces $\mathbf{F}_i$ on particle positions $\mathbf{x}_i$

Figure 4.11 Flowchart of particle-mesh code (PM)

Nevertheless, if the opposite is the case, where close neighbours contribute primarily to the force on a particle (so-called *correlated systems*) a very fine mesh resolution is required to achieve even modestly accurate individual particle trajectories. Consequently, the PM method gets expensive in terms of memory as well as computational costs. A more efficient approach, namely the *tree code method*, tailored to this type of systems will be discussed in the following subsection.

Unfortunately, the PM method bears further restrictions. PM is limited to the handling of forces which are actually representable by a field, such as electrical or gravitational forces. Consequently, only forces with a globally valid characteristic are acceptable, which excludes all systems that involve spring-forces, which have an undesired two-body connection pattern.

4.5.3 The Top Down Tree Code Method (TDTC)

Another improvement over the PP are the TDCT algorithms, with their main representative being the so-called Barnes-Hut method (BHM) – which will be discussed in the following. It was first introduced in 1986 by J. Barnes and P. Hut [BH86] and is based on an earlier algorithm of A. Appel published in 1985 [App85]. It addresses the problem of the expensive far-field force calculations by substituting a large group of far away particles with a single virtual particle with the total mass of all particles and with its position in the center of mass of the distant group.

The algorithm is based on a data structure called *oct-tree* in 3D (or *quad-tree* in 2D), which partitions space and groups particles that are close to each other. The tree structure for a given input distribution is obtained starting from a root cell, which is large enough to hold all particles during he whole simulation. Then we recursively split cells in each direction into two smaller parts of equal size, until each cell contains one single particle only.

In addition, the BHM introduces a superior approach for the per particle P_i force dealing with the subsequent force superposition concept.

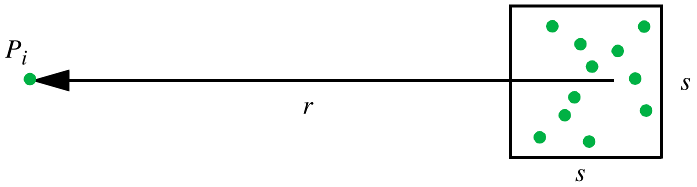
$$\mathbf{F}^i_{tot} = \mathbf{F}^i_{external} + \mathbf{F}^i_{nearest-neighbors} + \mathbf{F}^i_{far-field} \tag{4.15}$$

Consequently, the resulting force $\mathbf{F}^i_{tot}$ gets decomposed into three independent force components, where external forces $\mathbf{F}^i_{external}$ such as friction can be computed for each particle independently. Further, the forces caused by the nearest neighbours $\mathbf{F}^i_{nearest-neighbors}$ only require interaction with just those particles. Computationally much more expensive is the evaluation of the far field forces $\mathbf{F}^i_{far-field}$, like electrostatic forces, because for each particle they depend on the parameters of all other particles

$$\mathbf{F}^i_{far-field} = \sum_{i \neq j} \mathbf{F}_{ij}$$

where $\mathbf{F}_{ij}$ is the force of particle P_j on Particle P_i.

We can conclude that the computational cost for the external and nearest neighbour forces seem to rise with order $O(N)$ only. However, the effort for the far field forces grows with $O(N^2)$. The BHM addresses exactly this problem with a divide-and-conquer strategy following simple physical intuition. Particles are grouped into clusters by an adaptive oct-tree data structure. The idea is to approximate the force exerted on a particle by a sufficiently distant cluster of particles, by substituting the singe particles and computing an interaction between the particle and the cluster's center of mass instead.



$$\frac{s}{r} = \frac{\text{side length of oct-tree cell containing cluster}}{\text{distance from center of mass to particle}} \leq \delta$$

A common condition for a cluster being "sufficiently far away" from a particle is, that the ratio of the side length of the oct-tree cell s containing the cluster to the distance r between particle P_i and cluster is smaller than some threshold value δ. The approximation error can be controlled by choosing a suitable value for δ. A smaller δ usually improves the simulation precision, but increases the computational effort.

This technique achieves a cost reduction from $O(N^2)$ to $O(N \cdot \log(N))$ in the uniform case. Since the BHM operates without grid and has no preferred geometry, it is generally more accurate than the PM method. Especially, because its space subdivision works adaptively based on the current particle distribution the algorithm performs excellently for highly non-uniform distributions.

The basic steps of one iteration of the BHM are as follows:

Table 4.6 Time step loop of top down tree-code method (TDTC/BHM)

```
1a. Build (or update) the oct-tree data structure, by
    recursively subdividing the computational domain into
    smaller and smaller cells.

1b. For each cell compute the center of mass (or some
    higher-order approximation) and the total mass of the
    particles it contains.

1c. For each particle, perform a traversal of the oct-tree in
    order to compute the force exerted on the body. The
    traversal starts at the root. At each visited node, if the
    cluster defined by all bodies inside this node is
    sufficiently far away, compute the interaction $F^i_{far-field}$
    with its center of mass. Otherwise continue with visiting
    all its child nodes.

1d. Based on Equation 4.15 compute the resulting forces $F^i_{tot}$

 2. Integrate equations of motion (e.g. Euler's method)

      for i = 1 to N-1 do
        $v_i^{new} = v_i^{old} + F^i_{tot}/m_i*dt$
        $x_i = x_i + v_i^{old}*dt$
      od

 3. Update time counter

      t = t+dt
```

The main disadvantage is that BHM, like other tree-based methods, requires a large amount of auxiliary memory to store additional information (center of mass, total mass per cell) for each single cell on each tree-level.

4.5.4 The Fast Multipole Method (FMM)

The FMM was first published 1985 by V. Rokhlin and L. Greengard [Rok85, RG87 and Gre90]. The algorithm is based on a tree code that uses two representations of the potential field. The two representations are: The *far field (multipole)* and the *local expansion*. The two representations are referred to as the *duality principle.* With its subdivision scheme and the field-based force computation, the FMM combines the characteristics of the PM and the TDTC methods.

The strategy of the FMM is to compute a compact expression for the *potential* ϕ (similar to PM), which can be easily evaluated along with its derivative, at any point $\mathbf{x}$. The FMM achieves this by evaluating the potential ϕ as a *multipole expansion*, which can be viewed as a kind of Taylor expansion, which is accurate when $|\mathbf{x}|$ is large. For optimization reasons concerning the computational costs, the multipole expansion is subdivided into two auxiliary expansions: an *outer expansion* and an *inner expansion.* The outer expansion computes the potential outside a certain cell, due to the particles inside a cell. In contrast, the inner expansion, computes the potential inside a cell, due to the particles outside. The core of the FMM a is clever algorithm which converts an outer expansion to an inner expansion in a constant amount of time for any cell.

Since, the computation of an outer expansion is reasonable and the conversion to an inner expansion has been highly optimized, it can be shown that it is computationally less expensive to compute the potential ϕ indirectly by evaluating the outer expansion first and then run the conversion step subsequently. This approach contrasts a direct evaluation of the multipole expansion, which would result in clearly higher computational costs.

A more accurate explanation of the multipole expansion is beyond the scope of this subsection. At this point we refer to [RG87] for further information and we now present a high-level listing of the algorithm's time step loop:

As we can see, the algorithm shares the divide-and-conquer and the oct-tree paradigm with BHM, but differs in the following points:

- FMM computes an expression for the potential ϕ at each point $\mathbf{x}$, not the force as does BHM.
- FMM stores more information than only the mass and center of particles per oct-tree cell. This more complicated expansion determines a more accurate, but also more expensive approach.
- FMM applies a fixed oct-tree subdivision to compute the potential, rather than a set varying with the parameter δ and location of the center of mass.

In contrast to the BHM, the FMM is capable to achieve full machine precision, where the computational complexity of the FMM is lower by an order of magnitude. Since we are traversing the tree in step (1b), (1c) and (1d), and doing a constant amount of work per node, the total work is proportional to the number of nodes. The number of nodes is in turn proportional to the number of particles, so these steps have computational costs in the order of $O(N)$. Step (1e) has obviously order $O(N)$ since we visit each particle once to compute the resulting force vector. Step (1a), building the oct-tree, has order $O(N \cdot b)$

Table 4.7 Time step loop of fast multipole method (FMM)

```
1a. Build the oct-tree containing all the points.

1b. Traverse the oct-tree from bottom to top, computing the
    outer expansion for each cell in the tree.

1c. Traverse the oct-tree from top to bottom, computing the
    inner expansion for each cell in the tree.

1d. For each leaf, add the potential contributions of nearest
    neighbours and particles in the leaf to the result of the
    inner expansion computed in the previous step.

1e. Compute $F^i_{tot}$ exploiting the simple coherence between the
    potential and the force shown in Equation 4.14.

 2. Integrate equations of motion (e.g. Euler's method)

      for i = 1 to N-1 do
        $v_i^{new} = v_i^{old} + F^i_{tot}/m_i*dt$
        $x_i = x_i + v_i^{old}*dt$
      od

 3. Update time counter

      t = t+dt
```

supposed the number of oct-tree levels is set to a fixed value b. Thus, the FMM's overall costs amount to the order of $O(N)$ [Ess92].

The FMM is very popular in particle physics and astronomical simulations. However, for our purpose the FMM lacks the possibility to consider spring-forces in the particle model, a restriction inherited from the affinity to the PM method. The latter similarly accelerates the force computations by evaluating a potential representation instead.

4.6 NUMERICAL TIME INTEGRATION METHODS

The particles in our simulated systems move according to the second-order ODE shown in Equation 4.7 and later on specified as a system of first-order ODEs (see Equation 4.9). Due to the forces involved and the form $\mathbf{F}^i_{tot}$ takes, the equation systems encountered in the context of our application are very difficult to solve analytically. Thus, numerical methods must be used to obtain solutions approximating the system changes over time accurately. Since, we are especially interested in the animation of particles' trajectories iterative approaches are preferred.

With this, we reach the second of two core parts of a physics-based model, as mentioned in the introduction of this chapter. We will discuss different approaches of *numerical time integration methods* with a special emphasis on the actual approximation algorithm, to its stability and efficiency. The information contained in this subsection is based on the excellent contributions of [HE88, HNW91, HW96, HWB95, Sch93, WBK95 and Wit96].

For simplicity, we focus our discussion to the problem of a single first-order ODE (analogous to Equation 4.10). However, generality is preserved since higher-order ODEs can always be transformed into a system of first-order ODEs. As we will see, most of the common numerical integration schemes are for this reason limited to process only on the latter. So, using a generalized formulation we have an equation

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t)), \tag{4.16}$$

where $\mathbf{x}$ depends on time t and the function f expresses the right hand side of the equation. In addition an initial condition

$$\mathbf{x}(t^0) = \mathbf{x}^0 \tag{4.17}$$

is given. Besides, we take it for granted that the requirements for the existence of an unambiguous solution are met and will not touch on these questions. For a further study of this point we refer to [HNW91].

The particular class of this type of problems is called *initial value problems*. In contrast to *boundary value problems*, in which values for $\mathbf{x}$ are given at two end points, these problems are defined by a single start value $\mathbf{x}^0$ at a certain position t^0. Starting from this position, we take a time-step h by involving the derivative function f to compute an approximate change in $\mathbf{x}$, also referred as $\Delta\mathbf{x}$. Before calculating the next time step we add $\Delta\mathbf{x}$ to $\mathbf{x}$ in order to obtain the new value. In numerical methods the derivative function f usually is encapsulated and viewed as a black box, where we provide the numerical arguments and receive the corresponding numerical value for $\dot{\mathbf{x}}(t)$ in return. Numerical integration methods operate by performing one or more instances of these derivative evaluations at each time step.

In general, this means that in the step from the mathematical description of physical laws of particle systems to their numerical simulation the continuous ODEs (Equation 4.10) are substituted by linear algebraic relationships. Continuos functions $\mathbf{x}$ and f are replaced by values at discrete time intervals. Since, the problem of numerical integration of ODEs appears in many different domains (astrophysics, chemistry, classical mechanics, etc.), there are a wide variety of integration schemes which approximate continuos first-order ODEs by discrete analogs. We can unite those schemes under the generalized linear multistep equation:

$$\sum_{i=0}^{n} a_i \mathbf{x}^{k+i} = \Delta t \cdot \sum_{i=0}^{n} b_i \cdot f(t^{k+i}, \mathbf{x}^{k+i}) \tag{4.18}$$

Further on, we can classify discrete equations of this form depending on the value of b_n.

Classification	Criterion
Explicit Scheme	$b_n = 0$
Implicit Scheme	$b_n \neq 0$

If b_n is zero, the scheme is called *explicit* and we may solve it for $\mathbf{x}^k$ directly in terms of known quantities. In contrast, if b_n is nonzero, the scheme is called *implicit* and we must utilize iterative approaches to find $\mathbf{x}^k$, unless $\mathbf{F}^i_{tot}$ is simple.

In the following, we briefly summarize the most commonly used integration schemes, which are also implemented in our framework, which is described later on in Chapter 6. Thereafter, a more detailed discussion will focus more soffisticated variations, with a special focus to our field of application. In particular, we take a look at an integration algorithm compounding the standard schema with a specially customized variable step size strategy. We will also examplify a modified Runge-Kutta method especially optimized to work directly on second order ODEs.

4.6.1 Runge-Kutta Methods

Runge-Kutta methods belong to the group of one-step techniques and can either be of explicit or implicit nature. The *Euler's method* denotes the most simple representative of a Runge-Kutta method. It is termed a first-order integration method because the integration error over a finite interval scales with h^1.

The method is straight-forward, easy to understand and to implement. Particularly, it serves as a good illustration to explain the priciples of Runge-Kutta methods. Table 4.8 shows the pseudo-code for the Euler's update algorithm.

Table 4.8 The Euler's update algorithm

```
1. Choose a fix step size h, then

     $t^k = t^0 + kh,$                $k = 0, 1, 2, \ldots$

2. Computation rule to get approximated solutions

     $\mathbf{x}^0 = \textit{initial value}$

     $\mathbf{x}^{k+1} = \mathbf{x}^k + h \cdot f(t^k, \mathbf{x}^k)\,,$      $k = 0, 1, 2, \ldots$                (4.19)
```

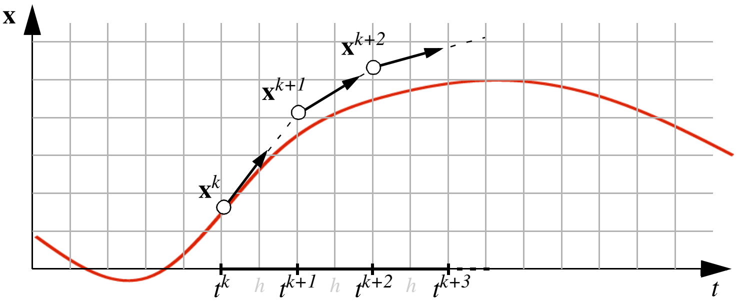


Figure 4.12 Euler's method uses the derivative at time t^k to compute an approximative function value $\mathbf{x}^{k+1}$ for the next time step t^{k+1}. Here shown by means of a one dimensional scalar example.

Although Euler's method can produce good function approximation in theory, it is not used in practice because of its lack of accuracy and numerical stability. We may show this by means of two small examples [WBK95]. Consider the case of a 2D function f whose integral curves are concentric circles, like it is shown in Figure 4.13a. Starting from an arbitrary point $\mathbf{x}^0$ governed by f we are supposed to orbit forever on whichever circle it started on. Instead, according to Euler's method, with each step we will move on a straight line to a circle of larger radius, so that the covered path follows an outward spiral. Notice, that shrinking the step size doesn't cure the problem, it only decreases the grade of the outward spiral. On a second example we show a potential unstable behavior of this method. Consider a 1D function $f = -kx$, for which the solution should decay from any starting point x^0 to zero. For step sizes $h \leq 1/k$ we get reasonable results, but for larger step sizes the solution starts oscillating around zero and even diverges if $h \geq 2/k$, which means that the system blows up.

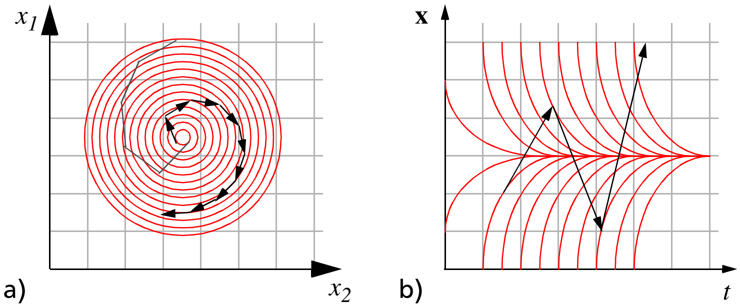


Figure 4.13 Two small examples illustrating a possible (a) inaccurate and (b) instable behavior of the Euler's method.

Classical Runge-Kutta Approach. There is a large number of derivative methods that essentially base on the Euler's integration schema. Virtually all of them face the problem of instability by including additional Euler steps into one time step, which results in higher order methods. The most popular of this kind is the *4-stage* or *classical Runge-Kutta method of fourth order*. However, we abandon a further discussion at this point of the classical Runge-Kutta approaches. Instead of we refer to the comprehensive literature already mentioned in the introduction of this chapter.

Adaptive Runge-Kutta Approach. A general problem of all the numerical integration methods presented up to this point is the appropriate selection of the step size h. Supposed the forces get temporarily large, and consequently so do the velocities, particles will travel too far and the error can get unacceptably large or still worse, the system may get unstable. On the other hand, if the chosen step size is small enough to handle the preceding scenario, it might need a vast number of steps to integrate over the entire time-interval, even if the entire rest of the simulation is well-behaved. Thus, if we choose a fixed step size – which we obviously want to determine as large as possible – we can only proceed as fast as the "worst" section of $\mathbf{x}(t)$ in $[t^k, t^{k+1}]$ will allow.

As an improvement, we think of a method which adaptively controls h as we march forward in time. Starting from a predefined value for h the algorithm increases h whenever the *local discretization error* falls below a certain threshold value. By contrast, the step size h will be automatically cut down when the error gets inadmissible large. In essence, this summarizes the idea of adaptive step sizing: varying h over the course of solving the ODE.

Adaptive with Maximum Error Criterion. The step size control algorithm proposed in the previous section is not suitable in any case. It belongs to a class of step size altering methods, that tend to avail the accuracy of the simulation only. That is because the variation of the step size is driven by the local discretization error. There are problems, particularly in computer graphics, which demand other step size adjustment criterions. For example, for many of the animation simulations, computational speed is much more important than meticulous numerical accuracy. In fact, we are interested in a visually pleasing result which is usually met with a stable solution. So, the applied step size adjustment should be driven rather by stability preserving factors than accuracy. Hence, we look for a method which selects the step size in each step the biggest possible, always at the boundary of loosing stability. The trick is to recognize instability just before it gets visible on the screen.

A very promising approach based on a standard Runge-Kutta method is proposed in [BW98]. Although, the application area is quite different from ours, the fundamental goals are the same. Applied to our context, instability arises almost exclusively from strong spring forces and typically becomes apparent in the form of vastly fast diverging particles (comparable with an explosion). Therefore, after each explicit step, we treat the resulting position changes $\Delta \mathbf{x}_i$ as a proposed change for each particle P_i. If any of these values exceeds a prescribed threshold ϑ then we discard the proposed changes, reduce the current step size and try it again.

Admittedly this step size adjustment method gives quite a heuristic impression. Nevertheless, since this method is based on one of the established Runge-Kutta integration schemes, this method – though simple – has performed very well in practice.

Reduced Runge-Kutta optimized for Second Order ODEs. In [Col66] Zurmühl suggests an adapted version of the classical Runge-Kutta method, which is optimized with regard to second-order ODEs, such as Equation 4.8.

As mentioned in Section 4.3 for most of the numerical integration schemes those ODEs have to be split up into a system of first-order ODEs. Those may be arranged using a vector notation, which allows a simplified treatment as a single vector-based first-order ODE. Nevertheless, the resulting computational costs for second-order ODEs are approximately double compared to those emerging from their first-order instances.

This algorithm recombines the previously divided second-order ODE and processes the first and second derivative slotted into each other trying to reuse calculations in each case. Consequently, the algorithm saves operations and therefore cuts down computation time. The algorithm runs with a constant step size h and has the following structure:

Table 4.9 A reduced Runge-Kutta update algorithm optimized for second order differential equations

1. Choose a step size h, then

$$t^k = t^0 + kh, \qquad k = 0, 1, 2, \ldots \tag{4.20}$$

2. Computation rule to get approximated solutions

$$\mathbf{x}^0 = \textit{initial position value} \tag{4.21}$$

$$\mathbf{v}^0 = h \cdot \textit{initial velocity value}$$

$$k_1 = \frac{h^2}{2} f(t^k, \mathbf{x}^k, \mathbf{v}^k)$$

$$k_2 = \frac{h^2}{2} f\left(t^k + \frac{h}{2}, \mathbf{x}^k + \frac{1}{2}\mathbf{v}^k + \frac{1}{4}k_1, \frac{1}{h}(\mathbf{v}^k + k_1)\right)$$

$$k_3 = \frac{h^2}{2} f\left(t^k + \frac{h}{2}, \mathbf{x}^k + \frac{1}{2}\mathbf{v}^k + \frac{1}{4}k_1, \frac{1}{h}(\mathbf{v}^k + k_2)\right)$$

Table 4.9 A reduced Runge-Kutta update algorithm optimized for second order differential equations

$$k_4 = \frac{h^2}{2}f\left(t^k + h, \mathbf{x}^k + \mathbf{v}^k + k_3, \frac{1}{h}(\mathbf{v}^k + 2k_3)\right)$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{v}^k + \frac{1}{3}(k_1 + k_2 + k_3)$$

$$\mathbf{v}^{k+1} = \mathbf{v}^k + \frac{1}{3}(k_1 + 2k_2 + 2k_3 + k_4), \qquad k = 0, 1, 2, \ldots \tag{4.22}$$

Implicit Runge-Kutta. Implicit *implicit Runge-Kutta methods* have their natural role in the solution of so-called stiff problems, where strong stability is turned into a severe handicap for traditional explicit methods. The first methods were introduced around 1824 by Cauchy for the sake of error estimations [HNW91]. Cauchy inserted the mean value theorem into the explicit integral of

$$\mathbf{x}^{k+1} - \mathbf{x}^k = \int_{t^k}^{t^{k+1}} f(t, \mathbf{x}(t))dt \tag{4.23}$$

to obtain in a first step

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \int_{t^k}^{t^{k+1}} f(t, \mathbf{x}(t))dt \tag{4.24}$$

and finally

$$\mathbf{x}^{k+1} = \mathbf{x}^k + h \cdot f(t^k + \theta h, \mathbf{x}^k + \Theta(\mathbf{x}^{k+1} - \mathbf{x}^k)) \tag{4.25}$$

with $0 \le \theta, \Theta \le 1$. The two remarkable extreme cases are $\theta = \Theta = 0$, yielding

$$\mathbf{x}^{k+1} = \mathbf{x}^k + h \cdot f(t^k, \mathbf{x}^k), \tag{4.26}$$

which is already known to the reader as the *explicit Euler* method we discussed previously and $\theta = \Theta = 1$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + h \cdot f(t^k + h, \mathbf{x}^{k+1}), \tag{4.27}$$

which we call the *backward* or *implicit Euler's method.*

Table 4.10 The implicit Euler's update algorithm

1. Choose a step size h, then

$$t^k = t^0 + kh, \qquad k = 0, 1, 2, \ldots \tag{4.28}$$

2. Computation rule to get approximated solutions

$$\mathbf{x}^0 = \textit{initial value}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + h \cdot f(t^k + h, \mathbf{x}^{k+1}), \qquad k = 0, 1, 2, \ldots \tag{4.29}$$

4.6.2 Leap Frog Method

There is a group of numerical integration algorithms which can be applied directly to a system of coupled ODE's, but only a few respect the symmetric structure of the position and velocity calculations occurring in our problems. One of them that does is the *Leap Frog* integrator. The name of the algorithm comes from the fact that the velocities are evaluated at the mid-point of the position evaluation, the velocities *leap* by a half-step $h/2$ over the positions, and vice versa. Thus, position and velocity updates are computed in an interlaced manner.

Table 4.11 The Leap-Frog update algorithm

1. Choose a step size h, then

$$t^k = t^0 + kh, \qquad k = 0, 1, 2, \ldots \tag{4.30}$$

2. Computation rule to get approximated solutions

$$\mathbf{x}^0, \mathbf{v}^0 = \textit{initial values}$$

$$\mathbf{v}^{1/2} = \mathbf{v}^0 + \frac{h}{2} \cdot f(t^0, \mathbf{x}^0)$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + h \cdot \mathbf{v}^{k+1/2}$$

$$\mathbf{v}^{k+3/2} = \mathbf{v}^{k+1/2} + h \cdot f(t^{k+1}, \mathbf{x}^{k+1}), \qquad k = 0, 1, 2, \ldots \tag{4.31}$$

The advantage of this method is that the velocities are explicitly calculated giving an additional handle for controlling purposes. On the other hand, the disadvantage is that they are not calculated at the same time. So, a linear approximation is required to extract the instantaneous velocities at an integer time-steps t^k:

$$\mathbf{v}^k = \frac{1}{2}(\mathbf{v}^{k-1/2} + \mathbf{v}^{k+1/2}) \tag{4.32}$$

After all, we accomplished an integration scheme which "jump-starts" from an offset in velocity by $O(h^2)$ from the specified initial value $\mathbf{v}^0$. Additionally, similar errors are made in extracting $\mathbf{v}^k$ at later times.

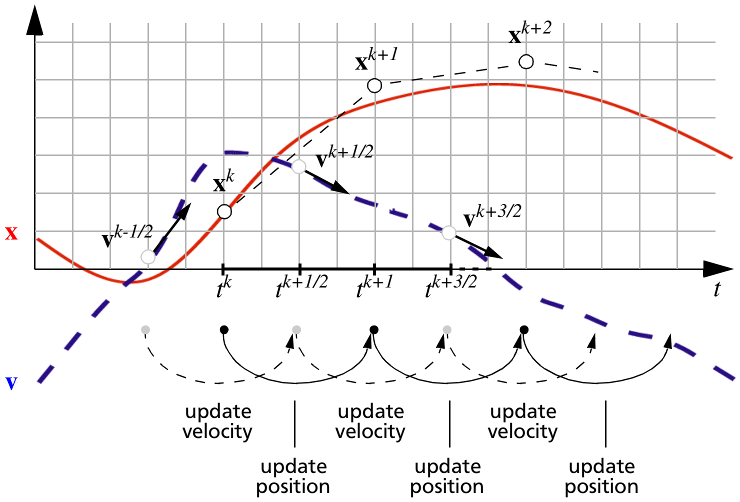


Figure 4.14 Leap Frog integration scheme. Positions at time t^k are updated using velocities at time $t^{k+1/2}$, velocities at $t^{k+1/2}$ are updated using forces at time t^{k+1}.

$$\mathbf{v}^{1/2} = \mathbf{v}^0 + \frac{h}{2} \cdot f(t^0, \mathbf{x}^0) + O(h^2) \tag{4.33}$$

4.7 STIFF PROBLEMS AND NUMERICAL STABILITY

While the intuitive meaning of *stiff* is clear to all specialists, a precise formal description is hard to define. One of the first and also one of the most pragmatic opinion goes back to 1952 (Curtiss & Hirschfelder): *Stiff equations are equations where certain implicit methods, in particular BDF*[1]*, perform better, usually tremendously better, than explicit ones*. Certainly the eigenvalues of the Jacobian $\partial f / \partial x$ have some relevance, but also quantities such as the dimension of the system, the smoothness of the solution or the integration interval are important.

ODE systems which describe physical, biological or chemical processes tend to produce resulting functions composed of parts that exponentially fade away in strongly varying manner. In the context of this work, this may happen if a particle system consists of a mixed configuration of very strong and very weak springs.

Choosing a method for a numerical approximation of such ODE systems one should be well conscious of the specific characteristics of the given system and the potentially arising solution functions. Ignoring these aspects may lead to completely inaccurate approximate solutions, evidently having nothing in common with the exact result.

Such a behavior is called *instability* and occurs mainly together with *stiff differential equations*. However, talking of “stiff differential equations” – strictly speaking an equation is not stiff. A particular initial value problem for that equation may be stiff, in some regions. The size and location of these regions depend on the initial value, the chosen numerical integration method and the error tolerance.

---

[1] BDF: Methods based on differentiation [HW96]

The mathematical theories which establish a connection between stiff problems and numerical stability help us to distinguish between different types of instability, choose an adequate numerical approximation method and adjust its parameters.

4.7.1 Stability Analysis for Runge-Kutta Methods

Though we mentioned in the introduction of this section that stiff systems call for implicit methods, we may still apply explicit methods to stiff problems under certain conditions. In order to get more specific, we undertake a so-called *stability analysis* for the selected set of numerical methods presented in Section 4.6. As a result we will be able to determine domains of stability for each of them.

Definition. If we can express an integration step in the form of $x^{k+1} = R(z) \cdot x^k$, the function $R(z)$ is called the *stability function* of a numerical integration method. It can be interpreted as the numerical solution after one step for

$$\dot{x} = \lambda x, \qquad x^0 = 1, \qquad z = h\lambda, \tag{4.34}$$

the famous *Dahlquist test equation.* The set

$$S = \{z \in \mathbf{C};\ |R(z)| \le 1\} \tag{4.35}$$

is then called the *stability domain* of the method.

The step size h of a numerical method has to be chosen such that for $Re(\lambda) < 0$ the condition $h\lambda \in S$ is always valid. In the case of ODE systems we have to be aware of this circumstance, because the step size h has to be determined small enough particularly to ensure $h\lambda_i \in S$ for all λ_i. Otherwise the computed results are inaccurate, which means that the method becomes instable.

The following stability analysis procedures will be performed in analogy to the procedures presented in [HW96]. All reflections will be based on the scalar equation

$$\dot{x} = f(t, x). \tag{4.36}$$

Let $\varphi(x)$ be a smooth solution of Equation 4.36. We linearize f in its neighborhood as follows

$$\dot{x}(t) = f(t, \varphi(t)) + \frac{\partial f}{\partial x}(t, \varphi(t))(x(t) - \varphi(t)) + \dots .$$

Writing $\bar{x}(t)$ for $x(t) - \varphi(t)$ we obtain

$$\dot{\bar{x}}(t) = \frac{\partial f}{\partial x}(t, \varphi(t)) \cdot \bar{x}(t) + \dots = J(t) \cdot \bar{x}(t) + \dots$$

In the sense of a first approximation we consider the Jacobian $J(t)$ as constant and neglect the error terms. Omitting the bars we arrive at

$$\dot{x} = Jx. \tag{4.37}$$

We now may apply different numerical integration methods to Equation 4.37 to study their behavior and determinate their specific stability characteristics.

Euler's method. We apply Euler's method (Table 4.8) to Equation 4.37 and obtain

$$x^{k+1} = R(hJ) \cdot x^k \tag{4.38}$$

with

$$R(z) = 1 + z .$$

We suppose that J is diagonalizable with eigenvectors $v_1, \ldots, v_n$ and the corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$. We transform J to Jordan canonical form (see HNW91, Section I.12) and write x^0 in this basis as

$$x^0 = \sum_{i=1}^{n} \alpha_i v_i \tag{4.39}$$

Inserting this into Equation 4.38 we get

$$x^k = \sum_{i=1}^{n} (R(h\lambda_i))^k \alpha_i v_i \tag{4.40}$$

Obviously the value for x^k remains bounded for $k \to \infty$, if for all eigenvalues λ_i the complex number $z = h\lambda_i$ lies in the set

$$S = \{z \in \mathbf{C}; |R(z)| \le 1\}$$

which is the circle of radius 1 and centre (-1, 0) in the complex plain.

Midpoint Method. The midpoint method can be viewed as a low order Runge-Kutta method. Thus, its stability analysis will proceed comparably to the more general case, the classical Runge-Kutta method. That's why we present the midpoint's stability region without its derivation and refer for such to the more general case in next subsection.

Thus, because the midpoint method corresponds to a 2nd-order Runge-Kutta ($s = 2$), its stability domain is defined by the polynomial

$$R(z) = 1 + z + \frac{z^2}{2!} . \tag{4.41}$$

The shape of the corresponding stability domain is shown in Figure 4.15.

Classical Method. Now, we consider the classical Runge-Kutta method. Formulated in a more general way, this method applied to Equation 4.37 gives

$$x^{k+1} = x^k + hJ \sum_{i=1}^{s} b_j \cdot g_j \tag{4.42}$$

where

$$g_i = x^k + hJ \sum_{j=1}^{i-1} a_{ij} \cdot g_j .$$

Since the variable s in Equation 4.42 reflects the order of the method we focus on and the method is of 4th order, the value for s is defined by $s = 4$. Inserting g_j repeatedly into Equation 4.42 while applying the correct values a_{ij} and b_j that apply to the classical Runge-Kutta method, yields

$$x^{k+1} = R(hJ) \cdot x^k$$

where

$$R(z) = 1 + z \cdot \sum_j b_j + z^2 \cdot \sum_{j,k} b_j \cdot a_{jk} + z^3 \cdot \sum_{j,k,l} b_j \cdot a_{jk} \cdot a_{kl} + \ldots \tag{4.43}$$

is a polynomial of degree $\le s = 4$. As a consequence, the classical Runge-Kutta method with $p = s = 4$ exhibits the stability function

$$R(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!}. \tag{4.44}$$

The corresponding stability domain is represented as a part of Figure 4.15.

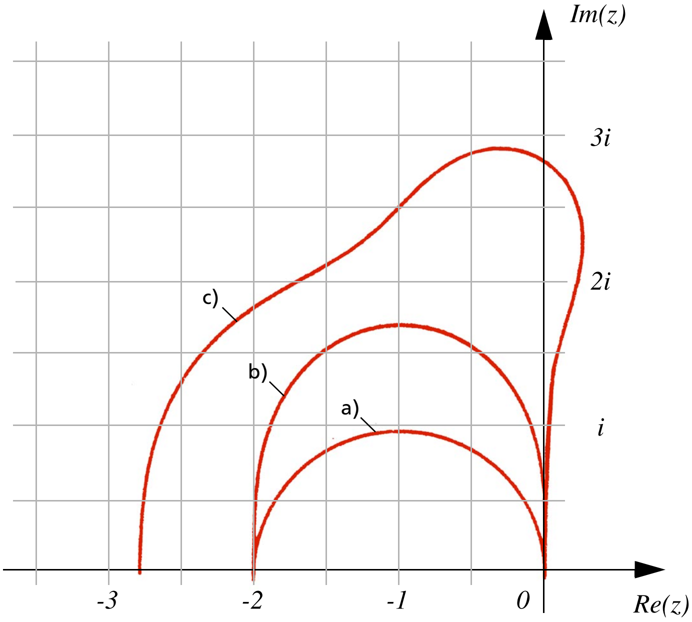


Figure 4.15 Domains of absolute stability plotted in complex λh-space for the set of presented numerical time integration methods a) Euler, b) Midpoint and c) Runge-Kutta. Note that the size of the regions grows with the order of the method.

4.8 IMPLICATIONS FOR INFORMATION VISUALIZATION

One of the crucial issues concerning the application of physics-based models in the context of graph layouting and information visualization are stable and reproducible simulation results.

So, which algorithms are suitable for the field of information visualization? In the context of our research work we applied the *classical Runge-Kutta* or the *Leap-Frog* method to virtually all of our examples. This can explained out of the following two reasons. Firstly, we were actually never faced with a mal-conditioned or stiff problem. Thus, there were no requirements to use an implicit approach. However, if there were indications of a stiff problem we could easily reconfigure the physical-system by involving modified similarity functions (see Section 3.4) or different mapping operators for the layout model (see Section 6.3). Secondly, all of the examples were mainly good-natured in the sense of there was no need for applying adaptive approaches in order to handle heavily varying dynamics. In addition, a rudimentary performance test has shown that for the size of problems we deal with there was no significant speed-up when using a higher-order, adaptive or implicit method.

Finally, we did some experiments with stochastic methods, such as [FLM94]. However, these approaches were not followed up in this work because of their considerable lack of reliability.

5

GRAPH VISUALIZATION

In contrast to other authors, we reject the commonly used term of *graph drawing* to title this chapter. Usually, *graph drawing* covers the problem of graph layouting as well as the actual drawing of a graph under one single term. Since visualization techniques have tremendously advanced lately, the actual part of drawing the graph has gained more importance. Originated by the visualization community, there are various convincing new approaches to represent such structures, including methods using 3D scenes and alternative geometries. Consequently, in our work we explicitly distinguish between two different tasks: the *graph layouting* and the *graph rendering*. Graph layouting deals with the problem of constructing geometric representations of graphs in space, whereas graph rendering covers the task of representing graphs visually exclusively. In order to keep a clean naming and to avoid possible misunderstandings we replace the original term *graph drawing* by *graph visualization*, which includes the layouting and rendering each as a separate subprocess.

The main purpose of this chapter is to introduce the basics of graph visualization. Section 5.1 provides the fundamentals and theory to understand subsequent sections, which treat the process of embedding similarity information in graphs. The subsequent Section 5.2 examines various graph layouting techniques, with a particular emphasis on algorithms which preserve the embedded information. Relating to our research we devote a separate subsection to the category of force-directed methods. Section 5.3 then presents progressive techniques to depict a previously layouted graph. Finally, Section 5.4 closes this chapter with an application-specific analysis of implementation and complexity issues, which results in the introduction of an new type of graph data structure, the *implicit adjacency matrix*.

5.1 GRAPH THEORY

The origin of graph theory probably goes back to a paper written by Leonhard Euler in 1736. It was titled *Solutio problematis ad geometriam situs pertinentis* [Eul36] and treated the question of whether or not it is possible to plan a walk over the seven bridges of

Königsberg in such a way that each of the bridges will be crossed only once. To solve the problem in a general manner Euler abstracted the bridges into *edges* and pieces of land into *nodes* of a *graph*. In this way, he was able to analytically determine the conditions which are required to permit such a walk.

In a more general way, managing this kind of connectivity information is fundamental. Today, connectivity information is present for example in geographic information systems (GIS), in transportations, in network routing tables or in parent-child relationships defined by hierarchical data-structures. Indeed, connectivity information can be defined by all kinds of relationship that exists between pairs of objects.

Applications of graphs being popular and diverse, researchers developed a common terminology to describe different components and properties in an abstract manner. In this section we introduce the most important terms with respect to our context. A more detailed treatment may be found in [BM76, Har72 and BET+98] from where most of the following presentation was compiled.

5.1.1 Basic Definitions

A *graph* $G = (V, E)$ consists of a finite set V of *vertices* and a finite multiset E of *edges*, that is, unordered pairs (u, v) of vertices, where $u, v \in V$. We call V the *vertex set* and E the *edge set* of G.

Thus, a graph is a way of representing connections or relationships between pairs of objects from some collection V. One should be aware that, depending on the context, the vertices of a graph are sometimes called *nodes* and edges may also be called *links*, *arcs* or *connections*.

Edges in a graph are either *directed* or *undirected*. An edge (u, v) is said to be *directed* from u to v if the pair (u, v) is ordered, with u preceeding v. An edge (u, v) is said to be *undirected* if the pair (u, v) is not ordered, that is, (u, v) is the same as (v, u). If all edges in a graph are undirected, then we say the graph is an *undirected graph*. Likewise, a *directed graph* or a *digraph*, is a graph whose edges are all directed. A graph that has both directed and undirected edges is often called a *mixed graph*.

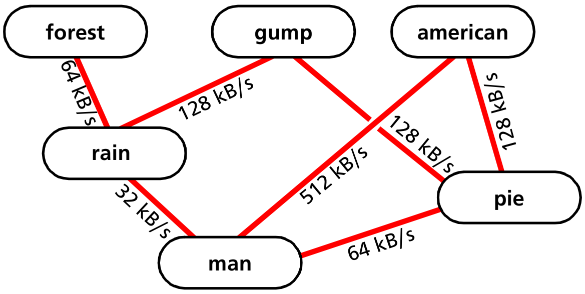


Figure 5.1 A simple undirected graph showing different computers and the network links connecting them. Connections are labeled with their respective bandwidth.

An edge (u, v) with $u = v$ is called a *self-loop*. The definition of a graph refers to the collection of edges as a multiset – not a strict set – thus allowing more than one edge to connect the same pair of vertices. Such edges are called *parallel edges* or *multiple edges*. A *simple graph* has no self-loops and no multiple edges.

A *path* in a graph $G = (V, E)$ is a sequence $(v_1, v_2, \ldots, v_h)$ of distinct vertices of G, such that $(v_i, v_{i+1}) \in E$ for $1 \le i \le h - 1$. A path is a *cycle* if $(v_h, v_1) \in E$.

The two vertices joined by an edge are called the *end vertices* or *endpoints* of the edge. Two vertices are said to be *adjacent* if they are endpoints of the same edge. An edge is said to be *incident* on a vertex if the vertex is one of the edge's endpoints. The *degree* of a vertex v is the number of incident edges of v. The degree of a vertex v always corresponds exactly to the number of vertices adjacent to v.

5.1.2 Weighted Graphs

In order to quantitatively express the strength of a relation between two objects, the features of a basic graph are insufficient. For example we might be using a graph to represent a network of computers, such as shown in Figure 5.1. Suppose our task is to find the fastest way to route a data packet from one computer to another. In order to solve this problem, the connectivity information provided by the network graph alone is insufficient. The network links need to be annotated with the connection speed. More abstractly, these (quantitative) annotations can be seen as a weight associated with each edge of the graph.

A *weighted graph* is a structure $G = (V, E, w)$, where (V, E) is a graph and w is a function mapping each edge e in E to its weight $w(e)$ in $\boldsymbol{R}$.

We are now able to define additional properties of a graph which allows a more accurate measurement of the relations represented by a graph. Among others, the computation of the shortest path between two vertices v, u belongs to this group.

Let $G = (V, E, w)$ be a weighted graph. We define the *length* (or weight) of a path P in G as the sum of the weights of the edges on this path. That is

$$w(P) = \sum_{i=1}^{h-1} w((v_i, v_{i+1})),$$

where h is the number of vertices of the path. We define the distance $d(u, v)$ from a vertex u to a vertex v in G as the length of the shortest path from u to v, if such a path exists. If there is no path at all from u to v we set $d(u, v) = +\infty$.

Suppose we are assigned the task to minimize the number of connections of the network shown Figure 5.1 without losing throughput. However, each computer still has to remain linked to the network. Again, the problem can be modeled by creating a weighted graph G, such that each vertex v represents a computer and each edge e a connection between two computers. We can then assign a weight $w(e)$ for each edge e that is equal to the inverse of its bandwidth. Thus, we want to find a tree T that contains all the vertices in G and minimizes the sum

$$w(T) = \sum_{(u, v) \in T} w((u, v)).$$

Such a tree, containing every vertex of a connected graph G is said to be a *spanning tree.* The spanning tree T with smallest total weight is known a the *minimum-spanning tree.*

Because of our specific application domain, we confine our further considerations to undirected, weighted, simple graphs. Thus, unless otherwise stated, we assume that graphs are simple and undirected. This assumption greatly simplifies the presentation of algorithms and data structures.

5.2 GRAPH LAYOUT

The community of mathematicians has been investigating this problems for centuries. In the 1960s, computer scientists began to use graph drawing to assist with the understanding of software. Probably the first paper presenting an algorithm for automatic generation of drawings of graphs was presented by Knuth in 1963 [Knu63]. Today a large variety of applications in science and engineering take advantage of this technology.

The core graph layouting problem can be formulated in simple terms: Given a set of vertices with a multiset of edges, calculate the position of the vertices and the curve to be drawn for each edge. Obviously, the problem itself is not so simple. When layouting a graph, we would like to take into account a variety of aesthetic criteria. For example, the display of symmetries as well as keeping the number of edge crossings at a minimum are often highly desirable in visualization applications. In this scenario, many graph layouting problems may be formalized as non-linear optimization problems.

Most of the following information is based on the consolidated findings reported in [BET+98].

5.2.1 Conventions

A *layout convention* describes the basic rules that a layout algorithm must follow. As an example, in drawing class hierarchy diagrams as part of a software engineering process, we could settle for the convention of representing all the vertices as boxes and all the edges as polygonal chains consisting of horizontal and vertical segments. For real-world applications a layout convention can become very complex, because it takes many details into account. Over time the following list of commonly used layout conventions has arisen.

Table 5.1 List of widely used layout conventions [BET+98]

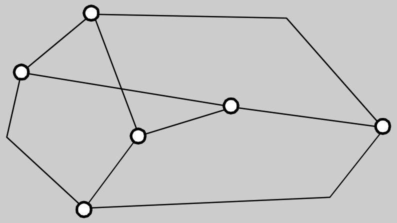
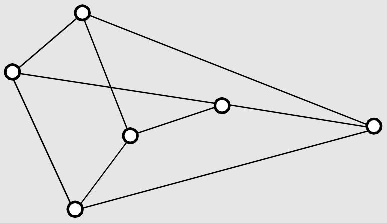
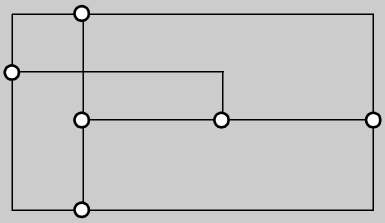
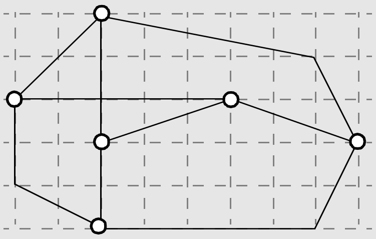
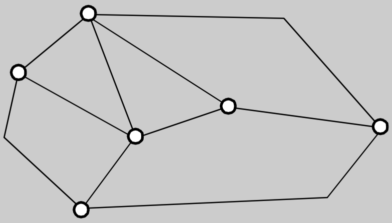
Name / Description	Example
Polyline Layout **Description:** Each edge is drawn as a polygonal chain.	
Straight-line Layout **Description:** Each edge is drawn as a straight line segment.	

Table 5.1 List of widely used layout conventions [BET+98]

Name / Description	Example
Orthogonal Layout **Description:** Each edge is drawn as a polygonal chain of alternating horizontal and vertical segments.	
Grid Layout **Description:** Vertices, crossings, and edge bands have integer coordinates.	
Planar Layout **Description:** No two edges cross.	

Layout conventions represent a rule that a layout must satisfy to be admissible. Obviously, some layout conventions do not work on certain types of graphs. Thus, the class of input graph is an essential decision parameter in favor or against a graph layout methodology.

5.2.2 Key Issues

Key issues specify layout properties that we would like to apply "as much as possible". Most of them concern themselves with aesthetic aspects in order to enhance readability. The following commonly adopted aesthetic rules are ordered by the impact they have on the readability of a graph layout (see [BFN85], [PCJ96] and [STT91]):

Table 5.2 List of commonly adopted aesthetic issues

Name	Description
Predictability	Two different runs on the same graph should result in the same layout.
Crossings	Minimization of the total number of crossings between edges.
Area	Minimization of the area of the drawing, where the area can be defined in different ways. For example, we can define this as the area of the smallest convex hull or simply as the area of the smallest rectangle with horizontal and vertical sides covering the layout.
Total Edge Length	Minimization of the sum of the lengths of the edges.
Maximum Edge Length	Minimization of the maximum length of an edge.
Uniform Edge Length	Minimization of the variance of the lengths of the edges.

Table 5.2 List of commonly adopted aesthetic issues

Name	Description
Total Bends	Minimization of the total number of bends along the edges.
Maximum Bends	Minimization of the maximum number of bends along one edges.
Uniform Bends	Minimization of the variance of the number of bends along the edges.
Angular Resolution	Maximization of the smallest angle between two edges incident on the same vertex.
Aspect Ratio	Minimization of the aspect ratio of the drawing, which is defined as the ratio of the length of the longest side to the length of the shortest side of the smallest rectangle with horizontal and vertical sides covering the layout.
Symmetry	Respect the symmetries of the graph in the layout.
Time complexity	Layouts should be computed at interactive speeds.

Unlike layout conventions, the above key issues are naturally associated with optimization problems. In order to optimize for speed many approximation strategies and heuristics have been developed.

5.2.3 Force-Directed Layout Methods

Most graph layout algorithms are based on the following two simple observations [BET+98]:

- Key issues often conflict with each other. Thus tradeoffs are unavoidable.
- Even if the adopted aesthetics do not conflict, it is often algorithmically difficult to deal with all of them at the same time.

Literature presents many layout techniques, such as [RT81, RMC91, CK95, Mun97], each using distinctive precedence relation among key issues. Hence, each technique aims to satisfy key issues with another priority. Excellent bibliographic surveys [BET+94, HMM00] books [BET+98] and articles [PCJ96] exist concerning this topic.

After having examined the key issues prioritization used by different categories of layout approaches, we have determined–with respect to our field of application–that the class of *force-directed methods* fits best. They deliver highly symmetric layouts, and tend to distribute vertices very evenly. Furthermore, as long as no statistical probability is involved force-directed methods achieve a high predictability. In general, however, these methods can be rather slow. Nevertheless, force-directed algorithms are very popular. Their physical analogies make them easy to understand and relatively simple to code.

P. Eades was probably the first to propose a force directed algorithm for graph layouting [Ead84]. In his paper vertices and edges of a graph were modeled as physical bodies tied with springs. Using Hooke's law describing forces between the bodies, he was able to produce layouts for undirected graphs.

Since then, his method was repeatedly improved and many force-directed algorithms have been proposed [KK89, BF95, HH91, Kam89a, FLM94]. Some of the best known algorithms are titled *Springs and Elastical Forces*, *The Barycenter Method*, *Forces Simulating Graph Theoretic Distances* and *Magentic Fields* [BET+98]. These methods differ, both in the way forces are applied, and in the method used to find an equilibrium configuration for the physical model. Still, in general, all of these have two parts:

1. Model	A *force system* defined by the vertices and edges, which provides a physical system for the graph.
2. Algorithm	This is a technique for finding an equilibrium state of the force system. This state defines a layout for the graph.

Roughly speaking, the model encodes the key issues for the layout, meaning that the forces are parametrized so that the equilibrium configuration is a pleasing layout. For our concrete case this means the embedding of the quantified similarity, which will be described in Chapter 6 more detailed. For the If a model has once be defined, then the layout algorithm may be watched as a abstract physical simulation process, which allows the application of the entire range of numerical methods already discussed in Chapter 4.

5.3 GRAPH RENDERING

Graph rendering is a process which is completely orthogonal to the graph layouting described previously. Independently of the graph's arrangement in space, the rendering determines the visual appearance of a graph. To make things clear, at this point we do not consider the group of techniques used to enhance the visual appearance of a graph as a whole. For those techniques we refer to Chapter 7. This subsection discusses methods to visualize the individual elements of a graph: the *vertices* and the *edges*.

Especially for applications emerging from our concrete field of interest, where graphs are employed as abstract representations for complex associations between data objects, an adequate visual representation is desirable. The procedures known from classical graph theory may not satisfy here, because they are designed with the objective to visualize the structure of a graph only. For this purpose the commonly applied line drawings, as shown in Figure 5.2a, are absolutely sufficient. Each vertex is represented by a labeled box or a circle. Edges are drawn as simple lines.

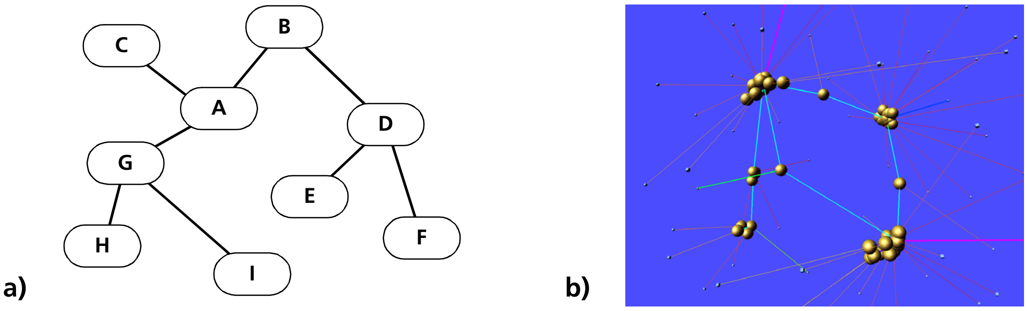


Figure 5.2 Different types of graph rendering: a) Classical line drawings b) More progressive visualization technique in 3D space using colored edges.

For applications where not only the graph structure is of interest, but also the actual attributes associated with the vertices and edges, more complex visualization methods are absolutely vital. The simplest approach would be to label the graph elements with the corresponding values. However, this method would rapidly lead to overloaded and confusing visualizations. To solve this problem, we introduce new visual metaphors by taking advantage of the immense potential of today's rendering methods. In this way, we are able to achieve a significantly higher information density, which results in clearer and more convincing visual representations. Therefore, the basic idea is to map attributes attached to graph elements, such as actual data values for a vertex or the corresponding connection strength for an edge, directly onto the visual parameters of the rendering process. For example, we may map the quantified strength of a connection to the thickness or color of the edge representing the connection (see Figure 5.2b). For the case of vertices we may also think of parametrizing the shape of its visual representation in order to get an meaningful image [CMS99].

Nevertheless, not all of the possibilities offered by today's graphics systems also have advantageous implications for the expressiveness of the visualization. For example, only a few kinds of shapes make sense to represent an edge. In particular, we have to distinguish strictly between the rendering of vertices and the rendering of edges. For these two graph elements two different sets of suitable parameters apply. The following Table 5.3 lists for each of the two elements the visual parameters that have shown expressive results in our research work.

Table 5.3 Compilation of visual parameters showing meaningful results

Graph Element	Parameter	Application Example
Vertex	Color	Discretely or continuously color-coded attribute values: E.g. the color may be switched between green and red to indicate the sign of the underlying value (green = positive, red = negative).
	Texture	If vertices represent for example products, an image of the product may be applied as texture.
	Scaling	The importance of a vertex may be used to scale the vertex. Thus more important vertices will appear larger than others.
	Shape	Static icons or automatically generated shapes based on attribute values: E.g. vertices having the same type of attributes associated may hold an equally shaped icon [CMS99].
	Short Label	Logical name for the vertex.
Edge	Color	Discretely or continuously color-coded attribute values: The color may denote the connection weight or strength.
	Texture	The texture may include an arrow like pattern to express the direction of the underlying dependence [HDH+00].
	Thickness	The thickness may vary dependent on the uncertainty of the underlying relation. Thus, more reliable relation will appear thicker than others.
	Short Label	Logical name for the edge or the connection strength in numerical format.

It goes without saying, that the various parameters may be also used in any possible combination. However, practical experience tells us that some parameters should be applied in a restricted way.

For example, special attention must be paid to the scaling parameter of a vertex when used within three dimensional visualizations. It should be varied by large enough discrete steps only. Otherwise, the scaling will interfere with the perspective projection resulting in confusing images, where one could hardly tell if an object is smaller than the other because it lies behind the other or because it was just scaled down. In addition to the parameters listed above we also tried to alter the shape of edges. However, unless they have a distinctive line characteristic which visually connects the two vertices one can hardly tell the difference between edges and vertices and, as a consequence, we would loose the connectivity information. In our work we therefore use rounded or squared pipes as typical edge shapes.

How the mapping from attributes values to visual parameters will actually be defined depends strongly on the concrete case of application. Thus, no generally valid procedure can be given here.

5.4 IMPLEMENTATION ISSUES

Based on the premise, that we move in an object-oriented environment this subsection covers interface and implementation issues of a graph class. Obviously, the content of a graph object consists of vertices (nodes) and the edges (connections). The graph object should export methods to add, delete and search for those. Additionally, a set of methods returning global information, as well as methods returning vertex and edge specific parameters is required. The internal data structure of a graph object may vary depending on the specific demands of the concrete application.

5.4.1 Graph Methods

We start with the definition of a practical interface for the graph class. One should be aware that this interface is defined for undirected graphs only. Hence, all methods dealing with directed edges are omitted [GT98].

General Methods. A collection of methods returning general graph information.

numVertices()	returns the number of vertices
numEdges()	returns the number of edges
vertices()	returns an enumeration of the vertices
edges()	returns an enumeration of the edges

Updating Methods. Methods enabling the adding and deleting of vertices and edges.

insertVertex(l, o**)**	inserts and returns a new vertex tagged with label l and an associated object o
insertEdge(u, v, l, o**)**	inserts and returns an edge between vertices u and v, tagged with label l and an associated object o
removeVertex(v**)**	removes vertex v and all its incident edges
removeEdge(e**)**	removes edge e

Edge and Vertex Methods . This group takes vertices and edges as arguments and return argument specific informations

degree(v**)**	returns the degree of v
adjacentVertices(v**)**	returns an enumeration of the vertices adjacent to v
incidentEdges(v**)**	returns an enumeration of the edges incident upon v
endVertices(e**)**	returns an two-array holding the end vertices of e
opposite(v**,** e**)**	returns the endpoint of edge e distinct from v
areAjacent(v**,** w**)**	returns true if vertices v and w are adjacent
getEdge($label$**)**	returns edge tagged with $label$
getVertex($label$**)**	returns vertex tagged with $label$

5.4.2 Data Structures

The literature describes several ways to implement a graph with a concrete data structure. In this subsection will discuss different quality aspects of three popular approaches, usually referred as *edge list*, *adjacency list* and *adjacency matrix*. In addition, we compare them to an implementation specially customized to meet the demands of today's graph visualization problems. It is called the *implicit adjacency matrix*.

Which data structure suites the best depends primarily on the characteristics of a graph. In all cases of the representations, the vertices of the graph are stored in a container, for example a list or a hashmap. The major difference the four structures consists in the way of organizing the edges. The edge list and the adjacency list store only the edges that are actually present in the graph. In contrast, the adjacency matrix keeps storage for each possible edge in the graph as a precaution, wether there exists such a edge or not. At last, the implicit adjacency matrix does not store any edge at all. It assumes that all attributes for a edge can be computed or retrieved on-the-fly without much drawback.

As we will explain in this section this difference implies that for a graph G with n vertices and m edges, an edge list or adjacency list representation consumes $O(n+m)$, a adjacency matrix representation $O(n^2)$ and finally a implicit adjacency matrix representation $O(n)$ space. But memory consumption is only one of the critical parameters. The following subsections will present a more detailed analysis.

Edge List. The *edge list* structure is possibly the simplest, thought not the most efficient, representation of a graph G. All vertices are stored in a container V, which typically implements a sequence or hashmap interface depending on type of the primary vertex identifier. If the vertices are numbered in sequence and this number is the vertex's identifier likewise, then a sequence would be the right choice. On the other hand, if vertices are tagged with labels we will prefer the hashmap as container implementation, in order to be able to conveniently search for specific vertices.

The distinguish feature of the edge list structure is the way in which it represents edges. In this structure, an edge e of G is explicitly represented by an edge object. The edges are stored in a container E, which would typically be a sequence or hashmap.

The main feature of the edge list structure is that it provides direct access from edges to its incident vertices. Consequently, methods running edge-centric operations on the graph

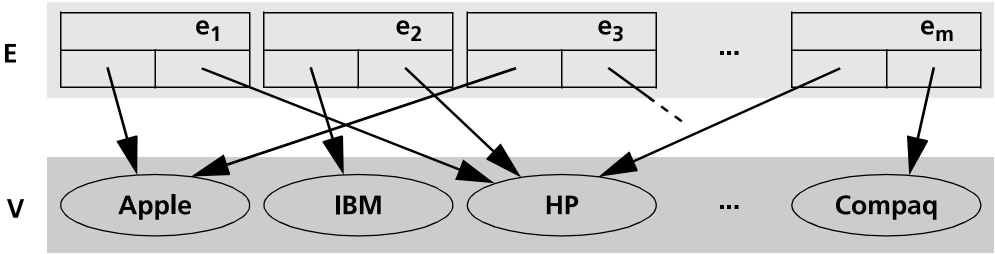


Figure 5.3 Basic structure of an *Edge List*.

are very simple to implement (see Section 5.4.1). Nevertheless, the opposite operations, like finding all edges that are incident upon a certain vertex, require an exhaustive inspection of container E.

A summary of the complexity of graph operations applied to the edge structure implementation of a graph is given in Section 5.4.3.

Adjacency List. The *adjacency list* structure extends the edge list structure by adding extra information that supports the acceleration of vertex-centric operations that require access to the incident edges of that vertex. The adjacency list structure provides direct access both from the edges to the vertices and from the vertices to their incident edges. Therefor, the adjacency list adds a incidence container $I(v)$ to each vertex v. The container stores references to the edges incident on v. Additionally, each edge (u, v) holds a reference to the positions in the incidence container $I(u)$ and $I(v)$.

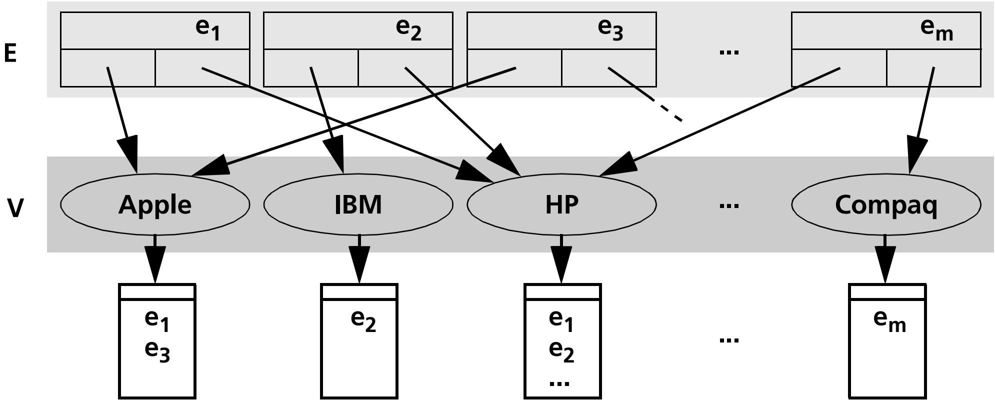


Figure 5.4 Basic structure of an *Adjacency List*.

Table 5.4 illustrates the improved running time of a number of the graph operations in comparison with the edge list structure. The space used is $O(n + m)$.

Adjacency Matrix. From the theory we know, that a graph $G = (V, E)$ with n vertices may be described by matrix A of size $n \times n$. Such a matrix is called *adjacency matrix*. Its rows and columns correspond to vertices, with cells A_{uv} holding a reference to the edge (u, v), if the edge exists.

Thus, similar to the adjacency list structure, the adjacency matrix representation of a graph extends the edge data structure with an additional component, the adjacency matrix A. It allows us to determine adjacencies between pairs of vertices in constant time $O(1)$. In this case, the performance gain comes at a price in the memory requirement, which is now $O(n^2)$, and in a increased running time of other methods (see Section 5.4.3). Moreover, if the matrix A is implemented by a two dimensional array, each vertex insertion or deletion now requires the allocation of a whole new matrix A, which takes time $O(n^2)$.

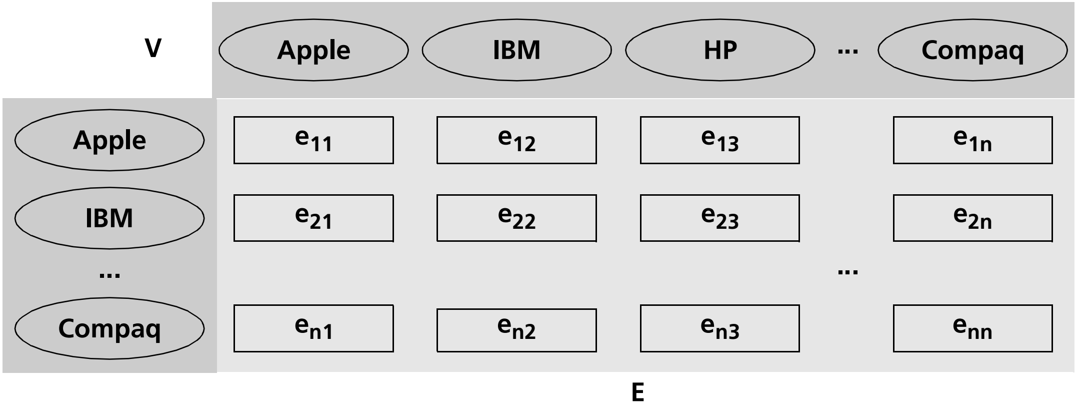


Figure 5.5 Basic structure of an *Adjacency Matrix*.

However, this drawback can be eased by choosing a more advanced data structure to implement matrix A. First of all we may implement the $n \times n$ matrix A by a linked list of n one-dimensional arrays each of size n. This drops the time needed to insert and delete a vertex to $O(n)$. Furthermore, since we work with undirected graphs, the adjacency matrix A is of diagonal-symmetric from, allowing us to reduce it to a lower-left triangular matrix. After all, the space needed decreases to $O(n^2/2)$. There are also approaches using hashmaps in order to represent an adjacency matrix A. Hashmap implementations typically allocate space for edges actually present in the graph only. Thus, depending on their load factor they show a much slimmer space usage pattern. However, since hashmaps do not preserve the order of elements they contain, heavy drawbacks may imply for graph operations normally profiting of the matrix structure ordered by columns and rows (e.g. operation `incientEdges`).

Implicit Adjacency Matrix. The *implicit adjacency matrix* structure takes into account, that the memory consumption denotes the main bottleneck for all of the previous structures when looking at real-world problems. On the other side, most of the time our computer's CPU and network run under moderate load only. Thus, it might be faster to recompute connection parameters on request, than storing them in a data structure that consumes huge amounts of memory and may put the machine in a slow swapping status.

The implicit adjacency matrix, shares the same interface as the standard adjacency matrix structure. But, it does not store any information about edges. Consequently, the implicit adjacency matrix demands no space for an edge data structure. Instead an *implicit rule* is given in order to create a requested edge and all its attributes on-the-fly. This rule mainly consists of a piece of code, containing the knowledge of where to get the edge data and how to compute the corresponding attributes. Whenever an edge object is accessed this rule is called and a so called *shadow edge* for the actually requested edge will be returned. A shadow edge may be considered as a read-only view onto the edge's attributes (see Figure 5.6).

In order to find a machine dependant balance between memory consumption and CPU load, a configurable cache might be attached storing already computed shadow edges. Edges that are requested more than once are then directly returned from the cache. Beyond that, if the represented graph G is small enough or sparse, the implicit data struc-

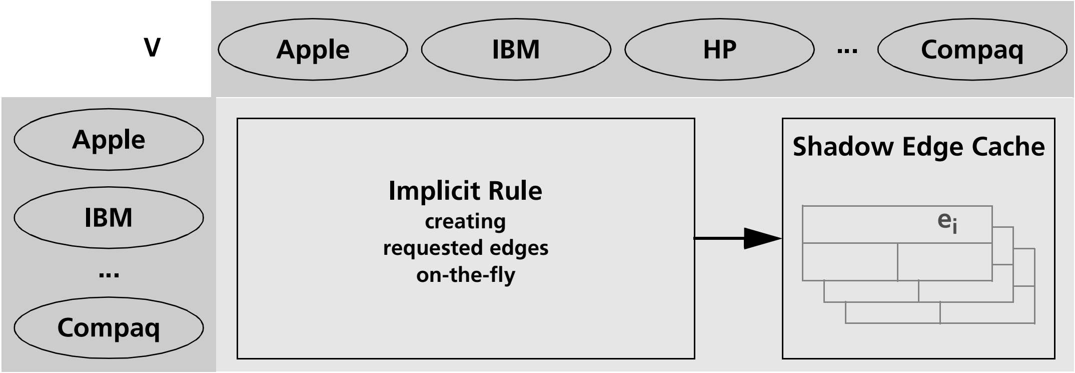


Figure 5.6 Basic structure of an *Implicit Adjacency Matrix*.

ture might be translated into an adjacency list structure (as described above) by applying a compilation mechanism. Such a structure then represents a adequately optimized structure for such type of graphs.

5.4.3 Complexity Considerations

The following table provides an overview of the complexity of some typical graph operations. The operations are listed in the table's rows, whereas each column names a distinct graph data structures on which the operation is performed.

Table 5.4 Complexity considerations of the graph data structures presented previously in Section 5.4.2.

Operation	Edge List	Adjacency List	Adjacency Matrix	Implicit Adjacency Matrix
numVertices, numEdges	$O(1)$	$O(1)$	$O(1)$	$O(1)$
vertices	$O(n)$	$O(n)$	$O(n)$	$O(n)$
edges	$O(m)$	$O(m)$	$O(m)$	$O(m)$
insertVertex	$O(1)$	$O(1)$	$O(2n)$	$O(1)$
insertEdge	$O(1)$	$O(1)$	$O(1)$	-
removeVertex	$O(m)$	$O(deg(v))$	$O(2n)$	$O(1)$
removeEdge	$O(1)$	$O(1)$	$O(1)$	-
opposite, degree, endVertices	$O(1)$	$O(1)$	$O(1)$	$O(1)$
incidentEdges, adjacentVertices	$O(m)$	$O(deg(v))$	$O(n)$	$O(n)$
areAdjacent, getWeight	$O(m)$	$O(deg(v))$	$O(1)$	$O(1)$
Memory requirement	$O(n+m)$	$O(n+m)$	$O(n^2)$	$O(n)$

In conclusion, we observe a trade-off between memory consumption and running time. Since, each of the discussed graph representations has its characteristic strengths, one has to take the graph's structure into account to achieve acceptable results.

6

THE IVORY FRAMEWORK

Encouraged by the convincing results obtained using some of our first visualization prototypes, which have clearly proven the feasibility of our idea to utilize physics-based models in order to effectively visualize relations in arbitrary data spaces, we decided already in a very early stage of our work to build a framework. This decision was supported by the observation that the individual prototypes shared a significant amount of structure and code, independently of the type of input data they were operating on.

The IVORY framework we present in this chapter has been developed as an open and flexible system for information visualization. It introduces a new set of visualization and interaction paradigms for abstract data sets. It closes the gap between the well proven general purpose visualization tools, such as AVS [AVS97] and the IBM DataExplorer [IBM91], mainly designed for the purpose of scientific visualization and the powerful but restricted information visualization methods. We refer to Section 1.1.4 on page 7 for a more complete overview of existing systems and algorithms.

From a technical point of view, the design of our frame work follows strictly the *Operator Framework Architecture* we have developed in Chapter 2. Within this architecture we also address the various theories presented in the earlier chapters of this work. Although this framework is primarily designed for physics-based approaches to multidimensional information spaces, the underlying design principles make it a versatile framework for the investigation and application of new visual metaphors.

6.1 REQUIREMENTS

The list of requirements for the IVORY framework was inspired by various interesting approaches [Wis+95, Cha96, BF95, Woo95, YR91, GK95, Ben96, HD95 and CEG96] that can be found in literature. Each of these shows its specific strengths and weaknesses in different application scenarios. We have consolidated these behaviors and combined it with the knowledge we gained from experimenting with our protoypes to define the requirements for the IVORY framework. The following enumeration lists the six most crucial requirements for the framework:

- *Scalability*: Usually, the kind of systems we build here only have an advantage over conventional visualization techniques – such as standard business charts – when the data to be analyzed is of large volume and high complexity. Thus, the system must be scalable in order to be able to handle large amounts of multi-dimensional data, which might even vary over time.
- *Flexibility*: One of the main requirements concerns the flexibility. As a framework, it must be designed with the potential to handle different type of input data with a minimal amount of configurational effort.
- *Extendability*: The framework must be open to handle future problems dealing with new – yet even unknown – data types. This calls for some kind of a plug-in mechanism, that allows the dynamic addition of data-specific functionality, which might include operations specific to a certain data type, visual metaphors or the support for data-specific interaction techniques.
- *State-of-the-art visualization technique*: The applied visualization technique should make use of the users's cognitive system while taking advantage of the performance of today's graphics hardware.
- *Clustering and hierarchies*: The huge amount of information forces the use of a multi-resolution setup in order to break down the complexity. Hence, appropriate methods for the clustering of objects and for interactive level-of-detail control are needed.
- *High Portability*: Two important aspect have to bear in mind to give a framework the chance for general acceptance: First the pervasive compliance to established standards (OpenGL, VRML, etc.) is essential. Second, the implementation should be as independent of operating systems as possible.

6.2 THE FUNDAMENTAL VISUALIZATION CONCEPT

Especially for the case of large data sets the individual data object becomes almost insignificant. In contrast, such data volumes inherently imply a dense network of multidimensional relationships, where in general each information unit is related to many other units. The resulting topological organization corresponds to a multidimensional graph, that describes these relations.

Our experience has shown that visualization techniques that rely on the connectivity information of data sets show very promising results. The main idea consists in the quantification of these high-dimensional relations, which yields a measure for the degree of similarity for each pair of two data entries. Thus, we now deal with a *weighted* multidimensional graph, where the weights are equal to the quantified similarity between the two data nodes attached to the ends of an edge. We may also think of the weight as the distance. Thus, we end up with the ability to express the similarity between two data objects by means of a distance.

However, the reader should be aware that we still deal with high-dimensional adjacencies, which cannot be visualized straightforwardly and have to be mapped into a subspace – the Euclidean space for our concrete case. There exist several techniques for topology-preserving transformations from a high dimensional data domain to Euclidean space [LVC95]. One of them is called multidimensional scaling (MDS) [You87]. Other widely spread methods are employing with neural networks, namely with topology-preserving Kohonen networks [Koh95, GS93], which belong to the group of self-organizing features maps (SOM). As a third technique spring-embedding systems (SES) perform the desired transformation by running a physics-based simulation process [BF95, HH91].

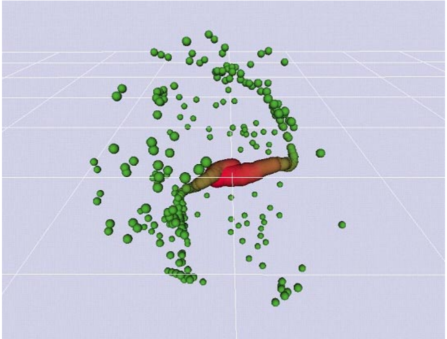


Figure 6.1 Typical result computed by the IVORY framework.

The IVORY framework will base on the latter technique, which means that we will make consequent use of the theory of physics-based models presented in Chapter 4. Together with the methods developed in Chapter 5, we are now able to express high-dimensional similarities between two data objects by means of a simple Euclidean distance. Thus, more similar data objects will be arranged closer together, than those having less in common. The result will typically present a set of object groups – so-called clusters – similar to the image shown in Figure 6.1.

6.3 IMPLEMENTATION OF THE ABSTRACTION MODEL

The IVORY framework strictly follows the *Operator Framework Architecture* (OFA) presented in Chapter 2. This section will describe how the architecture is actually put into action for the concrete case of the IVORY framework.

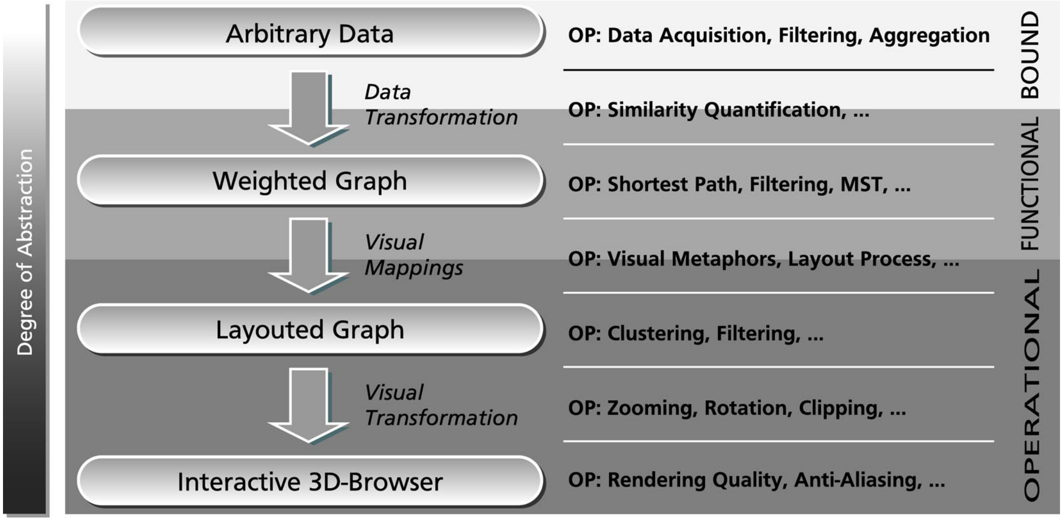

Degree of Abstraction
Arbitrary Data
Data Transformation
Weighted Graph
Visual Mappings
Layouted Graph
Visual Transformation
Interactive 3D-Browser
OP: Data Acquisition, Filtering, Aggregation
OP: Similarity Quantification, ...
OP: Shortest Path, Filtering, MST, ...
OP: Visual Metaphors, Layout Process, ...
OP: Clustering, Filtering, ...
OP: Zooming, Rotation, Clipping, ...
OP: Rendering Quality, Anti-Aliasing, ...
FUNCTIONAL BOUND
OPERATIONAL

Figure 6.2 IVORY's consequent implementation of the *Operator Framework Architecture* (OFA).

The organization of the following subsections follows closely the structure of the OFA. We start on the level of the raw data and continue in the direction of the increasing level of abstraction up to the visual representation of the original data. We describe the data representations for each abstraction stage as well as the concrete implementations of the different stage transition operators.

Raw Data. In IVORY the raw data may consist of an arbitrary data type. No restrictions apply to the type, volume, complexity or origin of the input data.

One of the most important operations on the level of *data stage operators* is the data acquisition. It is the starting point of every visualization or visual mining procedure and supplies the system with raw material. In the simplest case the data can be read from an ASCII-file. For more complex scenarios, typical data sources encompass conventional data bases (such as SQL, DB2, DBase), search-engines in the internet, the World Wide Web (WWW) or real-time data tickers (e.g. Reuters Ticker). The acquisition process obviously belongs to the data-dependent operators and therefore needs to be implemented for each data type anew.

Other typical *data stage operators* are filtering and aggregation operators. However, if we implement those on this level we have to be aware that – according to the OFA – they belong to the class of *bound operators* and thus require a new implementation for each new data type. Especially for the case of the filtering operators, one should take the possibility of a reusable filtering operator on a higher abstraction level into account.

Data Transformation. The process of data transformation embeds the original data into an abstract representation. In the concrete case of the IVORY framework the abstract representation consists of a weighted graph data structure (see Section 5.1). This transformation creates a vertex for each data object. In addition, relations between data objects are mapped to corresponding edges in the graph, where the weight for each edge is defined by the quantified similarity of the two associated data objects.

The set of *data transformation operators* are either of bound or functional type. *Bound operators* may be required prior to the similarity computations in order to convert the raw data to one of the formats proposed in Section 3.4. The actual similarity computations belong to the group of *functional operators*, because they are semantically similar but typically vary in their concrete implementations.

Analytical Abstraction. The analytical abstraction constitutes the core data structure of the IVORY framework. This layer implements one of the weighted graph data structures presented in Section 5.4.2. It stores the connectivity information as well as the meta-data for an application. Therefore, we refer to its edges also as *connections* and to its vertices also as *data objects*. The weighted graph is the bases for many of the following operations including any kind of connectivity analysis or layout operations.

Based on the OFA the operators of this layer are of functional type. However, practice has shown that we have to distinguish two different groups of *analytical abstraction stage operators*: on one hand there are those that work on the raw data, e.g. filtering operators working on the connectivity information as well as on some attributes of the underlying raw data. These operators consists of data-dependent parts and are thus of functional type. On the other hand we have the large group of classical graph operators (e.g. path-finders [TLA90, GT98], minimum spanning tree, etc.) that act on the abstract graph data only. These operators can even be classified as *operational operators*.

Visual Mappings. The *visual mapping operators* are responsible for transforming the analytical abstraction into a visual representation, thus determining its resulting visual appearance. Since our analytical abstraction consists of a standard weighted graph data structure, we may draw on rich resources from the field of graph visualization (see Chapter 5).

According to Chapter 5, IVORY applies a force-directed technique for the *graph layouting* (see Section 5.2.3). In the current setting the *advanced force model* introduced in Section 4.2 is used as the standard *layout model* for IVORY. As a part of the layout model we propose the following four different mapping configurations for the stiffness k and rest length l_0:

- Similarity s_{ij} mapped to the zero length l_0 only:

  $k = const$ (Chose $const$ sufficiently large)

  $l_0 = inv(s_{ij})$ (Rest length inverse proportional to stiffness)

- Similarity s_{ij} mapped to the spring stiffness k only:

  $k = s_{ij}$ (Stiffness proportional to rest length)

  $l_0 = const$.

- Similarity s_{ij} mapped to both parameters l_0 and k:

  $k = s_{ij}$ and $l_0 = inv(s_{ij})$.

- Similarity s_{ij} mapped to the rest length l_0 and the reliability r of the value of s_{ij} mapped on the spring stiffness k:

  $k = r(s_{ij})$ and $l_0 = inv(s_{ij})$.

All configurations listed above have in common, that similar data objects will be pulled together by a large k and/or a small l_0. The last version also takes the reliability of individual similarity values into account and is used for the visualization of uncertain or unreliable data. Physical model parameters not named in the configurations above can be considered as data independent and are mainly used to optimize the readability of the layout (see Section 5.2.2).

Concerning the *graph rendering*, a set of operators is provided to assign the desired visual metaphors to the elements of our visual representation (see Section 5.3).

Visualization Abstraction. The visual abstraction is implemented by a layouted graph. By "layouted" we mean, that each of the vertices has its coordinates attached and the graph is now displayable in a viewer.

Valid *visualization abstraction stage operators* are for example filtering and clustering operators. By the use of filtering operators, for instance, the user can select interesting subsets of objects. Conversely, clustering operators are used as a method to handle complex visualizations, where groups of objects are condensed to one meta-object, a *cluster*. Because clustering operators constitute a key issue of this work, they will be discussed in a separate chapter (see Chapter 7). All operators on this level belong to the class of *operational operators*. Hence, they act completely independently of the concrete type of the underlying raw data.

Visual Transformation. The group of *visual transformation operators* covers most of the user navigation functionality. With a small set of zooming, translation and rotation operators we enable the user to navigate through the layouted graph.

All these operators belong to the class of *operational operators*, because they work directly and exclusively on the layouted graph.

Viewer. The IVORY viewer is implemented by an interactive 3D-Browser. It is responsible for the correct rendering of the layouted graph. Operators on this level mainly serve to adjust some rendering parameters, such a the rendering quality or the settings for anti-aliasing.

6.4 SOFTWARE ARCHITECTURAL ISSUES

The IVORY software architecture results from a combination of the list of requirements (see Section 6.1), the understanding we obtained from the theory of the OFA (see Chapter 2) and the experience gained during this research project. Hence, the architecture was not defined in a single step, but rather crystallized from a dynamic development process.

As illustrated by the horizontal separation in Figure 6.3 we employed a frontend/backend-concept, where the backend is responsible for efficient number crunching and the frontend handles the graphical user interface and user interactions. In addition, we strictly distinguish between data-dependent and data-independent components (vertical separation in Figure 6.3). All data-dependent components are thoroughly distilled into so-called plug-ins (see Section 6.4.2). All other parts are packed into a highly optimized and data-independent kernel structure (see Section 6.4.1).

Thus, speaking in the consent of the OFA the kernel contains only abstract data structures and *operational operators*, whereas plug-ins accommodate bound as well as *functional operators*. Hence, in order to visualize a new raw data type, the user essentially needs to write an appropriate plug-in, while the remainder of the system will not be affected at all.

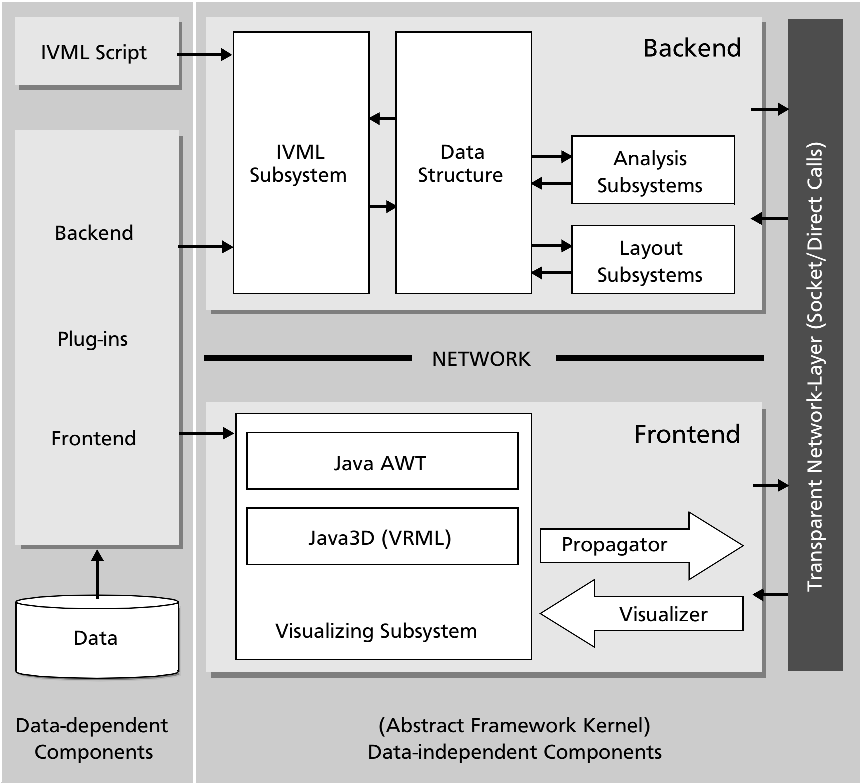


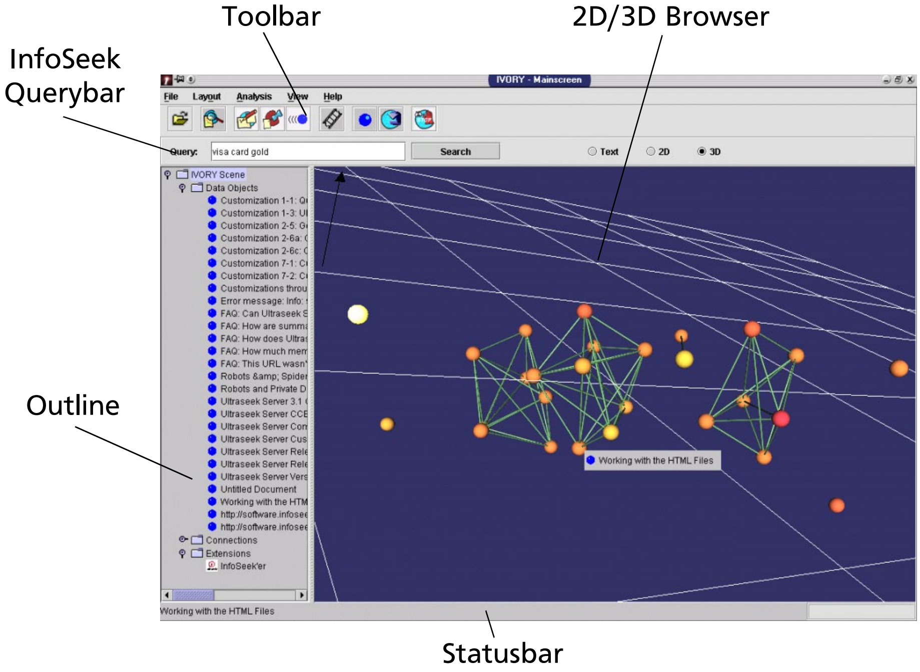
Figure 6.3 Schematic overview of the system-components in IVORY

The system is completed with a script language – called IVML – handling the configuration of the kernel and the individual plug-ins. For further details we refer at this point to Chapter 8.

6.4.1 Framework Kernel

The abstract framework kernel of IVORY contains generic implementations of all common components underlying our physics-based visualization approach. In particular, this also includes all *operational operators*. The kernel belongs to the set of data-independent components of the framework. Thus, kernel operators are highly reusable (see Chapter 2) and shorten the design cycles of novel visual metaphors. Unlike most visualization systems, where the frontend/backend-separation is introduced to detach system-dependent from system-independent code segments, we employed this notion primarily to run the system in a client-server setup over a network.

Frontend. The frontend is designed to run in a Java enabled WWW-browser. It consists of three parts. The *visualization subsystem*, which is responsible for all visual system outputs, as well as for the handling of user inputs. The *2D graphical user interface* (GUI), shown on the left hand side of Figure 6.4, covers all standard I/O tasks with appropriate menus and dialog boxes. By default an outline of all loaded objects (plug-ins) is shown as a collapsable tree structure. The *3D browser* is presented on the right side of Figure 6.4. It manages the visualization of the calculated object arrangements and basic navigation functions. Note that the visualized objects do not belong to the frontend. In order to decouple the frontend from the viewer we defined a generic 3D viewer interface. Thus, only the defined interface has to be re-implemented when changing the VRML viewer.



Query: visa card gold

Working with the HTML Files

Figure 6.4 Screenshot showing the main GUI elements of the IVORY frontend.

In addition we introduce two messenger components. The *propagator* is responsible to inform the backend whenever a user interaction has invalidated the integrity of the frontend and the backend data-structures. E.g. if the user manipulates the layout parameters, the backend has to recalculate the corresponding object arrangement and synchronize itself with the frontend. The *visualizer* has a similar task, but takes care of the opposite direction from the backend data-structure to the frontend visualization.

Backend. The backend supports two different execution modes: either it is directly attached to the frontend and runs in the same address space or it runs as a separate server-application on a different machine. In the second case the frontend and backend communicate over a Java socket connection.

One of the main design goals for the backend was a responsive data-structure (see Section 5.4.2), where all the instantiated objects (e.g. plug-ins) are stored. It is strictly optimized for fast object insertion, deletion and look up. The structure is initialized by the IVML subsystem (see Chapter 8), which is responsible for dynamic object instantiation according to the parsed configuration script. Hence, this subsystem must be able to load and link unknown plug-ins at run-time.

As indicated in Figure 6.3, we distinguish two different subsystem components accessing the core data-structure directly. Based on the objects stored in the core structure the layout subsystem builds up a extended mass-spring-network similar to the one presented in Section 4.2. It also contains state-of-the-art differential-equation-solvers to simulate networks relaxation over time. This includes specifically gradient (Euler, Runge-Kutta) or stochastic (annealing) based methods. In practice, gradient algorithms turn out to be much more suitable to treat time-varying data (see Section 4.6). The second subsystem is the analysis component comprising selection filters, path-finders (see Section 5.1.2) and clustering algorithms (see Section 7.2 and Section 7.3). The subsystem interfaces allow users at a *systems designer's* level to easily extend kernel algorithms (see Section 6.5).

Transparent Network-Layer. All communication between frontend and backend is streamed over the network-layer. Hence, the communication is generally transparent to all system components from above and guarantees full independence of the IVORY execution mode (stand-alone or client/server).

6.4.2 Plug-in Mechanism

From the conceptual point of view plug-ins serve as a container for all data-dependent operators (such as *bound* and *functional operators*). They have a standardized interface and are dynamically loadable at system run-time. The set of basic plug-in types (`BaseData`, `BaseConn`, `BaseCluster` and `BaseAbstract`) reflect the fundamental components the framework consists of. In fact, these are the only plug-in types that are actually known to the framework kernel.

While defining the abstraction level of IVORY's plug-ins, we focused on productivity and ease of use. Since plug-ins only contain functions implementing a specific metaphor or theorem the plug-in programmer can concentrate his efforts on the data-specific issues. Recalling the abstraction model shown in Figure 6.2 a plug-in may contain, for instance, a similarity metric and a visual metaphor parametrizing the visualization.

For reasons of simplicity plug-ins contain both frontend and backend components. Conceptually, the separation is reflected by two groups of methods. Unlike AVS 5 [AVS97], where separate computation and description modules exist, an IVORY plug-in is always viewed as one entity even though individual classes belong either to the client or to the server.

The design of new plug-ins takes advantage of the object-oriented approach of the framework, where we make extensive use of object inheritance. All plug-ins are derived

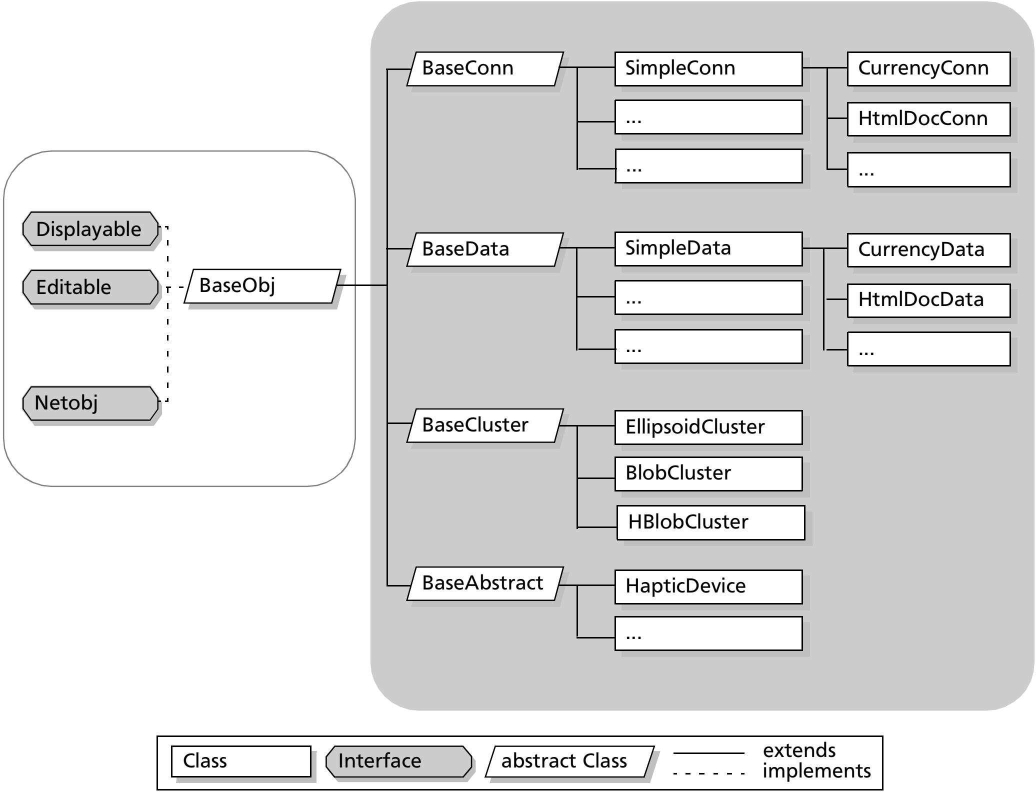


Figure 6.5 IVORY's plug-in hierarchy: The base classes.

from a small set of base classes and are thus organized in a hierarchical structure as illustrated in Figure 6.5. The most important classes are discussed in the following subsections.

Base Objects. The base object class (`BaseObj`) is the root of the plug-in hierarchy. Virtually it is the only object class known in the abstract information visualization kernel of the system. It essentially determines the set of methods the plug-ins may be accessed through.

A selection of the most important (abstract) methods are given below:

Frontend Methods

- `getIcon` (abstract)
  Optionally returns a reference to an icon resource. If available, it will be displayed in the 2D outline window.

- `getAppearance` (abstract)
  Returns the description of the representation of the object in the 3D viewer and is described in native VRML 2.0 (see [VRM97]).

- `getDisplayComponent` (implemented)
  This method provides the dialog component, which displays the information contained in the corresponding object. This includes administrative information (id, position, flags, ...) as well as stored user data.

- `getEditComponent` (abstract)
  If the object features editing, the method provides an editor component for the above information. This enables specifically data editing at runtime.

- `getVisPar` (implemented)
  Returns the previously calculated visual parameters by calling the method `calcVisPar`.

Backend Methods

- `calcVisPar` (abstract)
  Calculates the visual parameters (position, scale, orientation and color) of an object depending on the underlying data. This way, specific data-properties can be mapped onto visual attributes.

Data Objects. Data objects are directly derived from the base object class and serve as an access interface to the explored data. The data management is individually solved by the actual implementation of a data object. For each instance of a data object a corresponding particle-mass is automatically created in the layout-subsystem. The setup of particle parameters is handled by the additional backend method explained below.

Due to its paramount importance for many applications we support time-variant data and multiple data channels per data object in our implementation. A good example is the analysis of a set of critical interest rates of different countries over the last years. In this case, a data object is allocated for each country. In each data object one channel is opened for each data feed.

Additional Backend Methods

- `calcParticle` (abstract)
  In this method non-visual parameters of the particle attached to the data object could be parametrized. In this way the object behavior during the layout process can be defined.

Connection Objects. Connection objects are of the same inheritance level as the data objects. This object type serves as a binding object, which represents the relation of two data objects and is of fundamental relevance for the resulting object layout.

For each connection object instance a corresponding spring is created in the layout-subsystem. The physical parameters of the spring are defined via the additional backend method described below.

Additional Backend Methods

- `calcConnection` (abstract)
  In this method non-visual parameters of the spring corresponding to the connection object can be parametrized.

Cluster Objects. Another object type of the first derivation level are the cluster objects. They enable a hierarchical organization of the visualization and can be looked at as data object containers.

Note, that they are created by the corresponding clustering algorithm located in the analysis subsystem. The details of these algorithms are discussed in Chapter 7. For each identified cluster a new instance is allocated. The appearance of this instance is a result of the cluster analysis calculations. Figure 6.6 shows a scene containing several differently shaped cluster instances.

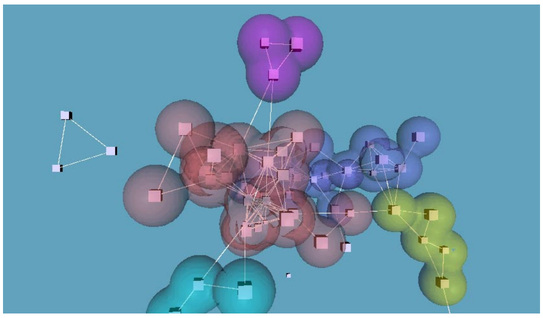


Figure 6.6 A example scene containing several cluster object instances. The corresponding shapes are computed using a blobby cluster method (see Section 7.3.2).

Abstract System Object. Our experience has shown that generic objects without a visual appearance are very helpful for efficient solutions. Thus, we introduce so-called abstract system objects. Examples are parameterized global functions, such as data or currency converters.

Another area of application is the attachment of additional I/O-devices. For example, our physics-based system is predestinated for the use of force-feedback devices. An object representing the device is derived from the abstract system class and helps to seamlessly integrate it into IVORY.

6.5 FRAMEWORK CONFIGURATION

According to the *abstraction model* of the OFA we may define configuration layers that enable the configuration of the framework on different abstraction levels. These layers result from a slightly different view of the framework architecture and are shown in Figure 6.7. At the bottom we find the physics-based information visualization kernel. It can be configured through a very low level interface only. The plug-ins are located in the middle of the scheme. They form the elements for a higher level configuration of the underlying kernel. Finally, on the top of the design, the IVML script language completes the configuration model of our framework. As described in Chapter 8 this script language offers the highest level of abstraction regarding the configurability of the framework. Hence, the level of abstraction in Figure 6.7 increases from the bottom to the top.

It is remarkable that the degree of abstraction for the configuration layers runs exactly in opposite direction to the one of the operators: operators of a high abstraction level offer generally configuration capabilities of a low abstraction degree only, and vice versa. For example, the kernel which accommodates operators of a very high abstraction level, provide configuration interfaces having only a very low degree of abstraction. In general, only technically experienced users, such as system-designers, are capable of configuring the framework kernel. On the other hand, the IVML script language offers a very high level of abstraction. But, since IVML scripts task and data-dependent, they belong to the operators with a low degree of abstraction.

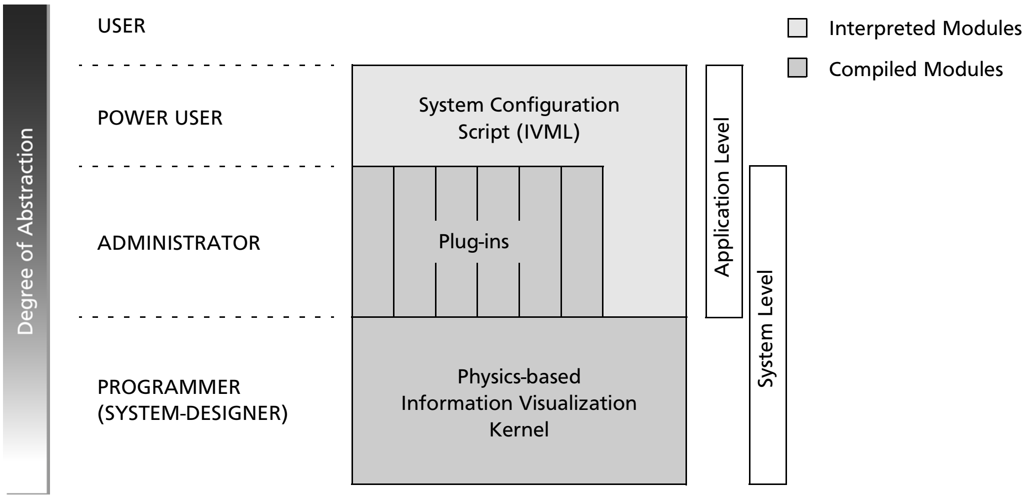


Figure 6.7 User abstraction model and system architecture

In order to offer an appropriate interface for different types of users, IVORY can be configured at 4 different levels of abstraction, depicted in Figure 6.7. The *standard user*, such as a financial analyst, will apply preconfigured instances of IVORY in his everyday work. The more advanced *power user* can customize existing configurations using the script language IVML (see Chapter 8). *Administrators* may program Java plug-ins (see Section 6.4.2) implementing new visual metaphors and layout algorithms. Finally, the *systems designer* may modify and extend the IVORY kernel by adding new classes for solvers, particle engines or other kernel methods.

6.6 IMPLEMENTATION ISSUES IN JAVA

Some Arguments pro Java: In this context, a new trend in portability emerges with Java [SUN97] and its *"write once, run everywhere"* philosophy. In this way applications run (almost) on any operating system supporting Java. We believe that the leak of performance compared to C++ will get obsolete over time. Here, the performance improvements made with every new Java release and the availability of native code Java compilers [IBM97] are good evidences.

In addition, besides all the euphoria, Java is widely accepted as a highly portable quasi-standard in the computer community and is available on many operating systems. Furthermore Java is strongly object-oriented, which is critical for a well structured implementation of a complex framework.

The IVORY implementation takes full advantage of all sophisticated features provided by JAVA. Each plug-in is mapped onto a Java class. Thus, the loading of individual plug-ins at runtime can be accomplished by the dynamic class loading mechanism of Java. The Java reflection model is used by the IVML interpreter to check, read and set the field values of plug-in objects.

We use the naming convention for setter and getter methods of Java Beans to access the underlying Java member variables. The Beans compliance enable to use all advanced Beans features [JavaBean96], such as property editing. Although our current client-server implementation is based on a proprietary object serialization protocol, we are currently working on an RMI-based method invocation.

One of the critical implementation issues of IVORY was the 3D API. The OpenGL bindings, available in beta-version from [Mag97], are fast, however, are on a low abstraction level and do not support advanced object management by scene graphs. In addition, some rendering features, such as texture mapping are not implemented. In addition, VRML parsing has to be provided by the user.

An early version of IVORY was based on Dimension X's Liquid Reality class library [MS97], which was the only appropriate 3D Java API at that time. The robust beta version supported VRML and has been ported onto many platforms. Since the scene graph data structures are maintained in Java, the system performance is low.

Our next implementation uses SGI's CosmoPlayer [Cos97] for visualization. Unlike the libraries from above, the software is essentially a VRML 2.0 viewer plug-in for WWW browsers which has no immediate 3D API for Java. However, viewer control and callback-functions can be invoked through the LifeConnect mechanism of the WWW-browser and the External Author Interface (EAI) of CosmoPlayer. The scene rendering is based on OpenGL and thus supported by a wide range of hardware accelerators. At this time CosmoPlayer is available for SGI, Windows 95/NT and Apple Macintosh.

Unlike OpenGL, Sun's Java3D [J3D97] provides a powerful API for 3D graphics including scene graph optimization and VRML extensions. The platforms comprise SUN, HP, SGI, Windows 98/2000 and Linux. The current version of IVORY uses Java3D to implement the 3D browser interface. In its present version Java3D provides a stable and reasonable fast graphics library. In addition, Java3D's rendering engine is build on top of OpenGL, which, if an appropriate graphics system is present, enables hardware accelerated rendering.

7

CLUSTERING

The term *clustering* refers to the process of grouping similar objects together, where similarity is captured by a metric function as it was presented in Chapter 3.

Clustering methods have been a hot topic in different research fields such as: statistics, pattern recognition, machine learning, etc. Due to the constantly increasing size of data sets handled over the last years, clustering also has advanced to a key technology in the area of *information visualization* and *data mining*. In fact, with the use of today's technology for data generation and collection, typical data sets have grown by magnitudes. Since the human cognitive system is limited to recognize only a very small number of objects at once (around 7 objects [Mil56]) as well as due to performance restrictions of today's graphics hardware we are forced to use an efficient level-of-detail strategy. Consequently, the literature describes various interesting data clustering approaches including their efficient and refined implementations [DO74, GGR99, GRS98, HH98, JD88, KHK99, ZRL96].

In this chapter we propose three additional clustering algorithm for postprocessing. The simplest one clusters individual objects by computing an ellipsoidal hull around them. The ellipsoid is parametrized by the principal components of the underlying object group. For more complex scenarios we introduce a blobby clustering mechanism, called BLOB, that enables encapsulation of similar objects by implicit shapes. Finally, we present a direct successor of the BLOB algorithm, called H-BLOB, which groups and visualizes cluster hierarchies at multiple levels-of-detail. Our clustering algorithms are highly reusable for a large variety of data clustering tasks, because they are all implemented as *visualization abstraction stage operators* (see Section 2.4).

7.1 PRINCIPLES

Because of the reusability criterion our main interest lies in designing clustering algorithms that operate on the *visual abstraction level* as defined by the OFA introduced in Chapter 2. Since the visual abstraction is implemented by a layouted graph we focus on the problem of clustering large data sets in *Euclidian space*, also referred to as the *coordinate space* [GRG+99], in which data objects can be represented as vectors $\mathbf{v} \in R^n$. Unlike raw data objects located in a *data domain*, also referred to as the *distance space* [GRG+99] or the *arbitrary metric space*, the vector representation gives access to various efficiently implemented vector operations (e.g. addition, multiplication, dot-product, etc.), which enables one to calculate simplified representations of complex data subregions at interactive rates. No similar operations are defined in distance space. The only possible operation is the computation of a distance function between two data objects, thus rendering the problem of clustering gets significantly more complex.

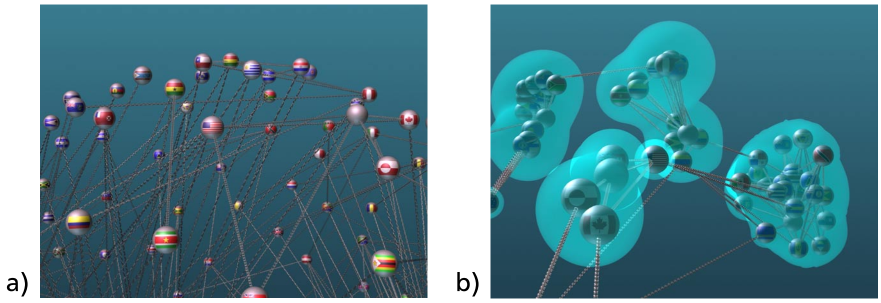


Figure 7.1 Clustering of a subset of objects performed with BLOBS. a) Initial object layout b) Clustered configuration with enclosing implicit surface.

In order to simplify the geometry and topology of complex object setups, we will provide a set of clustering algorithms for postprocessing. In contrast to many other cluster-based systems, we will not only calculate clustered object layouts including corresponding one- or multi-level partitions (as a group of cluttered single objects) but we will also compute different types of enfolding surfaces (ellipsoids, implicit surfaces, etc.) for each cluster [DO74, Epp95]. Aiming at a reduction of complexity, such a surface can replace a large group of single objects in a higher level of representation. Without losing significant visual information, the complexity of the scene may be drastically reduced. At the same time, the visual distinctness increases. In our terminology we call the computation of one- or multi-level partitions *analytical clustering* and the finding of enfolding surfaces *visual clustering*.

The remainder of this chapter is organized as follows. In Section 7.2, we give an overview of analytical clustering algorithms. In Section 7.3, we present different techniques we use for visual clustering. We introduce three different algorithms dedicated to visualize partitions and cluster hierarchies. The chapter closes with Section 7.4 describing implementation issues.

7.2 ANALYTICAL CLUSTERING ALGORITHMS

Clustering algorithms can be roughly divided into two categories: *partitioning* and *hierarchical* methods. In the following two subsections we present a variety of widely used partitioning, respectively hierarchical clustering algorithms, followed by a description of different advanced cluster visualization techniques.

The following list is far from being complete, but it should outline the main clustering techniques, upon which most of today's clustering algorithm are based. This section mainly intends to set our work into context and to better understand our new approaches.

7.2.1 Partitioning Methods

Partitioning cluster methods (PCM) attempt to analytically subdivide a set of data objects into a certain number of clusters, assuming that clusters are of hyper-ellipsoidal shape and of similar size. Like other centroid-based techniques they generally fail, if clusters differ significantly in shape or size. We will take a closer look at two representative algorithms and their qualities.

C-Means. The basic idea of the *C-means method* is to join an object obj_i to a cluster $clust_j$ if the distance between the position $\mathbf{x}_i$ of the data object obj_i and the center $\mathbf{c}_j$ of the cluster $clust_j$ is less than a threshold value δ:

$$|\mathbf{x}_i - \mathbf{c}_j| \leq \delta \tag{7.1}$$

The center position $\mathbf{c}_j$ of cluster $clust_j$ is defined by the arithmetic average of the positions of all data objects $\mathbf{x}_i$ enclosed by cluster $clust_j$

$$\mathbf{c}_j = \frac{1}{N} \cdot \sum_{i=1}^{N} \mathbf{x}_i \tag{7.2}$$

where N designates the number of data objects within the current cluster.

The C-means algorithm iterates over all data objects obj_i and verifies for each object obj_i if there exists a cluster $clust_j$ the center $\mathbf{c}_j$ of which is closer to $\mathbf{x}_i$ than δ. If there are such clusters the object will be added to the cluster that is closest to the object. Otherwise a new cluster is generated with the object x_i as its only member. After assigning the object to a cluster, its center position will be updated, i.e. the center will shift.

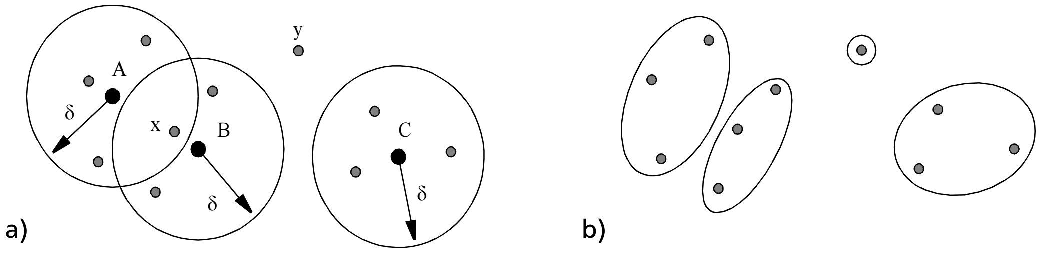


Figure 7.2 a) Partitioning using C-means method with threshold δ, where the assignment of object *x* is undetermined. Object *y*, on the other hand, could not be assigned to any existing cluster. Therefore, it generates a new one. b) Completely clustered scene.

A major disadvantage of the C-means method is the user defined selection of the cluster threshold value δ. In some cases, the determination of a proper value for δ may be very difficult. With too large a value clusters will contain objects which do not correspond. On the other hand, too small a value will result in clusters each holding only one single object. Another drawback is the sensitivity of the algorithm to the order of traversal of the given objects. In particular, the choice of the starting object has a great influence on the resulting cluster distribution.

The cost of the C-means algorithm is of order $O(n^2)$ being defined by the worst case scenario, with each object located in its own cluster. But due to the very simple operations the C-means method relies on, it is very fast in general. A pseudocode fragment is given below.

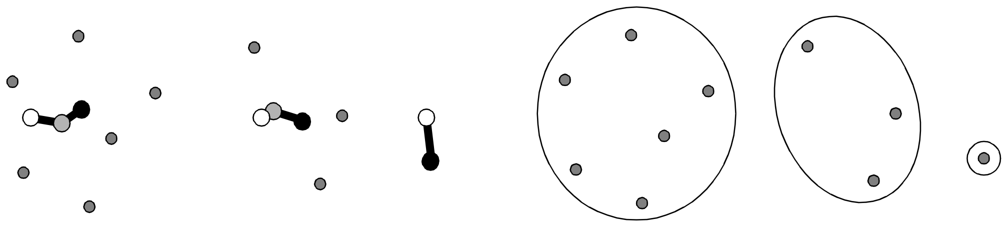
Table 7.1 Pseudocode listing of the C-mans algorithm

```
$O=\{O_1,..,O_n\}$ // initial object list; $r_i$ = position of $O_i$
$K=\{\}$           // set of clusters; $c_j$ = centroid of $K_j$
while (O not empty) do
  fetch object $O_i$ from O | minima = delta
    for each cluster $K_j \in K$ do
      if ($|c_j - r_i|$ < minima) then
        $K_{min} = K_j$
        minima = $|c_j - r_i|$
      fi
    od

    if (minima < delta) then
      add $O_i$ into $K_{min}$ | update $c_{min}$
    else
      create new cluster $K_{new}$
      add $O_i$ into $K_{new}$ | $c_{new} = r_i$
      add $K_{new}$ into K
    fi
  od
end
```

K-Means. K-means belongs to the class of iterative clustering techniques. Choosing the *K-means method* we have to preselect the number k of clusters that the algorithm will generate.

First k initial cluster centers are defined. An object obj_i is assigned to the cluster $clust_j$ when its center $\mathbf{c}_j$ is closest to the object position $\mathbf{x}_i$. In such a way, all objects are associated to exactly one cluster. At the beginning of the next iteration, the cluster centers $\mathbf{c}_j$ of all k clusters are updated to the arithmetical average of all positions $\mathbf{x}_i$ of associated objects. Thereafter, another assignment round starts using the recently computed cluster centers. The iteration loop stops if all cluster centers have converged into a stable position.



a) ○ 1. step ◎ 2. step ● 3. step

b)

Figure 7.3 The same scene as shown in Figure 7.2 clustered with the K-means algorithm a) The iteration steps for the 3 cluster centroids. b) Resulting clustered layout.

The K-means method poses a problem concerning the selection of the initial positioning of the k clusters. An unlucky choice may have an important influence on the resulting object clustering.

K-means' iterative behavior and the apriori unknown number of iterations makes the cost estimation more difficult than for the C-means algorithm. In each step, the algorithm calculates the distances between all n object and the k cluster centers, i.e. calculates nk distances. Since k is constant, the costs are of order $O(n)$ per iteration step.

7.2.2 Hierarchical Methods

Hierarchical clustering methods (HCM) are commonly used in the area of information visualization and data mining. In contrast to partitioning clustering methods, which subdivide a set of objects into a certain number of clusters, hierarchical clustering generates a nested sequence of partitions. We call this a *cluster tree* (as shown in Figure 7.4).

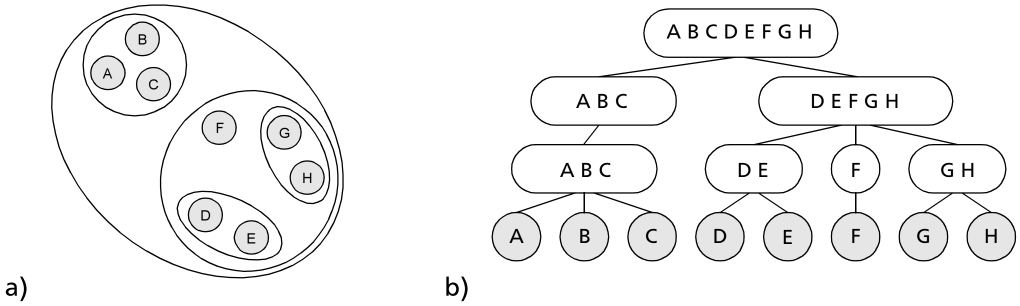


Figure 7.4 a) Probable object arrangement with 8 objects. b) Corresponding cluster tree with 4 levels generated by an agglomerative, hierarchical clustering algorithm

An *agglomerative hierarchical clustering* algorithm starts with n atomic clusters, each containing exactly one object. At each step, the algorithm merges the two most similar[1] clusters and thus decreases the total number of clusters by one. These steps recur until only one single cluster, containing all objects, remains. Any two clusters generated by such a procedure are either nested or disjoint. In contrast, *divisive hierarchical clustering* reverses the process by starting with a single cluster holding all objects and subdividing it into smaller sets [JD88].

Many variants of agglomerative hierarchical clustering methods are known, mainly differing in the definition of the metric applied in updating the similarity between existing and merged clusters.

Along with the incremental algorithms mentioned above, there is a group of non-incremental clustering methods (e.g. CLUSTER/S [SM86]). The discussion of those algorithms is beyond the scope of this chapter, and their methods are not considered in the following.

In the remainder of this section we shall discuss two different agglomerative hierarchical clustering methods: the *single linkage method* and the *complete linkage method*. For an in-depth description we refer to [Zup82].

Single Linkage Method. Another straightforward and quick clustering technique is called *single linkage method* (SLM) or *nearest neighbor technique*. For this algorithm we define the distance between two clusters as the minimal spacing between two arbitrary objects, each located in two different clusters. Assuming that d_{ij} is the distance between

---

[1] In the current context similarity of two objects is defined by the inverse of their distance. Thus the algorithm merges the two closest clusters in each step.

object obj from cluster $clust_i$ and object obj' from cluster $clust_j$. Then, the distance D_{ij} between clusters $clust_i$ and $clust_j$ is defined as

$$D_{ij} = min\{d(obj, obj') \mid obj \in clust_i, obj' \in clust_j\} = min(d_{ij}). \qquad (7.3)$$

That means we measure distances between two clusters as the distance of the closest pair of objects each belonging to a different cluster. The SLM synthesizes clusters analogous to the general description found at the beginning of this section.

A problem of SLM is the algorithm's tendency to generously accept object chains as clusters. Assume we have an object configuration like the one shown in Figure 7.5. The SLM would string objects between A and B to a chain. Thus, objects A and B will be assigned to the same cluster. SLM generates three clusters (drawn with a solid line). Building only two clusters (shown with a dotted line) would be a superior solution.

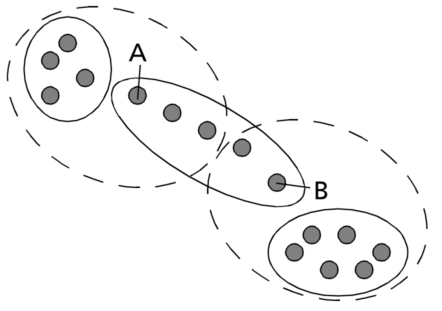


Figure 7.5 Generation of chains applying the single linkage method

Unlike centroid-based algorithms, this method could discover clusters of arbitrary shape and different size. Unfortunately, the procedure is highly susceptible to noise and outliers.

To build up the cluster tree, the single linkage method has to compute the distance in pairs between every two objects, i.e. supposed we have n objects, we have to perform $n \cdot (n-1)/2$ distance evaluations per iteration, which dearly is of order $O(n^3)$ over all n iteration steps.

Complete Linkage Method. Another clustering method, the *complete linkage method* (CLM), takes into account the chain formation and defines the distance between two clusters D_{ij} as the maximal distance between two of their objects

$$D_{ij} = max\{d(obj, obj') \mid obj \in clust_i, obj' \in clust_j\} = max(d_{ij}) \qquad (7.4)$$

Supposed we run the CLM on an object topology that already contains two shorter cluster chains, the distance between the two clusters is now defined by the two furthest away objects not located in the same cluster. This is equal to the distance of the outermost object on the one side of a chain and the outermost object on the other side of the other chain. Thus, chain formation is suppressed.

As mentioned at the beginning of this section, there are many other well known clustering algorithms, i.e. BIRCH [ZRL96], which is basically an extension of the K-means clustering, but adequately addresses the problem of large data sets. CURE [GRS98] remedies the drawback of single centroid representation by taking advantage of a multi-centroid representation of clusters. Hence this algorithm is more robust to outliers and identifies clusters varying in size and having non-spherical shapes. A recent approach is called CHAMELEON [KHK99], a hierarchical clustering algorithm that measures intercluster similarity based on a dynamic model. In addition to other algorithms, CHAMELEON clustering is based not only on vicinity of objects but also considers corresponding connectivity information. This combination results in a robust handling of data that consists of clusters being of different shape, size or density.

7.3 VISUAL CLUSTERING METHODS

Powerful visual clustering mechanisms to simplify the viewable structure of complex subregions are essential to provide an efficient level-of-detail strategy.

There is quite a large number of algorithms and systems treating the subject of cluster visualization. Virtually all of them take the problem of cluster visualization simply as a layout problem, thus focusing on optimizing the computation and spatial grouping of crowds of single data objects. The visualization then is limited to drawing just a simple shape (dot, icon, glyph, etc.) for each data object (shown in Figure 7.6a). Thus, the actual visual clustering process is rather done by the user's perceptual system than by the visualization system itself.

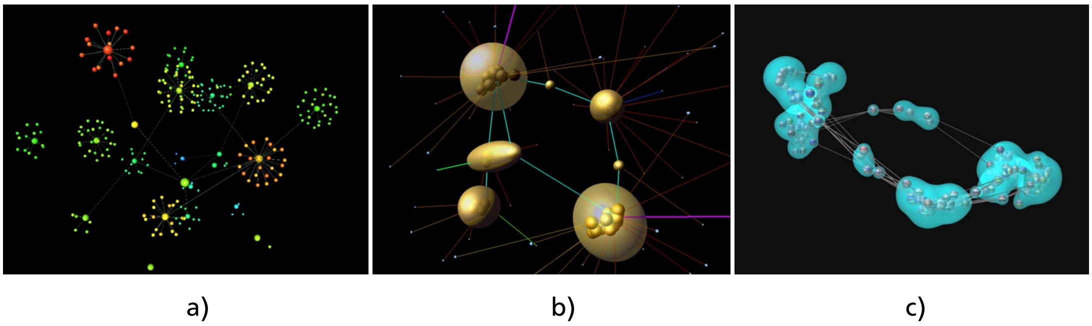

a) b) c)

Figure 7.6 Different techniques to visualize clusters of data objects. a) cluster represented by a cluttered group of single objects b) visualization with ellipsoidal surfaces wrapped around clusters c) objects visually combined by a BLOB surface (see Color Plate 4 on page 184).

There are two reasons to go a step further: first today's graphics hardware – though current progress in this area is tremendous – is not yet ready for the data volumes we would like to address with present data management systems (i.e. data warehouses). Second, the user's perceptual system should be relieved of gathering single points into a cluster object. In order to speed up the decision making process and to increase the decision's quality, cluster visualization has to take the step to the next higher level of visual representation.

Only a few approaches make an effort in this direction. Some of the systems attempt to break down complexity by running a preclustering algorithm on the initial data set. Afterwards the system confines itself to displaying only objects on a chosen clustering level, where clusters are represented by a simple shape at the position of their centroids. Doing so, we lose most of the information contained in a cluster. Only the cluster's position is visible to the user. Information about the internal object distribution, including size, orientation and variation is visually not available to the user.

Initial work about a more powerful visualization method is reported in [Hen+95], where wrapping hyperspheres accomplish the clustering of data objects. Building on this technique, we propose a PCA-based technique (Section 7.3.1) where the basic idea is to wrap ellipsoids around each object group whose shape is controlled by the principal components of the respective cluster (shown in Figure 7.6b). In either of these approaches the restriction to a quadric surface representation of the clustering hull represents an unnecessary restriction. The internal object distribution is only roughly approximated, as well in size as in orientation. This drawback gets addressed by an algorithm called BLOB-clustering (Section 7.3.2), the fundamental idea of which is to use *blob functions* combined with a *marching cube* [Blo94] algorithm to represent the enfolding cluster surface (see Figure 7.6c). The generated shape represents the distribution of the included data objects in the best possible manner. All of the cluster visualization methods mentioned above are limited to work on partitions only. None of them takes advantage of the hierarchical information cluster structures inherently contain. Therefore, we extend the BLOB-based approach in order to group and visualize cluster hierarchies at multiple levels-of-detail. This method is called H-BLOB clustering (Section 7.3.3).

7.3.1 Ellipsoid Clustering

In order to simplify the geometry and topology of complex object arrangements it is necessary to provide an efficient level-of-detail strategy. Initial work for information visualization is reported in [Hen+95] where simple clustering is done by wrapping hyperspheres around groups of objects. The transparency of the hyperspheres was controlled as a function of the distance to the viewer. Unlike this approach we propose a K-means and *principal component analysis* (PCA) based clustering mechanism which will be explained in the upcoming section.

The basic idea is to wrap ellipsoids around each cluster whose shape is controlled by the principal components of the respective cluster. The method is designed as a two pass procedure, where in a first step all objects in the scene are divided into a set K of disjoined subsets using one of the partitioning methods presented in Section 7.2.1.

The second pass comprises the parametrization of an affine map which transforms the initial 3D spherical shape appropriately into the scene. For a given cluster from K this transform is defined by a translation vector $\mathbf{c}$, a scaling matrix $\mathbf{S}$ and a rotation matrix $\mathbf{R}$. Thus, the transformation is figured out by the following set of equations. We start from the implicit equation of the unit sphere with surface vector

$$\mathbf{x}^0 = (x, y, z) \text{ such that } \mathrm{x}^2 + y^2 + z^2 = 1 \tag{7.5}$$

and perform a subsequent affine mapping by

$$\mathbf{x}^e = \mathbf{c} + \mathbf{R} \cdot \mathbf{S} \cdot \mathbf{x}^0 , \tag{7.6}$$

where

$\mathbf{x}^e$ ...... surface vector of the ellipsoid

Subsequently we describe how to compute the translation, the scaling and the orientation of an ellipsoid that encloses a set of N points $\mathbf{x}_i = (x_i, y_i, z_i)$ in space.

Translation of the Ellipsoidal Surface. The center $\mathbf{c} = (c_x, c_y, c_z)$ of the ellipsoid can be obtained immediately as the arithmetic average of the position of the points in space:

$$\mathbf{c} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{m}_i \qquad (7.7)$$

Scaling and Orientation of the Ellipsoidal Surface. The scaling and orientation of the ellipsoid can be computed with the covariance matrix $\mathbf{C}$ of the set of points. The matrix $\mathbf{C}$ is defined as follows:

$$\mathbf{C} = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{x}_i - \mathbf{c})(\mathbf{x}_i - \mathbf{c})^{T}$$

$$\mathbf{C} = \frac{1}{N}\sum_{i=1}^{N} \begin{bmatrix} x_i - c_x \\ y_i - c_y \\ z_i - c_z \end{bmatrix} \cdot \begin{bmatrix} x_i - c_x & y_i - c_y & z_i - c_z \end{bmatrix} \qquad (7.8)$$

DEFINITION. A value σ is called *eigenvalue* of a $N \times N$ matrix $\mathbf{M}$, if a vector $\boldsymbol{u}$ exists, such that $\mathbf{M} \cdot \mathbf{u} = \sigma \cdot \mathbf{u}$. The vector $\mathbf{u}$ is defined as the *eigenvector* of the *eigenvalue* σ. A matrix of dimension N has exactly N eigenvalues and N eigenvectors.

By solving the eigenproblem $\mathbf{C} \cdot \mathbf{u}_k = \sigma_k \cdot \mathbf{u}_k$ we then compute the 3 eigenvalues $\sigma_1, \sigma_2, \sigma_3$ and the associated eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ which define the required transformation matrices. Note, since the eigenproblem is of dimension 3x3 it can be solved analytically.

The scaling matrix $\mathbf{S}$ is thus constructed using the three eigenvalues σ_1, σ_2 and σ_3 of the matrix $\mathbf{C}$:

$$\mathbf{S} = \begin{bmatrix} \sqrt{\sigma_1} & 0 & 0 \\ 0 & \sqrt{\sigma_2} & 0 \\ 0 & 0 & \sqrt{\sigma_3} \end{bmatrix} \qquad (7.9)$$

The orientation of the ellipsoid is specified by the rotation matrix $\mathbf{R}$, which is defined by the eigenvectors $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$ of the covariance matrix $\mathbf{C}$:

$$\mathbf{R} = \begin{bmatrix} u_{1x} & u_{2x} & u_{3x} \\ u_{1y} & u_{2y} & u_{3y} \\ u_{1z} & u_{2z} & u_{3z} \end{bmatrix} \qquad (7.10)$$

The three eigenvalues of the covariance matrix $\mathbf{C}$ defined in Equation 7.8 are evaluated by setting the determinant of the matrix $\mathbf{C} - \sigma \cdot \mathbf{I}$ to zero and solving the resulting system for σ:

$$\|\mathbf{C}-\sigma\cdot\mathbf{I}\| = \left\|\begin{bmatrix} C_{00}-\sigma & C_{01} & C_{02} \\ C_{10} & C_{11}-\sigma & C_{12} \\ C_{20} & C_{21} & C_{22}-\sigma \end{bmatrix}\right\| = 0 \tag{7.11}$$

Equation 7.11 can be interpreted as a polynomial $P(\sigma)$ of degree three in the unknown σ. The three solutions σ_1, σ_2 and σ_3 of $P(\sigma) = 0$ correspond to the eigenvalues of the matrix $\mathbf{C}$.

Finally, the three eigenvectors $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$ of $\mathbf{C}$ can be computed from the eigenvalues of $\mathbf{C}$ as:

$$\begin{bmatrix} C_{00}-\sigma_1 & C_{01} & C_{02} \\ C_{10} & C_{11}-\sigma_2 & C_{12} \\ C_{20} & C_{21} & C_{22}-\sigma_3 \end{bmatrix}\cdot\mathbf{u}_i = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad \text{for} \quad i = 1\ldots3 \tag{7.12}$$

Due to the statistical properties of the principal components it is not guaranteed that all objects of a cluster are enclosed by the ellipse. Therefore we carry out additional postprocessing and adjust the scaling of the ellipsoidal hull until all objects are enclosed.

Adjustment of the scaling. In order to adjust the scaling matrix $\mathbf{S}$ it must be tested whether all the points $\mathbf{x}_i$ lie in the interior of the ellipsoid. If this is not the case, then it is necessary to adjust the scaling of the ellipsoid. To make the computations easier, the center of the ellipsoid must be set in the origin of the coordinate system. This is accomplished by translating and rotating all the points $\mathbf{x}_i$ using the position and orientation of the ellipsoid. Figure 7.7 illustrates this process with an example.

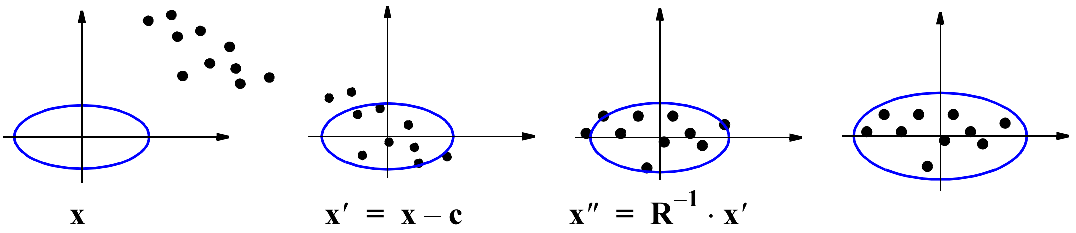


Figure 7.7 Adjustment of the scaling using a transformed point set, which is centered on the origin

The scaling factor τ can be computed from the new coordinates (x, y, z) of all the points $\mathbf{x}_i$ and the three eigenvalues σ_1, σ_2 and σ_3 of the covariance matrix $\mathbf{C}$ as:

$$\frac{x^2}{\sigma_1} + \frac{y^2}{\sigma_2} + \frac{z^2}{\sigma_3} = \tau \tag{7.13}$$

The position of a point $\mathbf{x}_i$ relative to the ellipsoid can be extracted from the magnitude of τ:

$\tau < 1$	The point lies in the interior of the ellipsoid
$\tau = 1$	The point lies on the surface of the ellipsoid
$\tau > 1$	The point lies outside the ellipsoid

If $\tau > 1$, then the values σ_1, σ_2 and σ_3 must be increased, in order to meet the condition $\tau \leq 1$.

The strategy to modify the values was constructed experimentally. The idea behind this method was to make the three semi-axis of the ellipsoid equally large, thus letting the ellipsoid approximate a sphere. If after this transformation the value of τ is still larger than one, the semi-axis are scaled by a factor of τ, so that the point is guaranteed to lie on the surface of the ellipsoid. This strategy can be implemented in four the steps shown below:

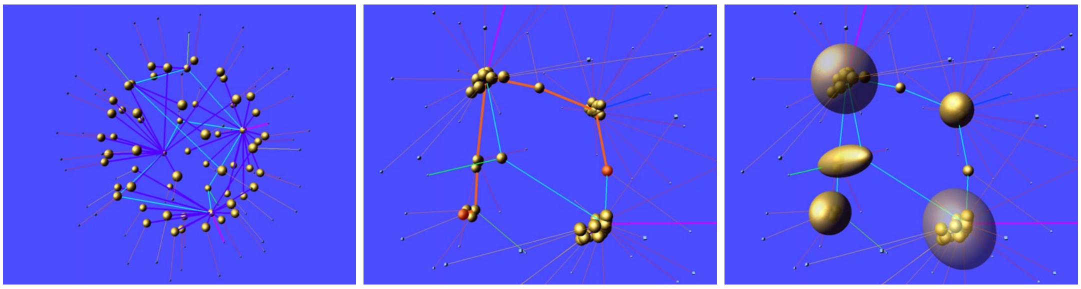
- increase the smallest semi-axis σ_{min} up to the largest semi-axis σ_{max} and check whether the condition $\tau \leq 1$ can be met.
- Increase the medium semi-axis σ_{mid} up to the largest semi-axis σ_{max} and check whether the condition $\tau \leq 1$ can be met.
- Increase both the smallest semi-axis σ_{min} and the medium semi-axis σ_{mid} up to the largest semi-axis σ_{max} and check whether the condition $\tau \leq 1$ can be met.
- If none of the previous steps was successful, re-scale all three semi-axis σ_{min}, σ_{mid}, and σ_{max} by a factor of τ.

Table 7.2 shows a pseudo code implementation of this strategy.

Table 7.2 Pseudocode listing of the ellipsoid scaling adjustment

```
for each point $x_i$=(x,y,z) of the cluster do
  compute $\tau$
  if ($\tau$ > 1) then
    //sort eigenvalues $\sigma_1$, $\sigma_2$ and $\sigma_3$
    determine $\sigma_{min}$, $\sigma_{mid}$ and $\sigma_{max}$
    // sort coordinates according to eigenvalues
    determine $x_{min}$, $x_{mid}$ and $x_{max}$

    if ($x^2_{min}/\sigma_{max}+x^2_{mid}/\sigma_{mid}+x^2_{max}/\sigma_{max}$ <= 1) then
      $\sigma_{min}=x^2_{min}/(1-x^2_{mid}/\sigma_{mid}-x^2_{max}/\sigma_{max})$
    elsif ($x^2_{min}/\sigma_{min}+x^2_{mid}/\sigma_{max}+x^2_{max}/\sigma_{max}$ <= 1) then
      $\sigma_{mid}=x^2_{mid}/(1-x^2_{min}/\sigma_{min}-x^2_{max}/\sigma_{max})$
    elsif ($x^2_{min}/\sigma_{max}+x^2_{mid}/\sigma_{max}+x^2_{max}/\sigma_{max}$ <= 1) then
      $\sigma_{min}=x^2_{min}+x^2_{mid}/(1-x^2_{max}/\sigma_{max})$
      $\sigma_{mid}=\sigma_{min}$
    else
      $\sigma_{min}=\sigma_{min}*\tau$
      $\sigma_{mid}=\sigma_{mid}*\tau$
      $\sigma_{max}=\sigma_{max}*\tau$
    fi
  fi
od
```



a) b) c)

Figure 7.8 Illustration of the clustering method: a) Initialization, b) Model after relaxation with a highlighted minimal path between two objects, c) Disjoined clusters as transparent ellipsoids

7.3.2 BLOB Clustering

The fundamental idea is to use blob functions to represent individual objects and clustering shapes [GSF97]. Blobs, as initially been introduced to the graphics community by [Bli82] allowing us to represent complex geometric shapes by implicit formulations. Here, a given volume is expanded by placing individual 3D basis functions $f_i(x, y, z)$ – so-called blobs – at some spatially scattered object locations (x_i, y_i, z_i). Introducing a set of weights a_i and b_i features an additional degree of freedom where the volume function $f(x, y, z)$ is obtained by the following linear expansion:

$$f(x, y, z) = \sum_{i=1}^{n} f_i(x, y, z) \tag{7.14}$$

An implicit definition of a 3D clustering isosurface is determined by selecting an isovalue c_{iso} with

$$f(x, y, z) = c_{iso} \tag{7.15}$$

Note that for a physical interpretation the 3D basis function f_i can be considered describing some scalar field or potential. An appropriate choice of the bases is required for efficient handling of the blobs. In most cases, Gaussians or cubic splines provide good results. In a Gaussian representation of the field we have:

$$f_i(x, y, z) = b_i \cdot e^{-a_i \cdot g_i(x, y, z)} \tag{7.16}$$

where a_i and b_i stand for individual scalar weights. For the exponent function g_i we adapted an approach proposed by [Mur91] who essentially used a superquadric as

$$g_i(x, y, z) = \left((x - x_i)^{2/\nu_i} + (y - y_i)^{2/\nu_i}\right)^{\nu_i/\mu_i} + (z - z_i)^{2/\mu_i} \tag{7.17}$$

where ν_i and μ_i are parameters to control the smoothness of the resulting shape. This formulation allows a maximum degree of freedom in modeling the implicit shape.

The effects of implicit clustering are illustrated by the picture series of Figure 7.9. Two initially disjoint isotropic (Gaussian) fields are clustered by an implicit isosurface which is computed by superimposing the associated basis functions. In the rightmost picture of the

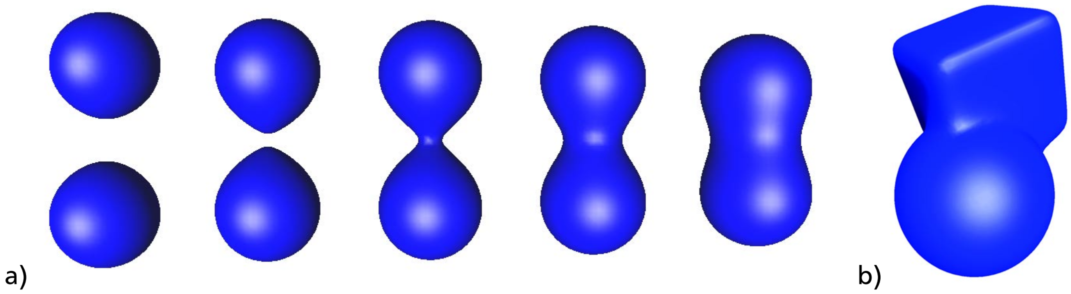


Figure 7.9 a) Illustration of implicit surfaces obtained from two field functions as their centroids are approaching. b) Modification of the shape by setting $\nu = \mu = 0.25$ for the upper field function (see Color Plate 1 on page 183).

series the shape of the upper basis function was modified by tuning the superquadric parameters ν and μ, respectively. This property enables adapting individual clustering shapes to the underlying application scenario.

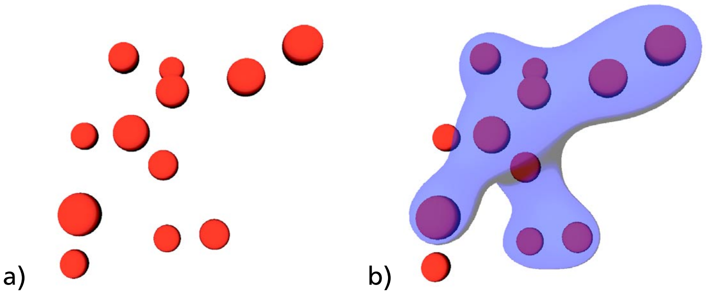


Figure 7.10 Clustering of a subset of objects scattered in 3D space: a) initial configuration, b) clustering isosurface as a transparent hull wrapped around the desired objects (see Color Plate 2 on page 183).

A more complex clustering problem is depicted in Figure 7.10 where a subset of scattered objects was wrapped by blob clustering using the approaches described above. We observe that the implicit shape resulting from the individual field functions smoothly encapsulates all desired objects. Unlike with PCA based clustering, almost no space is wasted.

Yet, a problem arising with clustering is an undesired intersection of the hull with individual objects outside the cluster, such as presented in Figure 7.11a. Here, an interesting property of the blob approach helps to solve the problem. By assigning a negative value to the weight b_i we generate an inhibiting field and the clustering hull is pushed away from the object (Figure 7.11b). Isosurfaces can be computed at interactive rates using appropriate techniques.

However, the blob approach from above essentially performs a visual clustering. The automatic identification of subsets of objects has to be computed by additional procedures upon isosurface reconstruction.

We propose to modify an implicit surface polygonizer, such as the one presented in [Blo94]. The following algorithm computes a semantic encapsulation of objects into sets of individual clusters under the following restrictions:

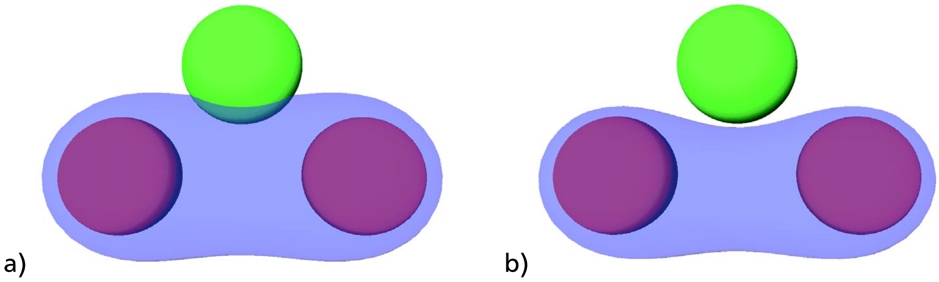


Figure 7.11 a) Undesired intersection of object and hull. b) repulse of the clustering surface from the object by assigning a negative value for b (see Color Plate 3 on page 183).

- *disjointness:* objects may belong to one and only one cluster. Multiple assignments are prohibited.
- *non-intersection of clusters:* clusters must not intersect. As illustrated in Figure 7.11, undesired intersections can be resolved using negative weights

The above preconditions guarantee a partitioning of objects into disjoint clusters encapsulated by disjoint isosurfaces. In this case the object belongs to the cluster whose surface is intersected first when traveling on a ray from the object center outwards. A pseudocode segment of the respective volume-based clustering algorithm is shown in Table 7.3.

Table 7.3 A pseudocode segment of the volume-based clustering algorithm

```
Input($c_{iso}$)

For (all DataObjects O) do
  Search(first cell V intersected by cluster C, $c_{iso}$)
  if(V already computed)
    // Object O belongs to cluster C
    add(O to children-list of C)
  else
    // Traverse recursively to compute isosurface
    traverse(surface C starting with V, $c_{iso}$)
  fi
od
```

The algorithm starts from the center of a given object, and after discretization of the field function it searches the nearest volume cell intersecting the isosurface defined by c_{iso}. If the isosurface segments (triangles) of this voxel have already been computed, the surface segment exists and the object is added to the corresponding cluster. If not, the clustering hull (isosurface) is initialized and computed recursively with the cell as a seeding point.

Note that this method requires to store the visited cells. For efficient implementation, we recommend to build a hash table with keys computed from cell coordinates.

7.3.3 H-BLOB Clustering

All of the cluster visualization methods mentioned above are limited to work only in combination with *partitioning clustering algorithms*. None of them takes advantage of the hierarchical information cluster structures inherently contain. Therefore, we propose a new simple and fast clustering technique that has its strength in the visualization of hierarchical clustering structures, that is to say *cluster trees*.

In this section, we present a new hierarchical clustering and visualization algorithm called *H-BLOB (Hierarchical-BLOB)*, which groups and visualizes cluster hierarchies at multiple levels-of-detail [SBG00]. Our method is fundamentally different from conventional clustering algorithms, such as C-means, K-means, or linkage methods which are primarily designed to partition a collection of objects into subsets sharing similar attributes. These approaches usually lack an efficient level-of-detail strategy that breaks down the visual complexity of very large data sets for visualization. In contrast, our method combines grouping and visualization in a two stage process constructing a hierarchical setting. In the first stage a cluster tree is computed making use of an analytical clustering process. Exploiting the inherent hierarchical structure of this tree, a second stage visualizes the clusters by computing a hierarchy of implicit surfaces.

The H-BLOB algorithm is considered to be a direct derivative of the BLOB clustering method presented in Section 7.3.2, extended by the capability to handle hierarchical settings. In fact, it is a combination of various techniques and algorithms described in preceding sections, each one applied to a suitable subtask according to its strengths.

Stage I: Edge Collapse Clustering. Inspired by the persuasive idea of the *edge collapsing algorithm* presented in [Hop96], we propose a new simple and efficient clustering method, called *edge collapse clustering* (ECC).

The algorithm we present, belongs to the category of *agglomerative hierarchical clustering* methods. Thus, the general structure is very similar to the methods presented in Section 7.2.2.

In contrast to the linkage methods the ECC bases on centroids; hence, it only works in *coordinate space*. We define the distance D_{ij} between two clusters $clust_i$ and $clust_j$ as the distance between their centroids $\mathbf{c}_i$ and $\mathbf{c}_j$

$$D_{ij} = \|\boldsymbol{c}_i - \boldsymbol{c}_j\| . \tag{7.18}$$

The process of cluster merging works analogously to the process shown in Section 7.2.2, but with the following extension:

Each cluster $clust_i$ is equipped with a weight w_i corresponding to the number of objects contained in $clust_i$. The weight w_i is initialized with a value of one. With each iteration, the algorithm merges the two closest clusters, i.e. the pair of clusters with minimal distance D_{ij}, into a new one, called $clust_{new}$ with centroid $\mathbf{c}_{new}$. At the same time, the parameters of the new cluster are updated corresponding to the formulas below:

$$\boldsymbol{c}_{new} = \frac{\boldsymbol{c}_i w_i + \boldsymbol{c}_j w_j}{w_i + w_j} \tag{7.19}$$

$$w_{new} = w_i + w_j \tag{7.20}$$

If the two clusters are of different weight, the new cluster will be located closer to the heavier, i.e. larger cluster, which is desirable in practice.

Figure 7.12 illustrates the algorithm by means of an example with 5 objects spread on a plane. Each iteration step is shown on a separate line, with the actual object arrangement in the left half and the current cluster tree on the opposite side. Starting with 5 single objects, the ECC algorithm merges them into a single cluster after the same number of iteration steps. The thick line, highlights the edge to be collapsed next.

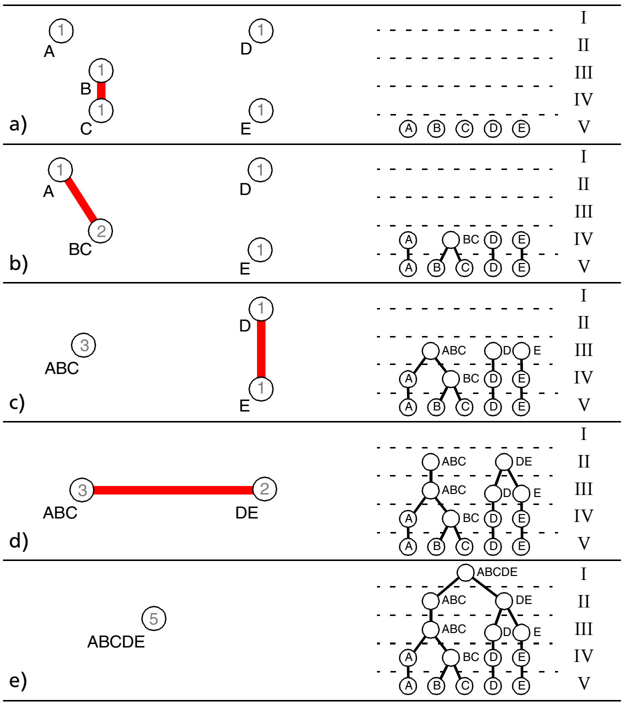


Figure 7.12 a) - e) Progressive edge-collapse algorithm. Red line indicates edge to be collapsed next. Current cluster tree levels (I-V) are shown on the right-hand side.

Since each cluster is defined by its centroid only and as the distance metric depends only on the centroid's coordinates, every two clusters are virtually interconnected with exactly one edge of length D_{ij}. Consequently, ECC takes advantage of the inherent hierarchical structure of a cluster tree. The computational complexity for each iteration step is defined by the corresponding number of clusters. This is an advantage compared to the linkage algorithms, which always operate on the initial set of all single objects. Hence, the ECC algorithm is computationally less complex than linkage methods.

The disadvantages concerning the fragile user-driven parameter preselection of the C- and K-means methods do not apply for ECC. Although this technique is partly based on centroids, it is more stable with respect to unconstrained shapes and different cluster sizes than C- and K-means. The undesirable effect of chain formation does not occur for ECC.

Unfortunately, the ECC is still in the same polynomial order as the linkage techniques. It also preforms n iterations steps and computes in each of the steps $n \cdot (n-1)/2$ distances. Since ECC computes distances based on centroids we get a triangular cost scheme over all iterations, which results in an complexity of order $O(1/6 \cdot n^3 - 1/6 \cdot n)$ regarding the number of computed distances.

Stage II: Cluster Tree Visualization. The cluster tree generated as a result of the first stage must now be visualized. Each hierarchy level should be handled separately, i.e. we compute a separate surrounding surface for each cluster at a specific hierarchy level.

As a basic idea we devote resources to the BLOB algorithm described in [GSF97]. The fundamental idea of BLOB clustering is to give each object a spatial extension by attaching a spherical *primitive* to its center. In general a *primitive* is a working model comprising a parameterized oriented shape and a corresponding 3D field function $f_i(x, y, z)$. Primitives and their parameterization will be explained in more detail in the next section.

To compute a BLOB surface, we superimpose all field functions $f_i(x, y, z)$ in space and accordingly run a *marching cube algorithm* [Blo94] to extract the implicit surface at a given *isovalue*. The subsequent sections explain how we extend this algorithm in order to handle hierarchical cluster structures efficiently.

Visualization using BLOBS. As a straightforward approach to visualize a single cluster on a given cluster level, we could assume a scenario where a primitive is attached to each of the cluster's objects. Supposed we choose a skillful parameterization of those primitives, we could obtain an isosurface that fully encloses all objects and the visualization problem would be superficially solved.

Even if this approach results in fair visual results, it has a tremendous handicap. For very large clusters holding a huge number of single objects the computational cost rises excessively. That effect occurs because in order to perform an isosurface extraction we have to evaluate the superimposed field at given points in space which involves the evaluation of the field equation for every single primitive. The problem could be alleviated if we would be able to find a way to limit the number of primitives during visualization.

We consider the cluster tree shown in Figure 7.13, which is subdivided into 3 hierarchical levels. The topmost cluster on level I contains all 5 objects (ABCDE). If we intend to visualize this cluster, we have to take into account five different primitives – one for each object.

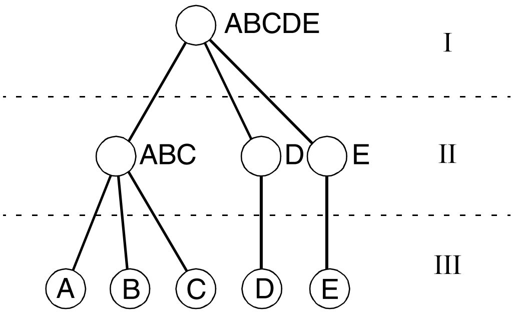


Figure 7.13 Cluster tree with three levels. It is a condensed view on the corresponding tree shown in Figure 7.12e without displaying level II and IV.

To limit the number of primitives we propose the following approach: instead of attaching primitives to every single object, we just consider the objects one level below the level of interest. Thus, in order to visualize the cluster in level I we attach primitives to the level II cluster objects, i.e. to the clusters (ABC), (D) and (E). Or, if we aim to visualize clusters of level II, we utilize cluster objects from level III and so forth.

For satisfactory results, we need to extend the characteristics of the primitives used, which – in the original BLOB paper [GSF97] – were restricted to be of radial symmetric shape. This is due to the fact that in contrast to the previous BLOB clustering algorithm primitives now have to account for the properties of a whole object set rather than of only one single object. We suggest the extension of our concept of a primitive to an elliptical feature, the so called *ellipsoidal primitive*. The following sections will give a more exact definition.

Extension to Ellipsoidal Primitives. *Ellipsoidal primitives* are a direct extension to the common primitives determined in [GSF97]. The characteristics of an ellipsoidal primitive is specified by an ellipsoidal shape and the field function f_i. For the definition of the shape and the computation of its size, orientation and position we refer to Section 7.3.1. The definition of f_i is

$$f_i(x, y, z) = \begin{cases} b_i & \text{if } (x, y, z) \text{ lies inside ellipsoid} \\ b_i \cdot e^{-a_i \cdot g_i(x, y, z)^2} & \text{otherwise} \end{cases} \tag{7.21}$$

where $g_i(x, y, z) = d_i(x, y, z)$ is the distance to ellipsoidal surface, b_i defines the maximal magnitude of the function inside the ellipsoid, and a_i influences the descent of the field function.

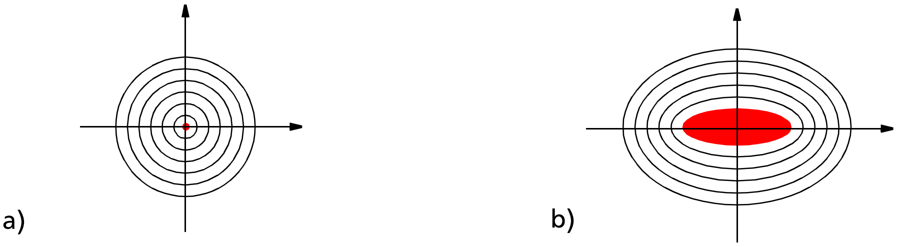


Figure 7.14 Isolines of a) a spherical, symmetric primitive and b) a new ellipsoidal primitive.

Figure 7.14 compares the fields of a spherical symmetric primitive to the field of a new ellipsoidal primitive defined by Equation 7.21 on the basis of their isolines. Inside the red area the field has a value of b_i.

The field f_i of a single ellipsoidal primitive could be described as follows: for all points inside the ellipsoid the value of the field is uniformly b_i. Starting at the surface of the ellipsoid the field descents exponentially and monotonously as a function of the distance to the surface.

Computation of Ellipsoidal Gaussian Fields. An ellipsoid is defined by its scaling matrix S, its rotation matrix R and its center $\boldsymbol{m}$. From the diagonal elements of the scaling matrix result the three half axes H_a, H_b and H_c.

Transforming the ellipsoid into the origin will simplify subsequent formulas. In order to compute the value of the field function f_i at a point $\boldsymbol{p} = (x, y, z)$ from Equation 7.21, the coordinates of $\boldsymbol{p}$ have to be transformed: first, $\boldsymbol{p}$ is translated by the negative values of vector $\boldsymbol{m}$ according to

$$\boldsymbol{p}' = \boldsymbol{p} - \boldsymbol{m}. \tag{7.22}$$

Then, $\boldsymbol{p}'$ is rotated by the inverse rotation matrix R:

$$\boldsymbol{p}'' = R^{-1} \cdot \boldsymbol{p}' = (x'', y'', z'')^T \tag{7.23}$$

To gather the distance between the transformed point $\boldsymbol{p}''$ and the surface of the ellipsoid, it is necessary to intersect the connecting line between the center of the ellipsoid – which is equal to the origin – and the point $\boldsymbol{p}''$ with the ellipsoidal surface. To this aim the line $\overline{\boldsymbol{Op}''}$ is parametrized with t running from 0 to 1.

$$\boldsymbol{p}_t(t) = (x_t, y_t, z_t)^T = (t \cdot x'', t \cdot y'', t \cdot z'')^T \tag{7.24}$$

A point $\boldsymbol{p}_t$ is located on the surface of the ellipsoid, if the ellipsoidal equation evaluates to 1:

$$\frac{x_t^2}{H_a} + \frac{y_t^2}{H_b} + \frac{z_t^2}{H_c} = 1 \tag{7.25}$$

Substituting Equation 7.24 into Equation 7.25 yields for the intersection point t_s:

$$t_s = \sqrt{\frac{H_a^2 \cdot H_b^2 \cdot H_c^2}{x''^2 \cdot H_b^2 \cdot H_c^2 + y''^2 \cdot H_a^2 \cdot H_c^2 + z''^2 \cdot H_a^2 \cdot H_b^2}} \tag{7.26}$$

If $t_s > 1$, then the point lies within the ellipsoid. With this f_i can be computed using transformed coordinates:

$$f_i(x'', y'', z'') = \begin{cases} b_i & , t_s > 1 \\ b_i \cdot e^{-a_i \cdot (1 - t_s)^2 \cdot (x''^2 + y''^2 + z''^2)} & , t_s \leq 1 \end{cases} \tag{7.27}$$

Parameter Definition for Ellipsoidal Primitives. The *ellipsoidal primitives* contain the two parameters a_i and b_i, which control the descent and magnitude of the corresponding field function. These two parameters should be determined automatically, because a configuration by the user may be tricky and instable. Whenever possible, the algorithm should disburden the user of such decisions.

The simplest approach would be a static setting for these two parameters. Unfortunately, this idea is not acceptable because the visualized clusters vary too much in both scale and position. Thus, it is impossible to find fixed values delivering satisfactory results under all circumstances. The parameters have to be set in context with the underlying ellipsoid. We will discuss two possible approaches solving this problem:

1. The heavier a cluster is, i.e. the more objects it contains, the larger becomes the value of the magnitude b_i of the ellipsoid primitive's field function.
2. The larger the maximum extension of the ellipsoid is, the weaker becomes the descent a_i of the ellipsoid primitive's field function.

Experiments have shown, rule one can lead to very big BLOB surfaces, e.g. if the object distribution in space is dense. Hence, this rule was dropped and a fixed value is assigned to b_i (e.g. b_i=1.0).

The second rule on the other hand has turned out to provide a relevant visual feedback. The parameter a_i is defined as

$$a_i = \frac{a_0}{ellipsoid's\ dimensions} \tag{7.28}$$

where the value for the constant factor a_0 must be determined experimentally, yet.

Determination of Isovalues to ensure connected BLOB-Surfaces. According to [GSF97] a BLOB's shape is strongly influenced by the corresponding isovalue $c > 0$. The smaller this value, the larger the BLOB's extension will get. In order to ensure that a BLOB encloses all its objects the correct choice of c is crucial. In this section, heuristics for the automatic determination of isovalues are presented.

Take the example of Figure 7.15a where an enclosing BLOB surface for three objects A, B and C has to be computed. The indicated number on the connecting edges illustrates the minimal value of the superimposed field along the edge. In order to assure as tight a BLOB as possible we have to look for the largest isovalue which still guarantees that the BLOB does not break apart.

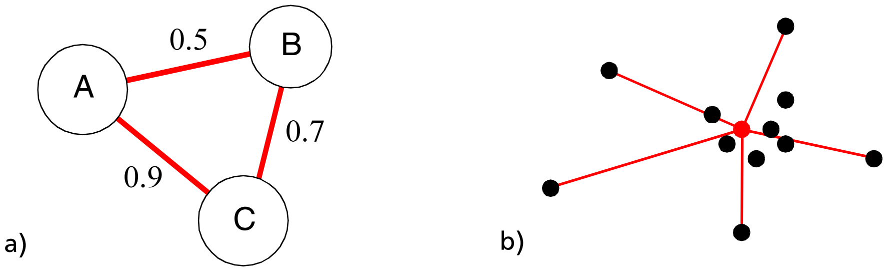


Figure 7.15 a) Three objects for which an enclosing tight BLOB surface has to be found. b) Objects of a cluster with so-called outlier objects. The interconnecting lines between outliers and the cluster center are marked in red.

Figure 7.16 shows three possible cases for the choice of an isovalue. On the left hand side, the chosen value results in the illustrated split-up into two subclusters because $c = 0.8$ is bigger than the minimal field value on edges AB and BC. On the right hand side, too small an isovalue does not provide for a distinctive shape. The case illustrated in the middle seems ideal. Choosing $c = 0.6$ – bigger than the minimum on edge AB but smaller than the minimal value on BC – results in a tight single BLOB surface enclosing all objects.

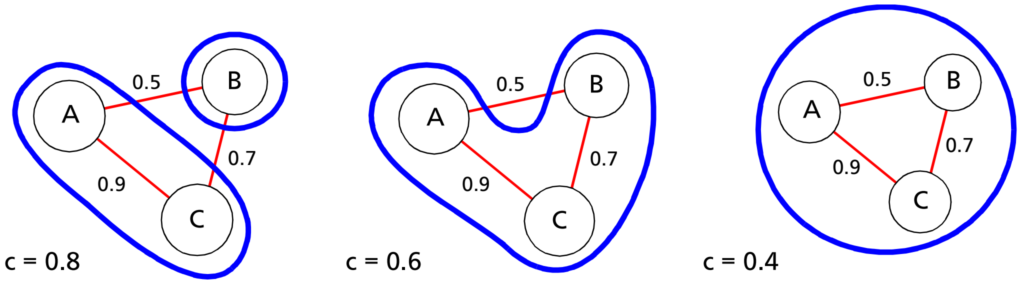


Figure 7.16 left: isovalue too big, BLOB breaks apart, middle: optimal isovalue, tight BLOB enclosing all objects, right: isovalue too small, non-distinctive shape

This example shows how to find an ideal isovalue: look for the biggest value that still guarantees for a single enclosing surface. This is equivalent to choosing a value such that all objects are connected by edges with minimal field value bigger or equal to the isovalue.

There are two problems in this approach: first, graph theory shows that it is very expensive to find a minimal spanning tree [BM76], at least if cluster sizes approach several hundred objects. Second, finding the minimal field value on interconnecting lines is expensive too, as it is impossible to find an analytic solution for arbitrarily superimposed fields. In the remainder of the section, we present an approach which in most cases yields suitable isovalues.

Figure 7.15b shows a constellation of several objects of a cluster for which an enclosing BLOB surface has to be found. The red dot marks the center of the cluster. Intuitively, objects close to cluster center will not cause problems. In contrast, it is troublesome to account for outliers – objects which are far apart from the cluster's center. Instead of looking for a minimal spanning tree for all of the cluster's objects we concentrate on the outliers. Therefore, we look for the minimal field value on the interconnecting lines between the outlier and the cluster center. Figure 7.15b shows these lines highlighted in red. The smallest value found is regarded as a good approximation to the ideal isovalue.

We are left with the problem of finding the minimal field value on the lines between outliers and the cluster center. To this aim, we employ a Newton iteration scheme in order to find the zero crossings of the first derivative of the superimposed field function with regard to the parametrization t of the interconnecting line

$$f'(t) = 0. \tag{7.29}$$

The corresponding Newton iteration step is given by

$$t_{n+1} = t_n - \frac{f'(t)}{f''(t)}. \tag{7.30}$$

As it is hardly possible to find symbolic expressions for the first and second derivative of the field function f, they are approximated in terms of central differences as follows:

$$\begin{aligned} f'(t) &\approx \frac{f(t+\Delta t) - f(t-\Delta t)}{2} \\ f''(t) &\approx \frac{f'(t+\Delta t) - f'(t-\Delta t)}{2} \\ &= \frac{2f(t+2\Delta t) - 2f(t) + 2f(t-2\Delta t)}{4} \end{aligned} \tag{7.31}$$

As the reader may have noticed, this procedure is not guaranteed to find the global minimum but is highly dependent on the choice of a favorable initial value t_0. In order to find a good value for t_0, we sample the value of the field function on equidistant points on the interconnecting line and choose t_0 to be the smallest value found during the sampling procedure. As a matter of fact, the outlined procedure still does not provide for finding the global minimum. However, practice has shown, that it yields suitable isovalues for non-pathological cases. For clusters of less than five objects the minimal spanning tree is computed which guarantees for the optimal isovalue.

The following small example illustrates the basic properties of the H-BLOB clustering algorithm. The scene consists of 5 single objects each represented by a colored sphere. We present two snap-shots of the cluster tree buildup sequence including the corresponding implicit cluster surfaces generated by the H-BLOB algorithm.

7.4 IMPLEMENTATION ISSUES

All algorithms have been fully implemented as part of a class library in JAVA. For the domain of 3D visualization we apply Java3D in the version 1.1.2. All computational work is done on a standard PC completed with a hardware accelerated graphics subsystem (Open GL). Even for more complex examples we still get interactive frame rates.

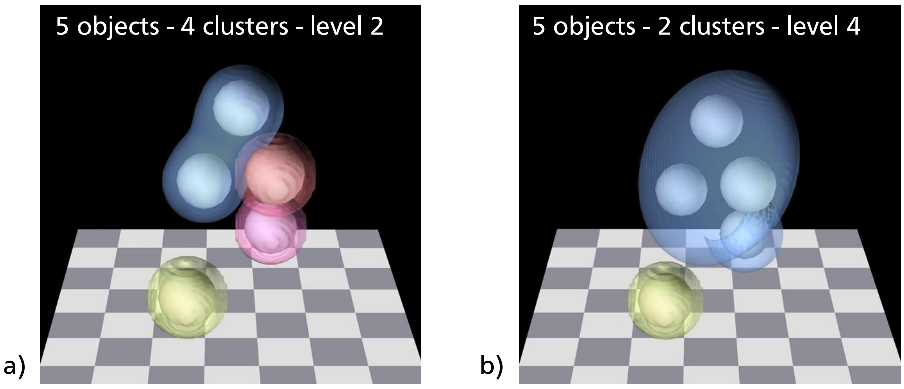

5 objects - 4 clusters - level 2
a)
5 objects - 2 clusters - level 4
b)

Figure 7.17 Small example showing the clustering process by means of 5 simple objects. Snapshots with 4 respectively 2 clusters are shown. Level indicates the hierarchy level in respect with the cluster tree.

The first issue concerns the isosurface extraction. In spite of the multi-resolution approach it remains the most time consuming part of the algorithm. Implicit surfaces may provide very nice shapes, but are computationally very expensive. There are many sources available for this topic, but for our prototype implementation our choice was [Blo94].

Regarding the implementation of the H-BLOB algorithm there is an additional issue worth to be mentioned. It concerns the data structure used for the edge collapse clustering. Since this stage of the algorithm makes heavy use of point-to-point distance calculations and cluster merging, together with the higher order characteristic of the problem, makes a good choice difficult. Employing standard data structures quickly leads to a performance bottleneck, mostly because of memory shortage. Some promising work addressing this type of problem can be found in [Epp95].

8

INFORMATION VISUALIZATION MODELING LANGUAGE (IVML)

Having a flexible and powerful framework is just one part of the game – in order to make the most of its abilities an equivalently potent configuration tool is needed. For the concrete case of the IVORY framework the *Information Visualization Modeling Language* (IVML) will cover this requirement. It accomplishes the configuration and parametrization of both the framework kernel as well as the various plug-ins to be loaded. IVML denotes an elementary component of the *framework configuration model* presented in Section 6.5 on page 91. Implemented as an open and object-oriented high-level script language, IVML is distinctively proposed to generically describe the topology of information visualization problems. The reader may find the detailed EBNF definition of IVML in Appendix B. In addition, Appendix B shows an overview of the IVORY plug-in classes and their inheritance dependencies.

8.1 SCOPE

The *Information Visualization Modeling Language* (IVML) is a language for modelling relation-based information topologies. This comprises all aspects of the input data as well as a description of the relations implicitly contained in the data. Hence, IVML mainly deals with *data objects* representing the actual information as well as *connection objects* expressing the relation between two data objects. In the case of IVORY these objects are implemented by plug-in objects (See “Plug-in Mechanism” on page 88.).

Since information visualization problems fall into a wide range of different problem classes, a high-level configuration language is a prerequisite for an efficient use of any framework in this area. In practice, problems appearing seemingly different may be visualized often with an identical set of data and connection objects just by re-parameterizing these. IVML is a high-level configuration language considering these aspects. As such, it is implemented as an interpreted script language that yields short turn-around times. Hence, IVML gives the advanced framework user (see Section 6.5 on page 91) a powerful tool to rapidly configure the behavior and parameterize the visual metaphors of the individual plug-ins, namely the data and connection objects.

In summary, the IVML script language serves the following four purposes:

1. Declaration of involved data sources
2. Definition of appropriate plug-in sets
3. Parameterization of chosen components
4. Description of the visualization problem's topology

Although, IVML is perfectly adapted to the concepts of IVORY, it does not have to be used in combination with the IVORY framework.

8.2 REQUIREMENTS

The IVML script language is designed to meet with the following requirements:

- Object-oriented and open
  Since plug-ins are built up hierarchically, the language should support a concept of inheritance. Based on the object-oriented approach IVML is supposed to provide the ability to add new object types that enable even the configuration of yet unknown plug-ins.

- Context-free
  For the sake of simplicity of the corresponding parser, the language should be context-free. As a consequence, a one-pass, one-token-look-ahead parser implementation is sufficient.

- Prototype support
  Language must support a kind of prototype-mechanism that enables the script programmer to declare templates of object groups in order to generate instances of them later on.

- Composability
  Provide the ability to use and combine externally defined objects within an IVML scene. A typical IVML scene will be composed of a set of subscripts: one to define the environment, another to define object appearances and a third to define problem specific configured object templates. With this mechanism modular script design and re-usability become feasible. Hierarchical inclusion enables the creation of scenes of arbitrary length.

- Support for active objects
  It must be possible to define objects capable of generating new objects. As a result, data objects may actively gather information autonomously. In addition, connection objects may be generated automatically by an active connector object joining data objects by means of a connection rule.

- Expressions and references
  In order to offer a flexible script language, expressions and references must be supported. Hence, calculations and mutual dependences of object parameters may be expressed in the script flow.
- Scene-global time line
  An integral part of each IVML scene is an attached time line defining the timewise start and end point of the underlying data as well as its temporal resolution. This enables the creation of interactive time line animations of dynamic data sources.

8.3 DESIGN CONCEPTS

Considering the requirements listed in the preceding section, we notice that a language well known in the field of computer graphics satisfies most of the conditions: The *Virtual Reality Modeling Language* (VRML)[1] in the current version 2.0. It is a format for describing interactive 3D objects and worlds. VRML is used in a variety of application areas such as engineering and scientific visualization, multimedia presentation, entertainment and educational titles, web pages, and shared virtual worlds. It was designed to be a universal interchange format for integrated 3D graphics on the World Wide Web [VRML97].

In our setting we will adopt most of the characteristics of VRML. One of the most important characteristics is related to the VRML object definition model. Similar to VRML the basic building blocks describing an IVML scene are objects defined by fields – also called *field containers*. Fundamentally, the number of fields and their names are determined by the set of configurable object parameters. The following example outlines a typical IVML object:

```
SimpleData {
  visual IVVisual {
    color <1.0, 0.0, 0.0>
    scale <4.0, 4.0, 4.0>
  }
  label "Demo Object 1"

  appearanceURL "./shapes/apple.wrz"
  datapath "../nov01/apple.data"
}
```

In the case of IVML each object is implemented by a corresponding IVORY plug-in (See “Plug-in Mechanism” on page 88.). The field container’s type thereby determines the key for the plug-in mapping. For the case of the preceding example, the objects `SimpleData and IVVisual` are implemented by corresponding IVORY plug-ins having the same name. The fields `visual`, `label`, `appearanceURL` and `datapath` of the `SimpleData` object are mapped to the corresponding plug-in parameters. The same applies to the type `IVVisual` with its fields `color` and `scale`.

8.3.1 Deviations from the VRML standard

Unfortunately, some of the requirements for IVML demand adaptations of the VRML language. Most of the changes affect the language syntax only.

---

[1] According to the name of the International Standard ISO/IEC 14772-1:1997, VRML 2.0 is also referred to as VRML97.

Expressions. The strongest deviation to VRML is the integration of expressions. Each IVML field can consist not only of a numerical or literal value, but also of arbitrary expressions. Expanding the language syntax in such a way, implies however further changes. In particular, problems concerning the parsing of multi-value fields (e.g. vectors) crop up. Therefore, in IVML the comma is demanded as a separator in multi-value fields.

Additionally, each single value of a multi-value field must be unambiguously parsable. This claims an extended notation for vectors using '<' to mark the start and '>' to mark the end of a vector.

References. In VRML references are introduced with the keyword `USE` followed by an object path leading to the field which is supposed to be referenced. Since in IVML references may be directly integrated in expressions, the keyword `USE` is not required. To keep the implementation of the reference handling simple, certain rules apply:

1. References must always contain a complete and absolute path, from the root up to the target field or object. Therefore variable names become indeed longer, but the resolving of such names gets considerably simpler (e.g. `system.cluster.transparency`).
2. To ensure that relative references may be constructed using the special pathnames me and parent. `Me` refers to the object in which the expression is located. Correspondingly, `parent` addresses the object one level above in the object hierarchy from the object denoted by `me` (e.g. `me.parent.label`).

Prototypes. As in VRML prototypes in IVML begin with the keyword `PROTO` followed by the name of the prototype. The prototype is in turn followed by an object definition in braces that applies to that prototype. In contrast to VRML, this definition may consist only of a single object on the uppermost level.

The following example defines a prototype with name `p1` and type `SimpleData`. In addition the prototype field `value` is preset to the expression `(2*me.a)/sin(0.334)`, where `me.a` can remain undefined yet:

```
PROTO p1 {
  SimpleData {
    ...
    value (2*me.a)/sin(0.334)
    ...
  }
}
```

The VRML keyword `EXTERNPROTO` that specifies an externally implemented prototype has been dropped in IVML. Since IVML supports a more generalized object approach, where each IVML object is implemented by a corresponding IVORY plug-in, the `EXTERNPROTO` mechanism is inherently contained in each IVML definition statement.

8.4 SPECIFICATION

8.4.1 Language Basics

As illustrated in Figure 8.1 below, an IVML script consists of several sections. Except for the *Header*, a section always consists of a sequence of (hierarchical) object definitions. Inside an object definition a set of fields can be defined. In general, the order of the sections does not matter except for the *Header* which has to be at the beginning and prototypes must be defined before they are applied.

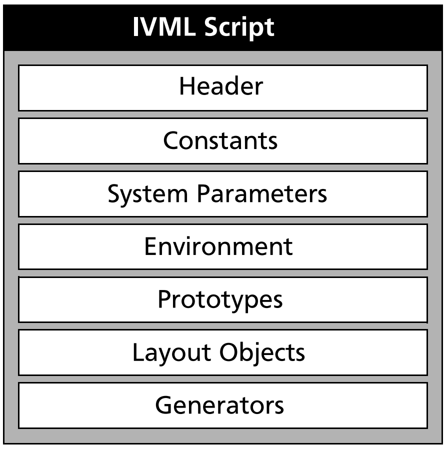


Figure 8.1 Schematic structure of an IVML script

Additionally, the language permits using the keyword `INLINE`, which enables the textual import of external resources. These may for example contain prototype definitions of general importance complementing the prototype set defined in the original IVML script. A further application concerns the import of object appearances usually stored in separate VRML files.

Below you will find a description of individual script sections in accordance with the structure shown in Figure 8.1.

Header. *The header* simply identifies the IVML script and version.

```
#IVML 1.0
```

Constant Section. Constants are declared as fields of type `SVString`. They are kept in objects of class `IvoryConst`. Since constants may be arbitrarily named the `IvoryConst` object is capable to manage arbitrary field names.

Type description:

```
IvoryConst {
  SVString any_fieldname_1
  SVString any_fieldname_2
  ...
  SVString any_fieldname_n
}
```

Example:

```
DEF myconst IvoryConst {
  myTitle "Hello World"
  myURL "http://hello.world.com/basics.html"
}
```

System Parameter Section. System parameters are stored in an `IvorySystem` object, which is automatically initialized at parse-time of the script. If no system object was defined anywhere in the script a default instance will be created. By convention, the `IvorySystem` object always has the name `system`. Thus an explicit naming can be omitted.

Type description:

```
IvorySystem {
  SVString codebase
  SVString scriptbase
  IVInfo info
  BaseLayout layout
  BaseCluster cluster
}
```

Example:

```
DEF system IvorySystem {
  info IVInfo {
    title "Economic indices"
    info  ["Example scene for Ivory V2.0"]
  }
  layout RKLayout {
    series IVTimeSeries {
      start 1.1.75
      stop  31.12.94
    }
  }
  cluster BlobCluster {
    visible TRUE
    transparency 0.8
  }
}
```

The various system fields are now explained briefly. The `codebase` contains the path to the plug-in classes. This field is set to be read-only. The `scriptbase` similarly points to the directory from where the IVML script was loaded. The `info` field holds general remarks and annotations about the current scene. This may be a scene title or a short description of the input data. Furthermore the layout algorithm to be applied and its specific parameters are determined by the field `layout`. In the same way the clustering method can be configured freely by setting the `cluster` field.

Environment Section. The environment object `IvoryEnvironment` contains VRML inlays to define individual backgrounds, light sources, cursors and static scene parameters. It implicitly called env. As with the `IvorySystem` object an instance is generated implicitly unless it is done explicitly elsewhere in the script. Since the various field names are rather self-explanatory, a more detailed explanation is omitted here for brevity.

Type description:

```
IvoryEnvironment {
  SVNode lights
  SVNode cursor
  SVNode billboards
  SVNode environment
  SVNode viewpoints
  SVNode script
  SVNode userdef
  MVRoute routes
}
```

Example:

```
DEF env IvoryEnvironment {
  lights       INLINE "brightLight.wrl"
  cursor       INLINE "crossCursor.wrl"
```

```
    environment  INLINE "financeEnv.wrl"
    billboards   INLINE "extendetBboard.wrl"
  }
```

Prototype Section. In order to construct a number of objects with common field values, a so-called *prototype* can be employed. Instead of defining all field values for every object instance individually, these can be set once in the prototype definition. The object instances are then created by cloning the prototype. As a consequence, all field values are inherited by the object. Individual entity characteristics can still be changed, however, at the time of the actual object creation. A prototype has the same type as the object it contains.

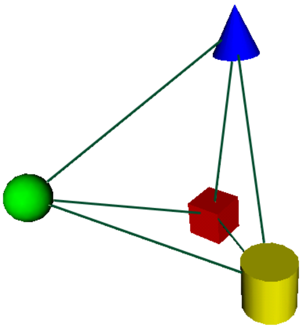


Figure 8.2 This small scene consisting of 4 objects arranged in a tetrahedron shows the use of prototypes. Each object is defined using a different prototype (`SphereData`, `CubeData`, `etc.`) which gives the object its particular shape. Nevertheless all objects are of the same type `SimpleData`.

On one hand, this approach increases the legibility of a script, and so decreases possible occurrence of typos and other errors. On the other hand, it reduces the script length and with that the parser's expenditure. Besides the performance improvement, prototypes increase the modularity and reusability of IVML scripts.

Syntax:

```
PROTO identifier {
  classname {
    # field settings
  }
}
```

Example:

First, two prototypes (`SphereData`, `CubeData`) are defined. Both of them base on the same object class (`SimpleData`), but have, however, different appearance fields.

```
PROTO SphereData {
  SimpleData {
    appearance "Sphere {}"
  }
}

PROTO CubeData {
  SimpleData {
    appearance "Box {}"
    }
  }
```

In the following an object `data1` gets instantiated by means of the prototype `SphereData`. Consequently, the new object adopts the appearance from the prototype and `data1` will be displayed as a sphere.

```
  DEF data1 SphereData {
    ...
  }
```

Again an object – this time `data2` – of class `SimpleData` gets created. However, in this case the appearance field of the prototype `CubeData` is redefined. The object `data2` will be represented by a cone.

```
  DEF data2 CubeData {
    appearance "Cone {}"
  }
```

Layout Object Section. The most important objects which are specified in a script are located in this section – the *layout objects*. With them the actual data and its implicit relation topology are brought into the system. One can distinguish between two types: First the *data object* that serves as a container for the data to be visualized. Second the *connection object* that abstracts the relation between two *data objects* (see Section 8.4.4).

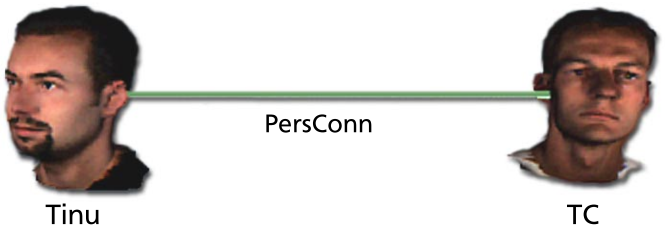


Figure 8.3 Illustration of the scene resulting from the script above

All *layout objects* find their implementation in the form of an IVORY plug-in, derived from either the base class `BaseData` or `BaseConn`, as the case may be. After the parsing has finished all objects are filled in into the backend data structure of IVORY. They form the foundation for all further operations (layout computation, clustering, connectivity analysis, etc.) in the system. In contrast to all other object types, layout objects may have their corresponding representation in the visualization.

Example:

```
  # Prototype for our data objects
  PROTO CGGPersData {
    PersonalData {
      visual IVVisual {
        scale <4.0, 4.0, 4.0>
      }
      datapath "sql:pers:cgg:" + me.name
      infoURL "http://info.server.org/pers/" + me.name
      appearanceURL system.scriptbase + "./faces/" + me.name
    }
  }

  DEF Tinu CGGPersData {
    label "Tinu Roth"
  }
  DEF TC CGGPersData {
    label "Tom Sprenger"
  }
  DEF PersConn PersonalConn {
    leftObj Tinu
    rightObj TC
  }
```

The shown example involves two data objects – named `Tinu` and `TC` – both of type `PersonalData`. Their definition and instantiation occurs indirectly by means of the prototype `CGGPersData`. It presets the field values for the data source, a URL probably containing additional information and a link to the object's appearance description. Within the actual object definition it suffices to put the correct label. Finally, a connection object `PersConn` is defined, which brings the `Tinu` and the `TC` data objects in relation to each other.

Generator Section. Up to now, all object were passive in the sense of dynamic object creation. Each single object has to be defined explicitly in the script. This is likely to be sufficient for lab examples and small applications. However, thinking of real world examples consisting of several hundreds or even thousands of data and connection objects one realizes easily that more powerful mechanisms are needed for such complex environments.

A rule-based generation mechanism is needed. That is what *generator objects* are for (see Section 8.4.4). Configured with a *generation rule* and a *template* of the objects to be generated they offer automatic object creation. Immediately after the parsing has completed the rules of all generator objects are applied one after the other to the scene.

This approach is in particular conceivable for connection objects, but also for data objects. Consequently, we may identify two main types of generator objects: *Data source handlers* manage the information de-multiplexing of a data source. As a main task they pass on the partitioned information to existing data objects or create a new object if required. *Connectors* on the other hand create new connections between data objects on request. Hence, they support the description of the topology of the visualization problem.

8.4.2 Coordinate System

Analogous to VRML IVML uses a Cartesian, right-handed, 3-dimensional coordinate system [VRML97]. By default, objects are projected onto a 2-dimensional screen by projecting them in the direction of the positive Z-axis, with the positive X-axis to the right and the positive Y-axis up. Moving the camera – using the keyboard or mouse – alters this default projection.

8.4.3 Fields

Fields represent the properties of the configurable objects. In general, we may distinguish between two different field types. A field is either of type *plain field* or a *substructure*.

Plain Fields. The plain fields actually hold the values of object properties. They thus comprise the basic configuration data containers. Plain fields may be again divided into two subcategories. If the field content consists of a single value only, it is a *singlevalue field.* Otherwise it is a question of a *multivalue field.*

Table 8.1 lists the plain field types currently supported by IVML[1]. All fields contain an internal value object storing the actual value respectively values of the field and optionally

---

1. Similar to the plug-in model for objects, also new IVML fields types may be added to the system.

an expression tree. The value object bases on either a built-in Java object (e.g. `java.lang.String`) or a customized container object (e.g. `ivory.ivml.Vec2f`). In the sense of an object-oriented approach we may assign a value of the same type or any derivation of that to a field.

Substructures. Substructures are sub-objects, which basically correspond to conventional object definitions but take place inside of another object. A sub-object is stored as a reference to an *IvoryFieldContainer*.

Table 8.1 List of valid IVML plain fields including their corresponding internal value types

Singlevalue Field	Internal Value type
SVBool	java.lang.Boolean
SVInt32	java.lang.Integer
SVFloat	java.lang.Float
SVDouble	java.lang.Double
SVString	java.lang.String
SVVec2f	ivory.ivml.Vec2f
SVVec3f	ivory.ivml.Vec3f
SVRotation	ivory.ivml.Rotation
SVColor	ivory.ivml.Vec3f
SVTime	ivory.ivml.Time
SVNode	java.lang.String

Multivalue Fields	Internal Value type
MVInt32	java.lang.Integer[]
MVFloat	java.lang.Float[]
MVString	java.lang.String[]
MVRoute	java.lang.String[]

8.4.4 Field Containers

A *field container* typically consists of one or more fields. A field can be either a *plain field* or a *substructure* (see Section 8.4.3). In the case of a plain field the container itself holds the assigned value. Otherwise a reference to another field container is stored in the container structure as it is shown in Figure 8.4.

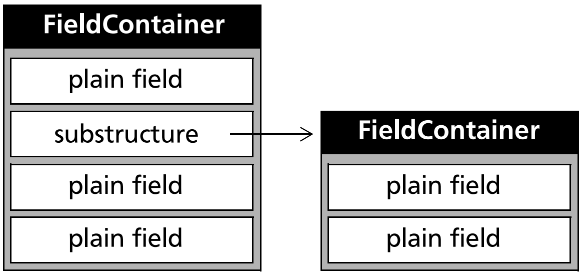


Figure 8.4 Example showing the basic structure of a field container.

The *field container* defines the basic functional component of the IVML script language. From the IVML point of view, *field containers* are pure structural objects that group together a number of fields. Apriori, they do not provide any further functionality. This looks different from the viewpoint of the framework. Every container has its equally named counterpart on the framework side in the form of an IVORY plug-in (see Section 6.4.2 on page 88). By definition the plug-in must – analogously to the JavaBeans specification [JavaBeans96]- provide a getter- and a setter-method for each field of the field container. While parsing the script each field container triggers the dynamic loading of its associated plug-in class. With every loading the framework is expanded by additional functionality. Thus, the sum of loaded containers finally defines the functional scope of the configured framework.

Definition:

```
DEF identifier PluginClassName {
    ...
    field definitions
    ...
}
```

As a consequence of the interdependence described above it is not only the framework gaining functionality with every new plug-in, but also IVML's vocabulary that grows. With every new plug-in automatically a new field container type becomes known in the script language. Field containers may be divided up into three main groups: s*ystem objects*, *layout objects* and *generators*. A similar subdivision into different Java packages can be found on the part of the plug-ins. In following, the different groups are explained in more detail.

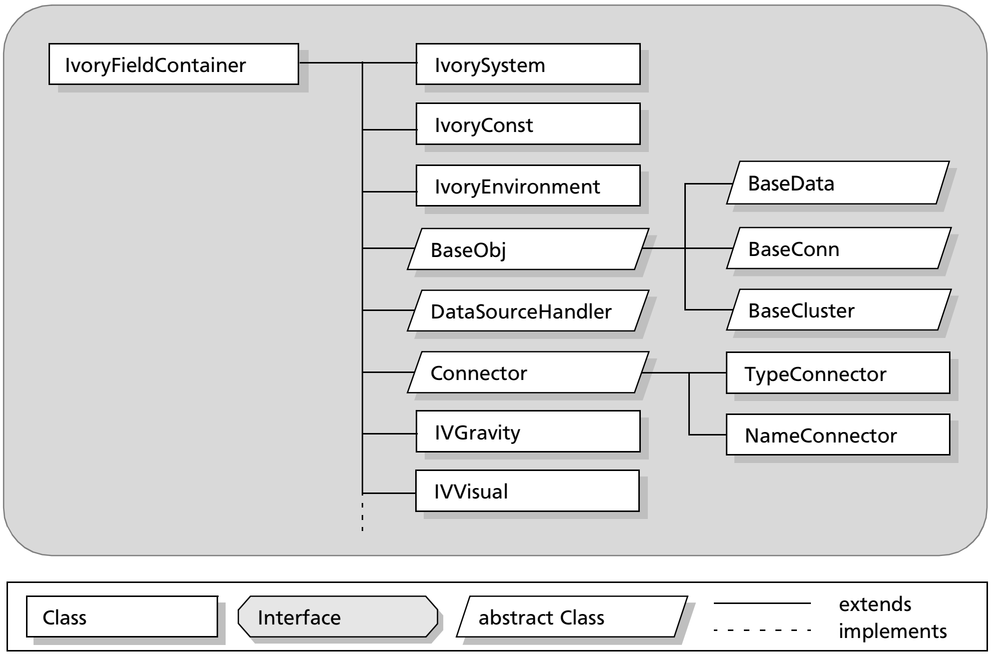


Figure 8.5 Plug-in hierarchy of generic IVML specific classes

System Objects. Global parameters, layout settings and the VRML environment are held in *system objects* (see also Section 8.4.1). All System objects get implemented by special plug-in classes located in the IVML system package.

The two main types of system objects are `IvorySystem` and `IvoryEnvironment`. If they are not defined in the script, the interpreter generates them implicitly. Since they may exist only once in the entire system, their names are generated automatically.

- `IvorySystem` is designated always as `system`

- `IvoryEnvironment` is designated always as `env`

A peculiar characteristic has the third system object, the `IvoryConst` container. It deviates from the restriction that for every field in the field container a method must be defined in the plug-in. The field handling is generalized to the effect that as many field as desired with arbitrary names can be managed. This extension allows a convenient administration of named constants.

Layout Objects. Basically, IVML recognizes two types of layout objects, namely *data objects* and *connection objects*. The former serve as placeholders for the actual data, while the latter define the connectivity between data objects. The joined plug-in classes are named `BaseData` and `BaseConn`. They both reside in the IVORY public plug-in package which also accommodates any of their user-written derivatives. As can be inferred from Figure 8.5, both plug-ins are derived from the same base class `BaseObj`. All types of layout objects may be defined either explicitly in the script or instantiated at run-time by generators. In addition, layout objects are the only entities that may have a visual representation shown in the visualization.

Base Object (`BaseObj`). This object establishes the basis for all layout objects. It contains the elementary properties that are common to all layout objects.

Definition:

```
BaseObj {
  SVString name          # "BaseObj"
  SVString label         # "DefaultBaseObj"
  IVVisual visual        # {see below}
  SVBool fixed           # FALSE
  SVBool gravity         # FALSE
  SVBool lookAtMe        # FALSE
  SVNode appearance      # "Sphere {}"
  SVString appearanceURL# {empty}
}

IVVisual {
  SVVec3f position       # <0,0,0>
  SVRotation rotation    # <0,0,1,0>
  SVVec3f scale          # <1,1,1>
  SVColor color          # <1,1,1>
}
```

`Explanation:`

Type	Name	Description
SVString	`name`	Indicates the name of the object. At declaration time the DEF identifier — if present — is assigned. The object's name is used for referencing purposes. This field is Read-Only.
SVString	`label`	Determines the visual text tag of an object. This can not be employed to reference the object. It has rather informative character.
IVVisual	`visual`	Defines the visual parameters of an object, such as scaling, rotation, etc. However, it does not include the object's shape description. Usually most of the fields are computed by metrics implemented in the plug-in.
SVBool	`fixed`	Determine whether the object's position is supposed to be fixed in the coordinate system.

Type	Name	Description
SVBool	`gravity`	Switches on a virtual gravity force, that causes the object to stay aligned in the direction of the X-axis independently of any other applied transformation.
SVBool	`lookAtMe`	Determine whether the object is supposed to align itself always in the direction of the camera (billboard effect).
SVNode	`appearance`	Defines the appearance of the object in syntax VRML.
SVString	`appearanceURL`	Names an URL to a VRML file, that contains an object appearance description. If both appearance fields are set, the field `appearance` is superseded.

Base Data Object (`BaseData`). The *data object* abstracts the actual data to be visualized. It is derived directly from the basic object (`BaseObj`). Hence, the data object automatically inherits all fields already defined in the basic object. Furthermore, the data object complements the container by fields required for the parametrization of the layout process. User-defined data objects must be derived from `BaseData` in order to work properly in the framework.

Definition:

```
BaseData {
  # These fields are inherited from BaseObj
  SVString name        # "BaseData"
  SVString label       # "DefaultBaseData"
  IVVisual visual
  SVBool fixed
  SVBool gravity
  SVBool lookAtMe
  SVNode appearance

  # These fields are members of BaseData
  SVFloat mass
  SVFloat spaceFactor
  SVFloat lamda
  MVString neighbors
}
```

Explanation:

Type	Name	Description
SVFloat	`mass`	Contains the mass value for this object. This will have an impact on the object position computed by a physics-based layout algorithm.
SVFloat	`spaceFactor`	Contains a spaceFactor value for this object. This will affect the object position computed by the physics-based layout algorithm.
SVFloat	`lamda`	Contains the charge value for this object. This will have an impact on the object position computed by a physics-based layout algorithm.
MVString	`neighbors`	Holds the names (`BaseObj.name`) of all objects connected to this. Of particular interest may be the length of this multi-value field which indicates the number of neighbors. This field is Read-Only.

Base Connection Object (`BaseConn`). The *connection objects* are used for the linkage of two data objects. Similar to data objects, connection objects inherit all fields from the base object and supplements those with additional fields required mainly for layouting. Extensions of connection objects must be derived from the class `BaseConn` in order to work properly in the framework.

Definition:

```
BaseConn {
  # These fields are inherited from BaseObj
  SVString name       # "BaseConn"
  SVString label      # "DefaultBaseConn"
  IVVisual visual
  SVBool fixed
  SVBool gravity
  SVBool lookAtMe
  SVNode appearance

  # These fields are members of BaseConn
  BaseData leftObj    # null
  BaseData rightObj   # null
  SVFloat correlation
  SVFloat strength
  SVFloat defaultLength
}
```

Explanation:

Type	Name	Description

BaseData	`leftObj,`
	`rightObj`
		These substructures refer to data objects, that are supposed to be linked by this connection object.

SVFloat	`correlation`	Indicates the similarity of the two connected data objects. This value is usually computed by the plug-in metric function.
SVFloat	`strength`	Contains a strength value for this connection. This will have impact to the positions of the connected data objects. The larger it is the stronger becomes the connection.
SVFloat	`defaultLength`	Contains a value for the default or rest length of this connection. This will have impact to the positions of the connected data objects. The larger it is the more far away from each other are the data objects positioned.

Generator Objects. The specific quality of *generator objects* is that unlike all other objects they have an active component. Generator objects can create new layout objects autonomously by following certain *rules* defined specifically to this purpose. They are activated after the complete parsing of the IVML script. After that, the generators can keep on working in the background, generating new objects or deleting existing ones as required.

Abstract Definition:

```
Generator {
  BaseObj template
}
```

Explanation:

Type	Name	Description
BaseObj	`template`	Defines the object, that will be cloned whenever the generator wants to create a new object instance.

Connector. In order to automatically generate connection objects, so-called *connectors* can be employed. The basic connector is defined in an abstract manner and is thus not applicable in a script. Of more practical interest are the two extensions `NameConnector` and `TypeConnector`. They are both derived from the base connector class `IvoryConnector`. The `NameConnector` includes an additional field that will contain the name-based linking rule. Similarly, the `TypeConnector` includes a field containing the type-based generation rule.

Definition:

```
NameConnector {
  BaseObj template  # inherited from Generator
  MFString names
  }

TypeConnector {
  BaseObj template  # inherited from Generator
  MFString types
  }
```

Explanation:

Type	Name	Description

MFString	`names`	By means of a pair of name patterns this field indicates, between which data objects the connector is supposed to construct connection objects. In the case of the `NameConnector` the connection rule is defined as follows:
		`names [ident1 ident2]`
		whereas `ident1` and `ident2` are the two name patterns of data objects to be linked. Thereby, all data objects whose name satisfies `ident1` are connected with all those fitting `ident2`. The names may contain wildcards (‘*’ and ‘?’).
		Example 1: The subsequent rule connects the data object with the name *data1* with all those having a name that starts with *B_Data*:
		`names ["data1", "B_Data*"]`
		Example 2: In order to generate a complete graph, that is to link every data object with every other, the following rule can be applied:
		`names ["*", "*"]`

Type	Name	Description

MFString	types	By means of a pair of type patterns this field indicates, between which data objects the connector is supposed to construct connection objects. In the case of the `NameConnector` the connection rule is defined as follows:
		`types [type1 type2]`
		whereas `type1` and `type2` are the two type patterns of data objects to be connected. Thereby, all data objects whose type satisfies `type1` are connected with all those satisfying `type2`. The types may contain wildcards (‘*’ and ‘?’).
		For the following examples we assume data objects of the classes *AppleData* and *PearData* were defined in the script. In order to interconnect all objects of class *AppleData* one writes:
		`types ["AppleData", "AppleData"]`
		To connect all objects of the two classes in pairs:
		`types ["AppleData", "PearData"]`

Datasource Handler. In principle, there are two ways of getting data into data objects. On the one hand, the data import may be governed by the data objects themselves. They are passed the required source parameters (filenames or URLs) and will load the data on initialization. Alternatively, data imports may be managed by a global *data source handler* that automatically generates the required objects in accordance with the kind of data that is found.

Definition:

```
DataSourceHandler {
  BaseObj template  # inherited from Generator

  # These fields are members of DataSourceHanlder
  NameConnector nameConnector
  TypeConnector nameConnector
  MVString sources
  }
```

Explanation:

Type	Name	Description
NameConnector	nameConnector	Handles the name-based connecting of newly created data objects. This connector can be configured as described in the previous subsection.
TypeConnector	typeConnector	Handles alternatively the type-based connecting of newly created data objects. This connector can be configured as described in the previous subsection.
MVString	sources	List of source URLs where the handler reads the data from.

8.5 IMPLEMENTATION ISSUES

8.5.1 Interpreted script language

The fundamental question of whether to use compiled or interpreted components is solved by a compromise. On one hand the interpreted approach is used wherever the execution repetition is very low and performance is therefore of secondary importance. On the other hand compiled code is used for all low-level and often-run-through program segments.

In contrast to all other system components which are built using a compiled language providing a maximum of speed, the IVML configuration scripts are handled by an interpreter. Since they execute only once at start-up time we balance time against flexibility. However, some speed optimization mechanisms are required – especially in the area of expression evaluations. The most important aspects are described in the two subsequent sections.

8.5.2 Extensibility through Java's Reflection API

The IVORY implementation takes full advantage of all sophisticated features provided by Java. Each plug-in is mapped onto a Java class. Thus, the loading of individual plug-ins at runtime can be accomplished by the dynamic class loading mechanism of Java. The Java reflection model is used by the IVML interpreter to check, read and set the field values of plug-in objects.

We use the naming convention for setter and getter methods of the JavaBeans API [JavaBeans96] to access the underlying Java member variables. The JavaBeans compliance enables the use of all advanced JavaBeans features, such as property editing.

8.5.3 Dirty-Tag Propagation

In order to accelerate expression evaluation the computed value will be cached. As long as the dirty tag is not set in an expression, it will deliver the cached value on request. But how can an expression get dirty?

Expressions are organized as trees of *value, variable, operator* and *function nodes*. If the value of a variable node changes, e.g. in cases were a user changes a property of an object, all nodes 'above' the respective node become invalid. In other words, every node undergoing a property change has to propagate the change to his parent as well as, possibly, to all variable nodes it is referenced by. This is done by setting the *dirty flag* on such nodes. These nodes will, in turn, inform their parent nodes, and so forth. Figure 8.6 illustrates how dirty propagation works. By changing the value of the reference of variable node, the dirty flag propagates via function node up to the root.

8.5.4 Constant Folding

An expression may contain several constant values. Every value is included in the expression tree as a *value node*. After resolving the references, constant subtrees may be reduced to a single node, provided that no other nodes reference the respective subtree. This is done as follows: First, the node is checked for constancy. If it is constant, it is then transformed into a value node, unless it already was a value node to begin with. For a node to

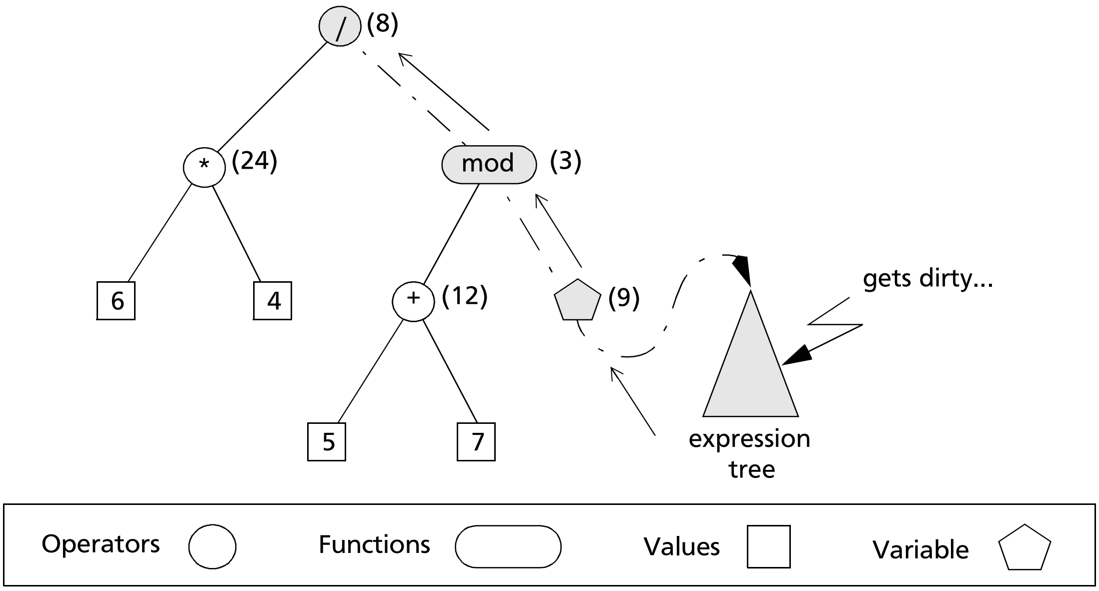


Figure 8.6 Dirty-Tag propagation within an expression tree

decided whether or not it is constant, it may have to check its child nodes. However, it can be said that value nodes are always constant, while variable nodes never are.

Figure 8.7b depicts the expression tree from Figure 8.7a after constant folding. The function node is not constant, because its right child node is a variable node. As a consequence, the operator node, cannot be constant either.

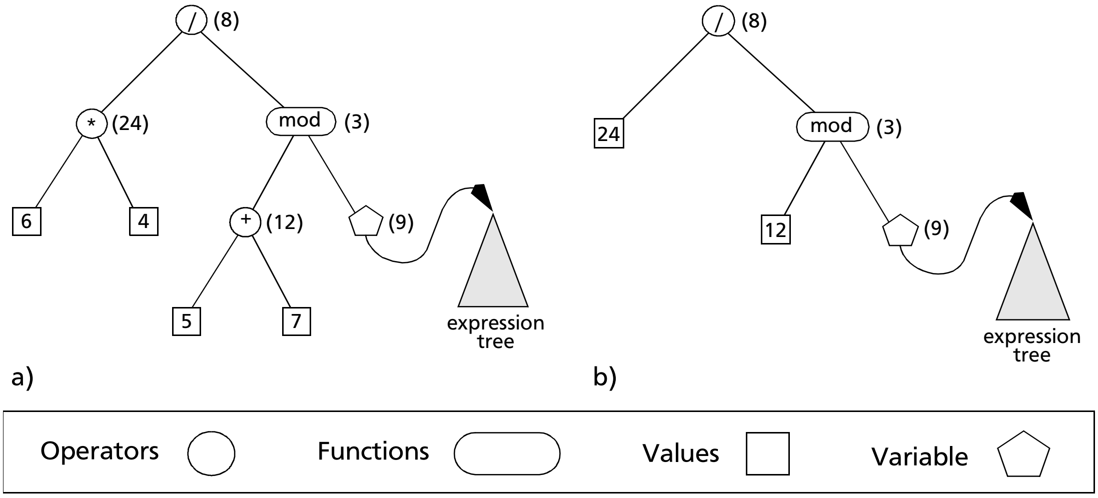


Figure 8.7 Example of an expression tree reduced through constant folding

8.6 EXAMPLE

The following example illustrates the usage of IVML. We define a simple, static graph consisting of 6 data objects and 12 interconnections. The associated connectivity matrix is given by:

The IVML code fragment depicted below contains the full definition of the graph. We start with a prototype definition of a generic data object from which all subsequent

Table 8.2 Connectivity matrix of the example shown below

	T1	T2	R1	R2	R3	R4
T1			1	1	1	1
T2			1	1	1	1
R1	1	1		1		1
R2	1	1	1		1	
R3	1	1		1		1
R4	1	1	1		1	

instances are derived. The prototype contains a default initialization of information assigned to all objects, such as color or scale factors. Note that the geometry of the prototype object is hardwired in the plug-in *SimpleData*, which itself is derived from the *BaseData* class (see also Appendix A.5). Next, two individual objects (ellipses in Figure 8.8) are instantiated overwriting some of the default parameters of the prototype. In particular, we overwrite the object appearance by an VRML 2.0 expression, which can be either inlined or given by a URL. Likewise, we generate the four spherical objects. The subsequent prototype definition of connection objects is used by the following expressions, which describe the links between all objects. It can be seen, that the wildcard expressions in IVML, such as "`R*`" tremendously simplify the definition of the graph topology of our example.

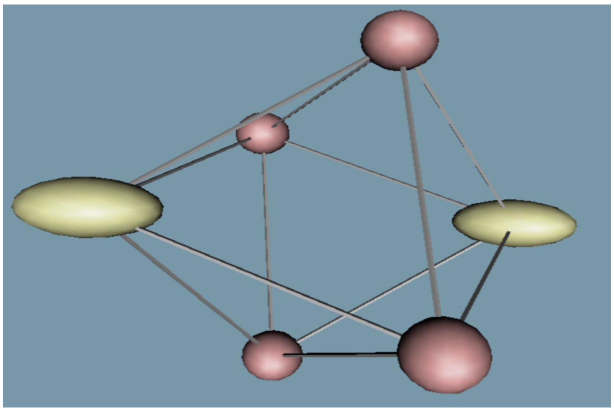


Figure 8.8 Layout computed from IVML script presented below

Corresponding IVML-Script:

```
#IVML V1.0
# Example: Simple Objects

# First set some global parameters
IvorySystem {
  info IVInfo {
    title "Simple Objects"
    info  ["Ivory V2.0 Example"]
  }
}

# Define a prototype for our
# data objects
PROTO data {
    SimpleData {
      label "Simple Object "+me.name
      spaceFactor 5.0
      visual IVVisual {
        color <1,0.5,0.5>
      }
    }
  }

  # Define the 2 Top-Objects special
  DEF T1 data {
    visual IVVisual {
      color <1,1,0.5>
    }
    appearance INLINE “ellipse.wrl”
  }
  DEF T2 data {
    visual IVVisual {
      color <1,1,0.5>
    }
    appearance INLINE “ellipse.wrl”
  }

  # Define the ring objects
  DEF R1 data {}
  DEF R2 data {}
  DEF R3 data {}
  DEF R4 data {}


  # Define a prototype for our
  # connection object
  PROTO conn {
    SimpleConn {
      defaultLength 10.0
    }
  }

  # make the connections
  # connect first top with all
  # objects in ring
  NameConnector{
    template conn {}
    names ["T1","R*"]
  }

  # connect second top with all
  # objects in ring
  NameConnector{
    template conn {}
    names ["T2","R*"]
  }

  # connect the ring objects
  conn {
    label "R1_R2"
    leftObj R1
    rightObj R2
  }
  conn {
  label "R2_R3"
    leftObj R2
    rightObj R3
  }
```

```
conn {
  label "R3_R4"
  leftObj R3
  rightObj R4
}
conn {
  label "R4_R1"
  leftObj R4
  rightObj R1
}
```

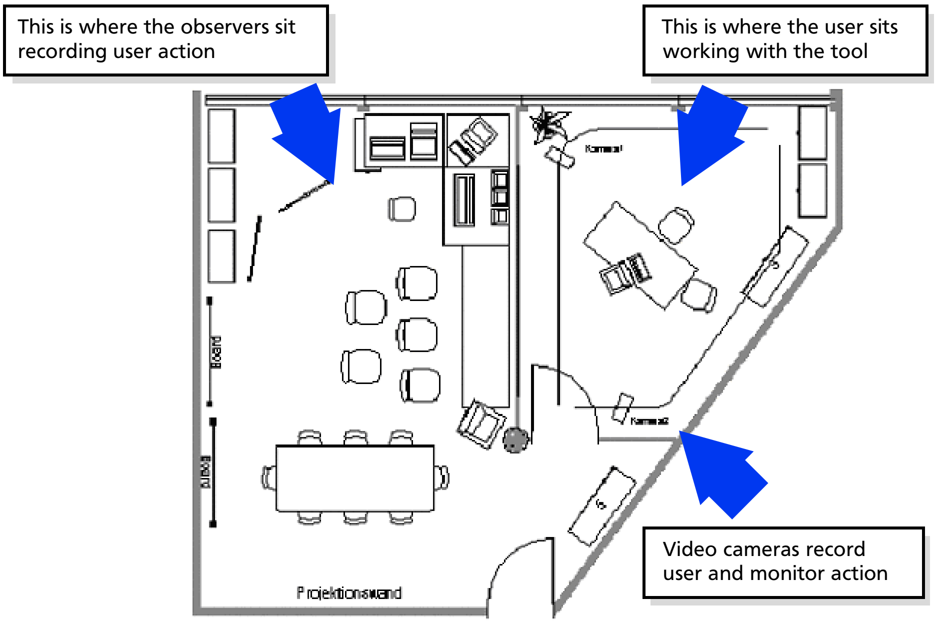
9

RESULTS

9.1 USABILITY STUDY

9.1.1 Objectives and Limitations

The present chapter describes the results of usability tests with the data visualization toolkit (IVORY) conducted from January 31 to February 4, 2000, at the usability lab of the software ergonomics department of UBS. The tests did not, as is generally the case, focus on ergonomic aspects of the application, but rather on the general applicability of a new visualization paradigm to document retrieval. Due to the limited test set size (6 test users), only qualitative statements could be expected.



This is where the observers sit recording user action

This is where the user sits working with the tool

Video cameras record user and monitor action

Figure 9.1 Floor plan of the UBS usability lab.

9.1.2 The UBS Usability Lab

The usability lab at UBS Zürich-Altstetten provides the infrastructure for systematic and effective usability testing. The lab consists of two rooms separated by a one-way mirror, i.e. an observation room and a testing room (compare Figure 9.1). In the testing room, the user is left to himself while working through the test tasks with the application. In the observation room, his actions are monitored and journalized, and design ideas are discussed. Video facilities allow recording of monitor and user actions.

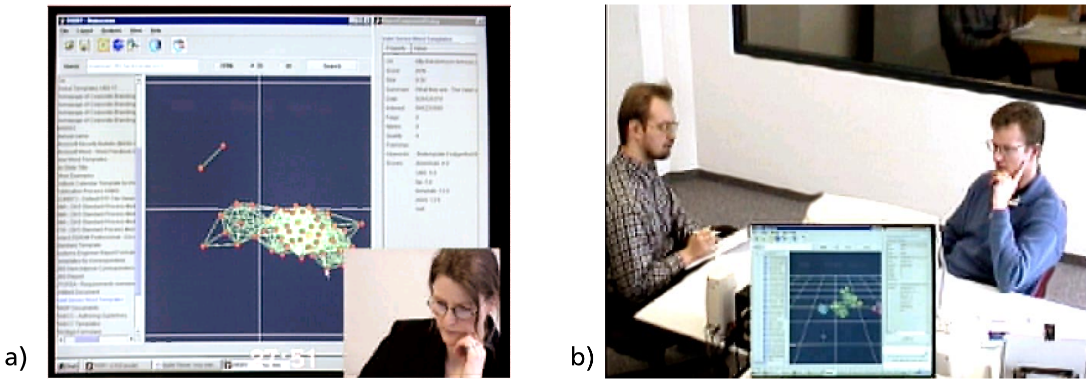


Figure 9.2 Video snapshots of a typical session: a) A test person currently working on an exercise, b) User and expert in the concluding personal interview

9.1.3 Scenario

The usability tests were conducted with a total of 6 test users (1 female, 5 male), all of them working in the IT department of UBS. After a short introduction on how to use the tool, each of the test users had to work through a total of 10 test tasks with the tool (see Appendix C for a copy of the questionnaire). The tasks consisted of finding textual information in the bank's internal network (Intranet). The tool offers various visualization techniques to support the search (HTML list, 2D visualization and 3D visualization). For the first six tasks, the visualization technique to be used was stipulated in the questionnaire. For the remaining four tasks, the choice was left to the user. During the testing, the users were left undisturbed to work through the tasks in the testing room, while usability engineers monitored their actions via the one-way mirror from the observation room. All user actions were logged and recorded on video.

9.1.4 Hypothesis

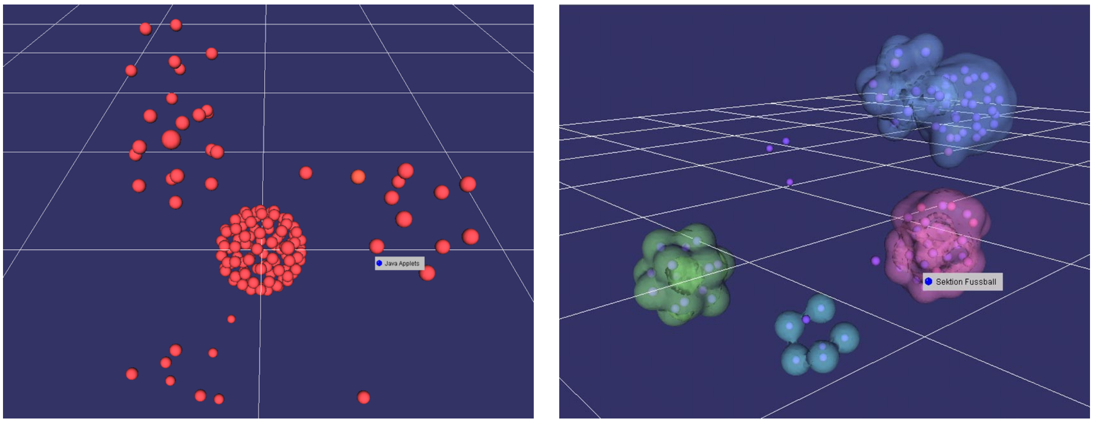
Our usability study aimed at putting to the test the following two hypotheses:

- 2D and 3D visualization are well suited to explorative searching process.
- For focused searching, the traditional HTML list presentation is preferred.

9.1.5 Observations and Results

All in all, the users appeared to have no problems using the tool. The results obtained in the tests basically confirmed the above hypotheses.

The users opted for 3D or 2D visualization of their own accord whenever the task asked for consulting an overview or making a comparison, for example for estimates on frequency or for explorative searching in cases where the clues were few. Navigation in 3D search space seemed to hardly pose a problem. Yet it has to be said that the tasks were solved just as efficiently with 2D visualization, and the 2D navigation was described as simpler. For the presentation of complex data, however, having a third dimension was recognized as an advantage.



a) b)

Figure 9.3 Two typical IVORY scenes from this test showing clustered Intranet documents arranged according to subject (see Appendix 11). a) Shows the result of scenario 2, no visual clustering, 150 documents b) Presents a snapshot of scenario 10, BLOB clustering, 100 documents (see Color Plate 5 on page 184).

Due to the simplifications introduced by the tool, users quickly learned how to apply the 3D presentation techniques (relaxing, clustering). Clustering in particular was repeatedly described as helpful. Some users stated that they would prefer the tool to offer further simplification by encapsulating the required steps in one function.

For focused searches, e.g. where the search criteria were clear, conventional list presentation was generally preferred, yet some of the users successfully solved the tasks using 2D or 3D presentations. List presentation was frequently described as impractical.

9.1.6 Conclusions

In the usability lab, the tool left a surprisingly good impression and caused hardly any problems. In real-life environments, the visualization approach is expected to provide a useful alternative to conventional methods. Particularly for the presentation of complex data involving comparisons and estimates, 3D visualization is clearly advantageous.

Assessment given by the UBS Usability Lab: “In our opinion, the visualization approach implemented in the tool provides an innovative, useful and functional alternative to conventional data presentation that is particularly useful for the presentation of complex data.”

9.2 APPLICATIONS AND COVERAGE EXAMPLES

The following four application examples are intended to give an impression of the variety and flexibility of the IVORY framework. Note however that the 2D pictures of this section do not reveal the full 3D arrangement computed by our method.

9.2.1 Prediction of Long Term Interest Rates

In this example our approach is applied to the visualization and analysis of the correlation of long term interest rates with other important economic parameters.

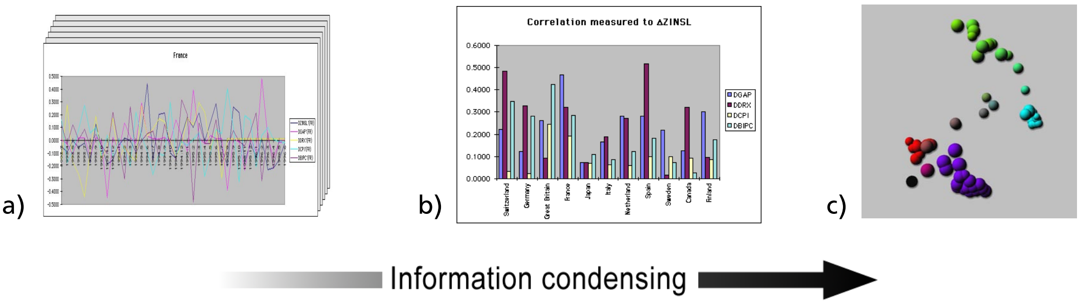
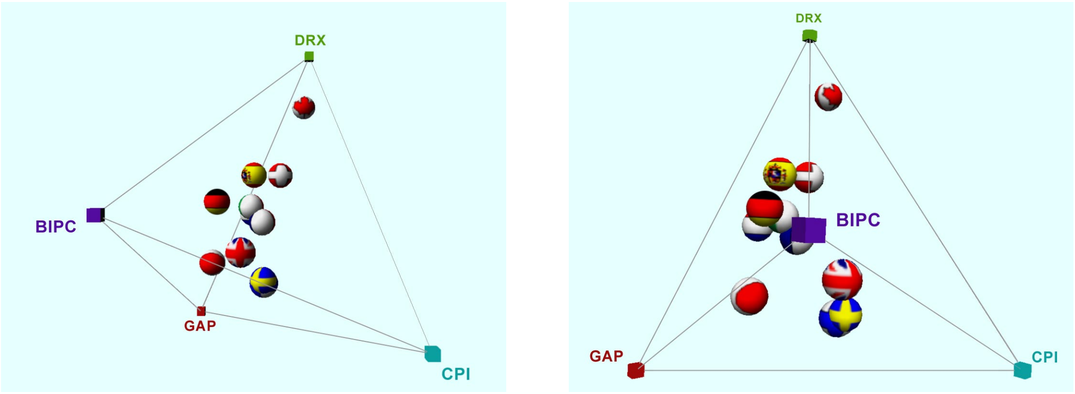


Figure 9.4 Condensing multivariate relationships: a) Stack of conventional diagrams b) Correlation tables c) Physically-based visualization paradigm.

First, our method is compared to a traditional way of analyzing multidimensional relationships of economic indicators. The goal was to evaluate the influence of the indicators presented in the diagrams of Figure 9.4a on the long term interest rates of individual countries. Each of these indicators was computed relative to the USA as a reference. The most common approach consists in producing bar charts, as depicted in Figure 9.4b, showing the correlation with individual indicators for different countries. These charts form a basis for further interpretation performed by the financial analyst. In order to map the problem onto our visualization paradigm we start from a special instance of our model. By imposing displacement constraints we first generate a subset of four objects which keep their position during relaxation. In order to arrange these objects favorably for the resulting visualization they are positioned each at the corners of a imaginary tetrahedron. Our indicators are mapped on these object types. All the other objects represent the countries and are connected via links to all rigid objects from above. The spring stiffness of a link corresponds to the correlation of the associated indicator to the long term interest rates of this country. Note that the movable objects are not interconnected.



a) b)

Figure 9.5 Two views showing the influence of individual economic indicators onto the long term interest rates of different countries (Data source: courtesy of UBS AG, Switzerland)

Figure 9.5 displays two views of the relaxed model. The cubes at the vertices of the tetrahedral structure stand for the different indicators taken into account and the spheres representing the countries are textured with their flags. Although the definition of individual indicators is beyond the scope of this paper we observe that the interest rates of Canada correlate tightly with the index DRX, whereas Switzerland relates more closely to GAP and CPI. Conversely, Germany is located near the center of gravity of the plane spanned by DRX, BIPC and GAP and is hence equally influenced by those measures.

The performance of the blobby clustering method (see Section 7.3.2 on page 106) is illustrated in the series of Figure 9.6, where different snapshots are presented as obtained from interactive settings of the isovalue. All blobs were equally weighted by the parameter a of Equation 7.16. Cluster sets of a single object are colored in red, those of two or three objects are in green whereas larger sets are rendered in blue. As expected, the system discriminates elegantly Canada – since heavily influenced by DRX – from the rest of the analyzed countries. Conversely, countries equally influenced by all indicators, such as France or Switzerland are grouped together in Figure 9.6b.

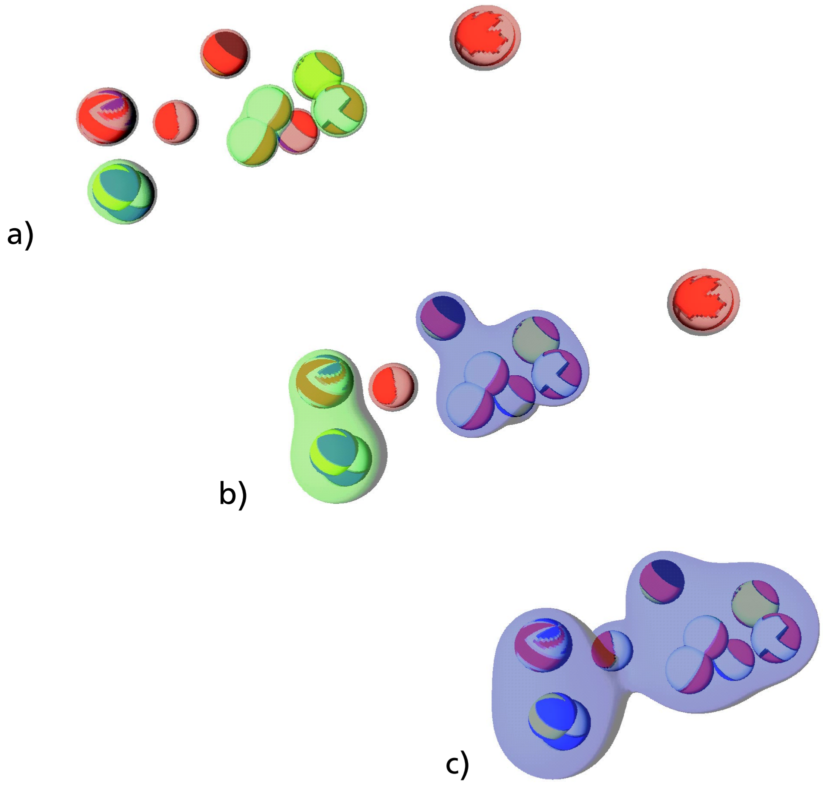


Figure 9.6 Clustering the results of Figure 9.5: a) a = 6.0 and c_{iso} = 0.1, b) a = 1.0 and c_{iso} = 0.9, c) a = 0.55 and c_{iso} = 1.1.

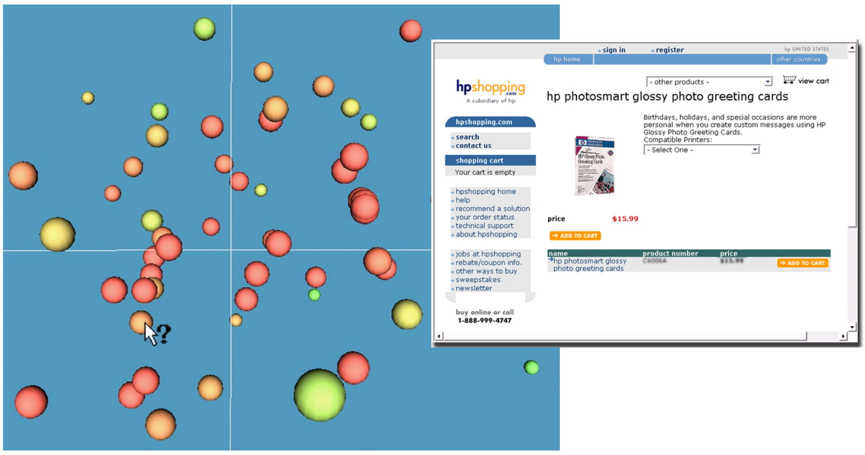
9.2.2 Shopping-Basket Analysis

The following example emerged as a feasibility study from a 3 month stay in 1999 at Hewlett-Packard (HP) Research Laboratories, Palo Alto, USA. The main goal of this knowledge transfer was to enable HP's visual mining platform *VisMine* [HP©] to use the IVORY technology. The main area of application consisted in the visual analysis of large e-commerce log files. With the kind consent of HP some of the results can be presented as a part of this work. Additional information has been published in [HDH+00, HDH+01, HHD+01].

Market basket analysis has become a key success factor in e-commerce. However, it is a challenge to mine customer's purchasing behavior from the millions of Internet transactions we typically find in a productive e-commerce systems. Such transactions are often composed of several products (items) that are purchased together in the same shopping basket. Understanding relationships across hundreds of product lines and among millions of transactions provides insight and predictability into product affinity and purchasing behavior.

The basic idea is to define a metric measuring the closeness of relationships between items that co-occur in transactions. As a result, the system visualizes frequent item sets by means of object clusters in 3D space. The chosen technique provides a fast and intuitive way for e-commerce managers to navigate through large-volume purchasing data in order to easily find product affinities by locating corresponding clusters in the visualization.

At HP Laboratories, the system has been used to visually mine over a dataset containing 500,000 transactions covering 600 different products for market basket analysis. The image below shows a view on spatially clustered product objects – computed by the *VisMine* system – that were frequently bought together.



hp photosmart glossy photo greeting cards

Birthdays, holidays, and special occasions are more personal when you create custom messages using HP Glossy Photo Greeting Cards.

Compatible Printers: - Select One -

price $15.99

name	product number	price
hp photosmart glossy photo greeting cards		

buy online or call 1-888-999-4747

Your cart is empty

Figure 9.7 Snapshot of a *VisMine* scene showing a set of clustered product objects. Additionally, detailed info is shown in an extra for a product of special interest. [HP©].

The raw data for the analysis consists of a huge amount of session log data coming from the HP's shopping village Internet store. It's internal structure is made up of a simple list of user transactions

$$List_{Trans} = (T_1, T_2, \ldots, T_N).$$

Each *transaction* T_i is composed of a set of products P_j that were bought in this transaction

$$T_i = \{P_1, \ldots, P_{M_i}\}, \text{ where } i \in [1 \ldots N].$$

Each *product* P_j is defined by a set of *attributes* A_k, such as type, product description, color or prize

$$P_j = \{A_1, \ldots, A_{L_j}\}, \text{ where } i \in [1 \ldots N].$$

The main idea is to represent the products in question as simple spherical objects in space. Furthermore, the connection between two objects is parametrized by the frequency of occurrence of the associated products in the same transaction. For the reason of a more efficient access during the following computations, we reorganize the data as follows. Basically, we merge the information distributed over T_i and P_j into an *extended product* P_j^*. This extended product contains all attributes A_k as well as a complete list PT_j of all transactions in which the product P_j occurs:

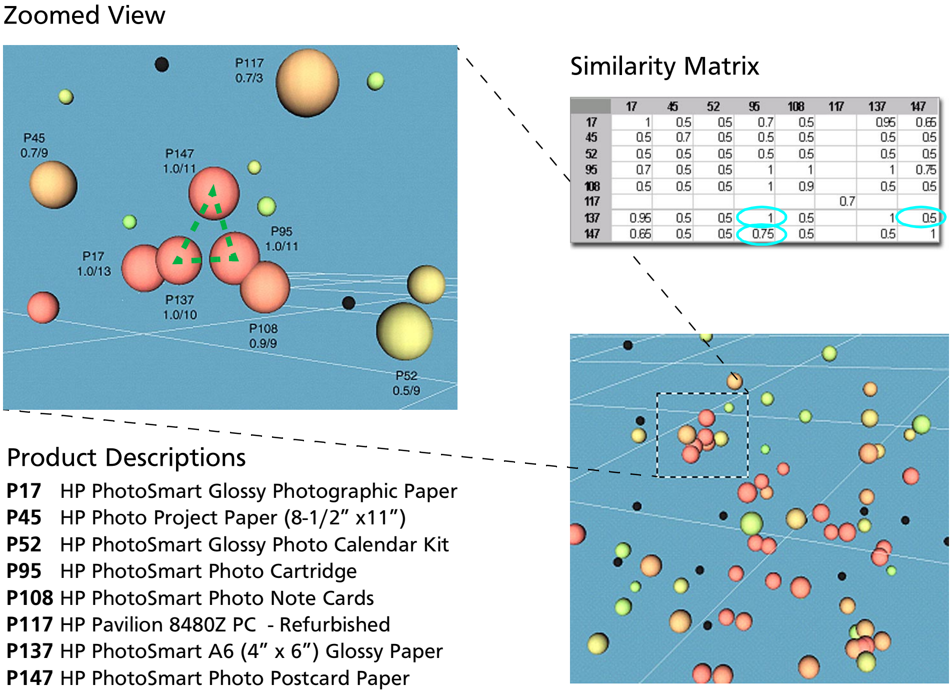
$$P_j^* = \{A_1, \dots, A_{L_j}, PT_j\}, \text{ with } PT_j = \{T_i | P_j \in T_i\}$$

We thus define the problem-specific similarity as

$$s_{mn} = s(P_m^*, P_n^*) = \frac{|PT_m \cap PT_n|}{max\ PT_j}$$

After the layout process has converged we discover several object clusters in space. One of them has been captured in Figure 9.8. The color of the spheres codes the product frequency. Red indicates that a product has been bought often, whereas yellow means that the corresponding product has been bought rather rarely.

We may easily locate a cluster of products, which have been frequently bought in the same shopping basket. The zoomed view shows more detail. As it is marked by the dashed triangle one may recognize a group of five spatially close product objects. Three of them (P95, P137 and P147) are part of a regular triangular relation structure, each having a high similarity with both others. The two additional product objects (P17, P108) are mainly associated with a single product. These observations are confirmed also by the values of the similarity matrix. We conclude that products of the PhotoSmart family are often



	17	45	52	95	108	117	137	147
17	1	0.5	0.5	0.7	0.5		0.95	0.65
45	0.5	0.7	0.5	0.5	0.5		0.5	0.5
52	0.5	0.5	0.5	0.5	0.5		0.5	0.5
95	0.7	0.5	0.5	1	1		1	0.75
108	0.5	0.5	0.5	1	0.9		0.5	0.5
117						0.7		
137	0.95	0.5	0.5	1	0.5		1	0.5
147	0.65	0.5	0.5	0.75	0.5		0.5	1

Product Descriptions

P17	HP PhotoSmart Glossy Photographic Paper
P45	HP Photo Project Paper (8-1/2″ x11″)
P52	HP PhotoSmart Glossy Photo Calendar Kit
P95	HP PhotoSmart Photo Cartridge
P108	HP PhotoSmart Photo Note Cards
P117	HP Pavilion 8480Z PC - Refurbished
P137	HP PhotoSmart A6 (4″ x 6″) Glossy Paper
P147	HP PhotoSmart Photo Postcard Paper

Figure 9.8 Different views on the results of the shopping basket analysis

bought together. In particular, the products **P95** *HP PhotoSmart Photo Cartridge*, **P137** *HP PhotoSmart A6 (4" x 6") Glossy Paper* and **P147** *HP PhotoSmart Photo Postcard Paper* can be found very frequently in the same basket.

9.2.3 Document Retrieval Visualization

This example is derived from a real world document retrieval research project. We applied our new technique to a hit list (result list) originating from an intranet document query. The actual case consists of 100 single document objects. To each of them a feature vector is attached, keeping all keywords (search words) with their corresponding frequency. All these vectors define a keyword space, where feature vectors exhibit a similar distribution of key words point in the same direction. With the aim of grouping documents treating similar topics close together, similarity has been defined by using the *angle measurement* from Section 3.4.3 on page 26.

For the visual clustering stage we applied the H-BLOB technique with a defined maximum of 20 clusters. We then repeatedly merged 50% of the clusters, yielding 6 hierarchy levels with 20, 10, 5, 3, 2 and 1 clusters, respectively. In Figure 9.9 we show 4 selected images from this session.

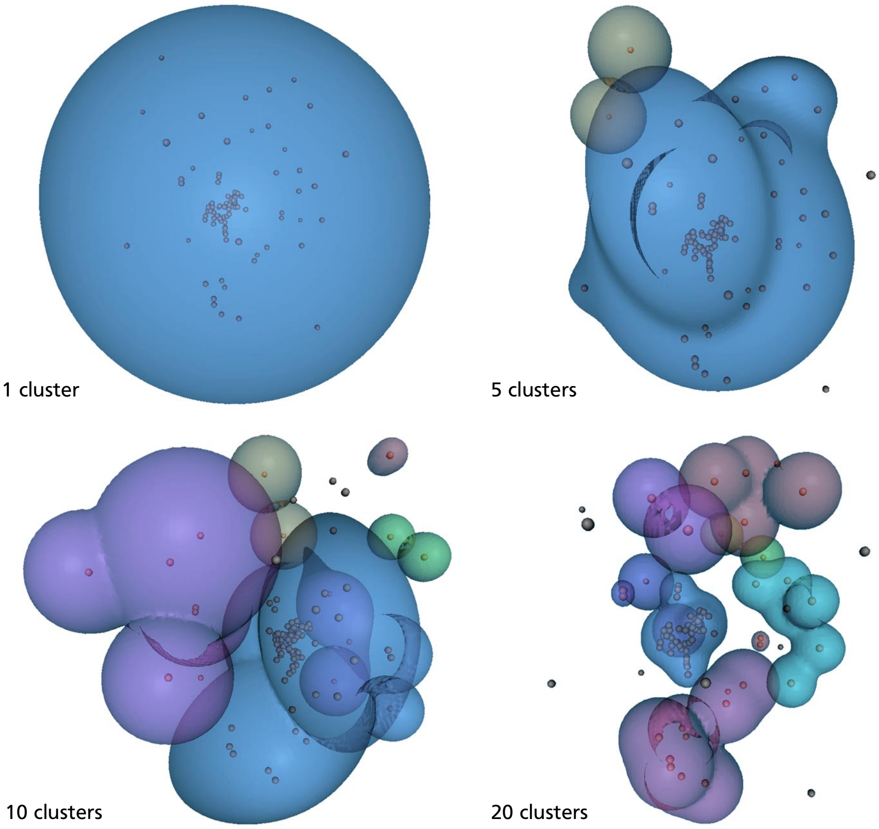


Figure 9.9 Document Retrieval Visualization using H-BLOB clustering. Cluster hierarchies are shown with 20, 10, 5 and 1 cluster (see Color Plate 6 on page 185).

9.2.4 Image Retrieval Example

This problem stems from the area of analysis and retrieval of images based on their content in very-large databases. The retrieval process requires a fast but also efficient feature extraction mechanism as well as a powerful interaction paradigm allowing the user to navigate through the image space. Here, our concern is obviously with the latter part.

Our example originates from the research project *Chariot* [WPL01] done by Roger Weber. Currently, he is a member of the database system research group [DBS01] of Prof. H.-J. Schek at ETH. This example has been chosen not only to demonstrate new visualization approaches but also to illustrate complementary techniques of analytical and visual retrieval procedures in the field of high-dimensional and complex data spaces.

In order to characterize color images, we apply a technique that extracts, according to [SO95], a 9-dimensional feature vector. The different feature values arise as follows: First, each pixel is transformed from the *RGB* into the *Lab*-space [FVF+90]. There, the first two momenta of the color channels are computed. The following list assigns the values to a position within the feature vector:

Table 9.1 Assignment of feature vector values

Pos	Value	Description
1	mean(L)	mean value (average)
2	mean(a)	
3	mean(b)	
4	var(L)	variance
5	var(a)	
6	var(b)	
7	cov(L,a)	co-variance
8	cov(a,b)	
9	cov(b,L)	

Note that all values are normalized by the standard distribution of the respective component over all pictures. Each image is identified by an unambiguous object ID (OID) and its URL to the location where the actual image data is retrieved.

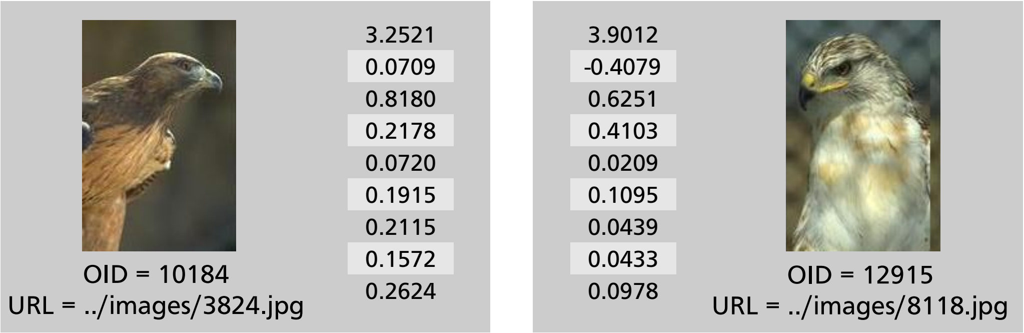
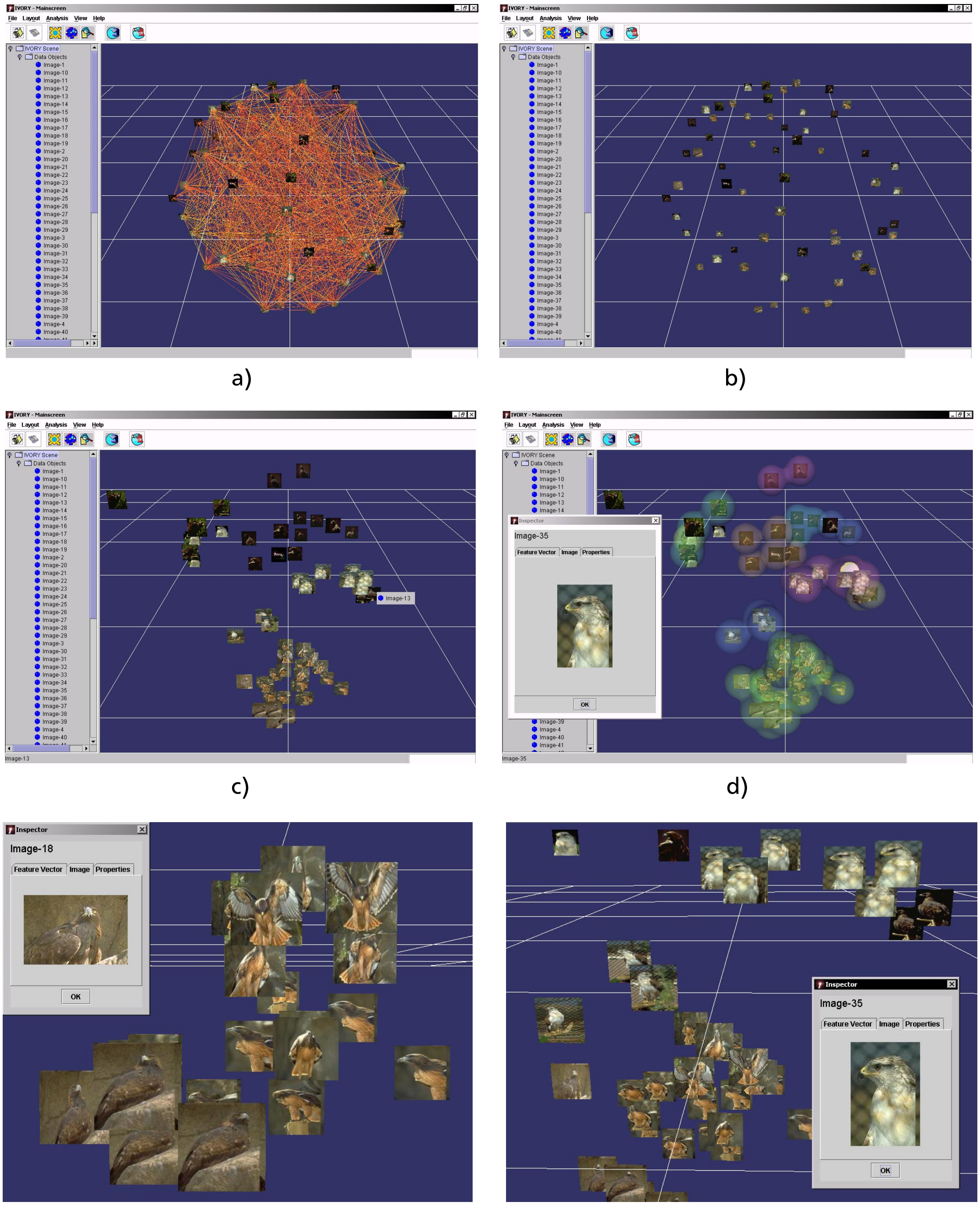


Figure 9.10 Two different images with their corresponding feature vectors

In order to measure the similarity between two images we apply the *distance measurement* formula for n-dimensional vectors (see Section 3.4.3 on page 26) to the two associated feature vectors. The pictures below illustrate the nice results we got from the IVORY system.



a) b)

c) d)

e) f)

Figure 9.11 Screen shots from the image retrieval example: a, b) Initial arrangement on a virtual spherical surface c-f) Different views of resulting object layout automatically arranged by the IVORY framework (see Color Plate 7 on page 186).

As one may recognize easily for this example the applied methods work as expected. Pictures showing the same visual characteristics are layouted in close distance to each other. This results in very nice and compact clusters that are especially well recognizable in Figure 9.11e and Figure 9.11f.

In Figure 9.11d the visualization is enriched with semi-transparent BLOB-clusters in order to visually support the object clusters. In addition, Figure 9.11d-f show an image inspector that was poped up as a result of a drill-down interaction.

10

CONCLUSIONS AND OUTLOOK

We presented a new variant for physics-based information visualization and illustrated its versatility. The fundamental idea bases on a quantification of the similarity of related objects, which governs the parameters of a mass-spring model. Relaxation of the model figures out the structural relations in information space.

In contrast to many other information visualization systems, our approach is supported by a solid framework of established theories. Thus, the amount of heuristics has been reduced to a minimum, which makes parametrization of such complex systems principally predictable.

Additionally, we presented a new approach towards a portable, object-oriented framework especially designed for physics-based information visualization. The system is open and expandable by adding new plug-ins. With the multi-layered abstraction model for users we provide adequate interfaces to configure our system at different abstraction levels. This covers a fast visualization prototyping using predefined plug-ins, but also a very flexible low-level system access.

We also introduced a new script language named Information Visualization Modeling Language (IVML), which is distinctively proposed to describe information visualization problems. As an interpreted high-level language it enables users to efficiently describe individual visualization problems and to build fast prototypes. The language is open, object-oriented and features scene graphs.

Another contribution of this work is a set of new visual clustering algorithms - called Ellipsoidal, BLOB, and H-BLOB clustering - which provides an efficient level-of-detail strategy and are consequently capable to cluster and visualize very large and complex data volumes. The key concept is an efficient multi-resolution setup, breaking down the structural and visual complexity of scenes.

We have shown the framework's versatility by experimental results, demonstrating its capability to visualize large and complex volumes of different types of abstract data. Furthermore, a usability test including daily-business cases in the areas of our cooperation partner has validated IVORY's performance in practice.

Future work has to encompass the development of hierarchically organized layout subsystems, which make use of the object clustering analysis embedded into the system. We expect a tremendous speedup for the layout.

Additionally, more effort has to be spent on the aspects of self-parametrization of algorithms. Up to now, for each new application domain quite an number of parameters have to be set manually by an expert. By finding rules that are able to determine suitable sets of initial parameter, the framework would take the next step in the direction of a user-friendly tool.

With the tremendous speed of technical progress in the area of graphics hardware, new possibilities of visualizing data using more meaningful representations becomes possible, even on standard desktop computers or notebooks. Thus, further efforts have to consider more advanced data representation techniques that achieve superior visual quality.

We are convinced that the physically-based approach fits also nicely advanced I/O concepts with force and tactile feedback. Only, for the benefit of other research topics little effort has been spent on this aspects. However, any work in this area will emphasize a natural human-computer interface and mediates an additional sensoric cue to the user.

A

PACKAGE LAYOUT

This chapter shows an overview of important IVORY classes. The arrangement is organized according to IVORY's package structure.

A.1 IVORY.BACKEND

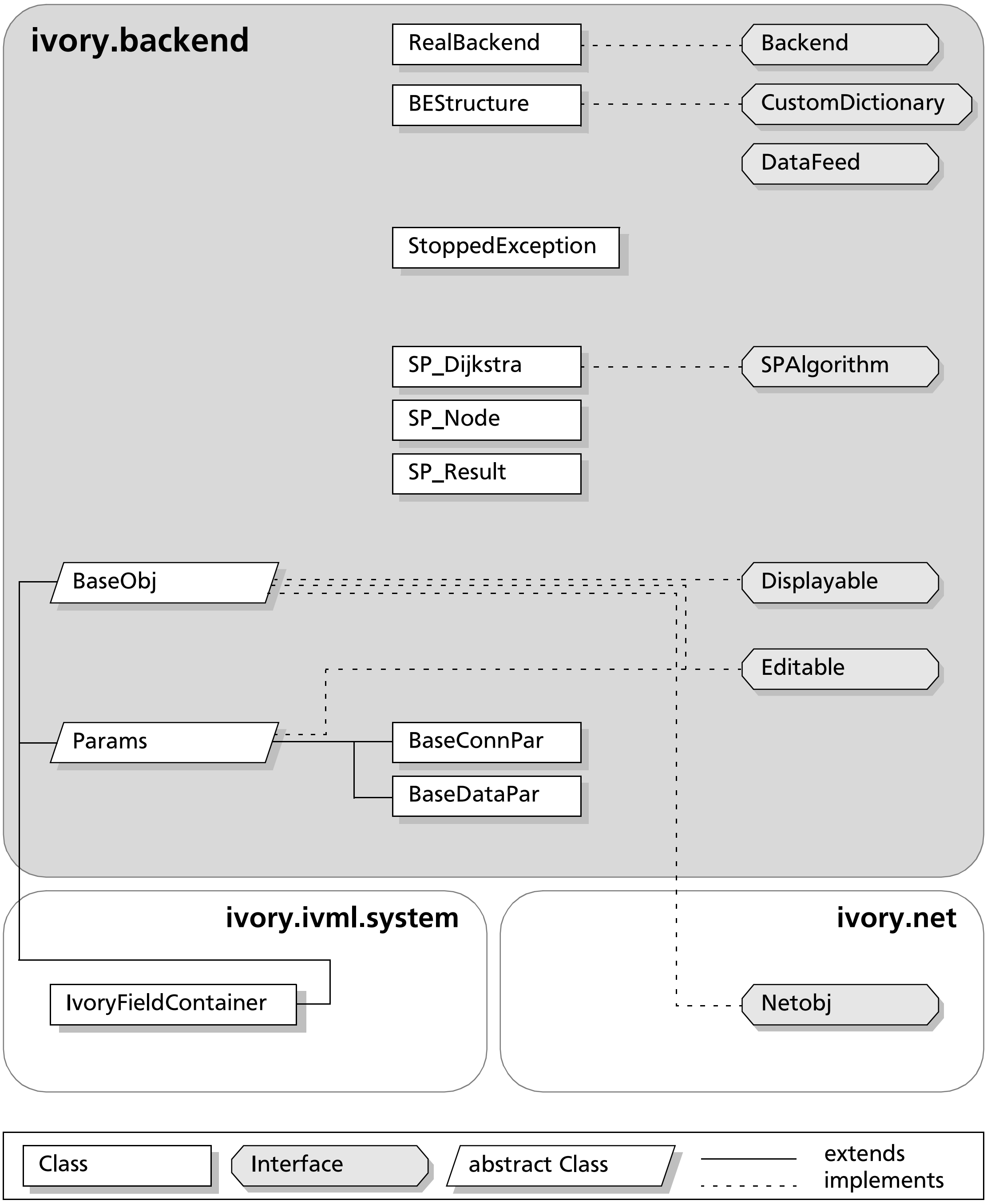


Figure A.1 Layout of package ivory.backend

A.2 IVORY.BACKEND.CLUSTER

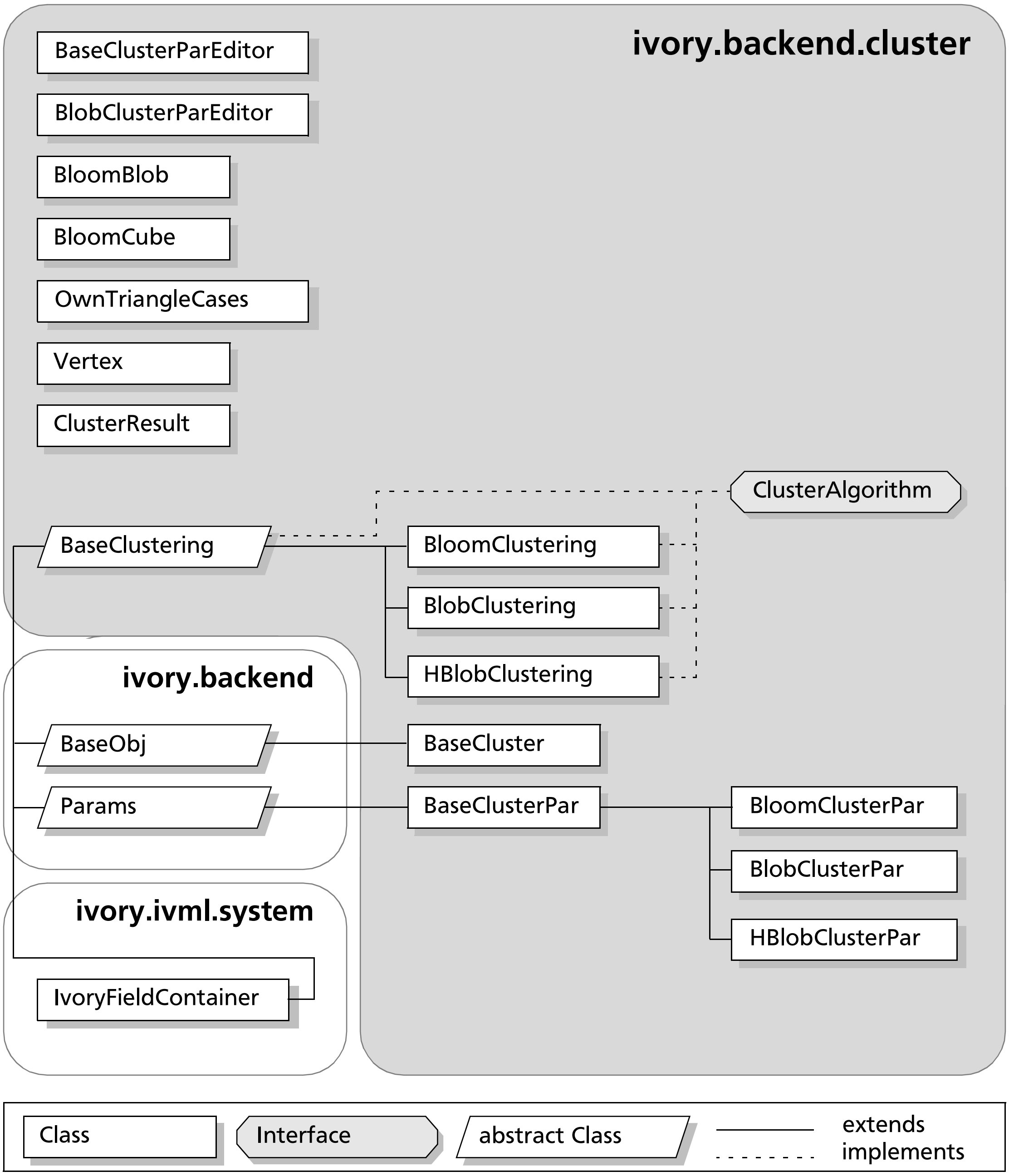


Figure A.2 Layout of package ivory.backend.cluster

A.3 IVORY.BACKEND.LAYOUTSYSTEM

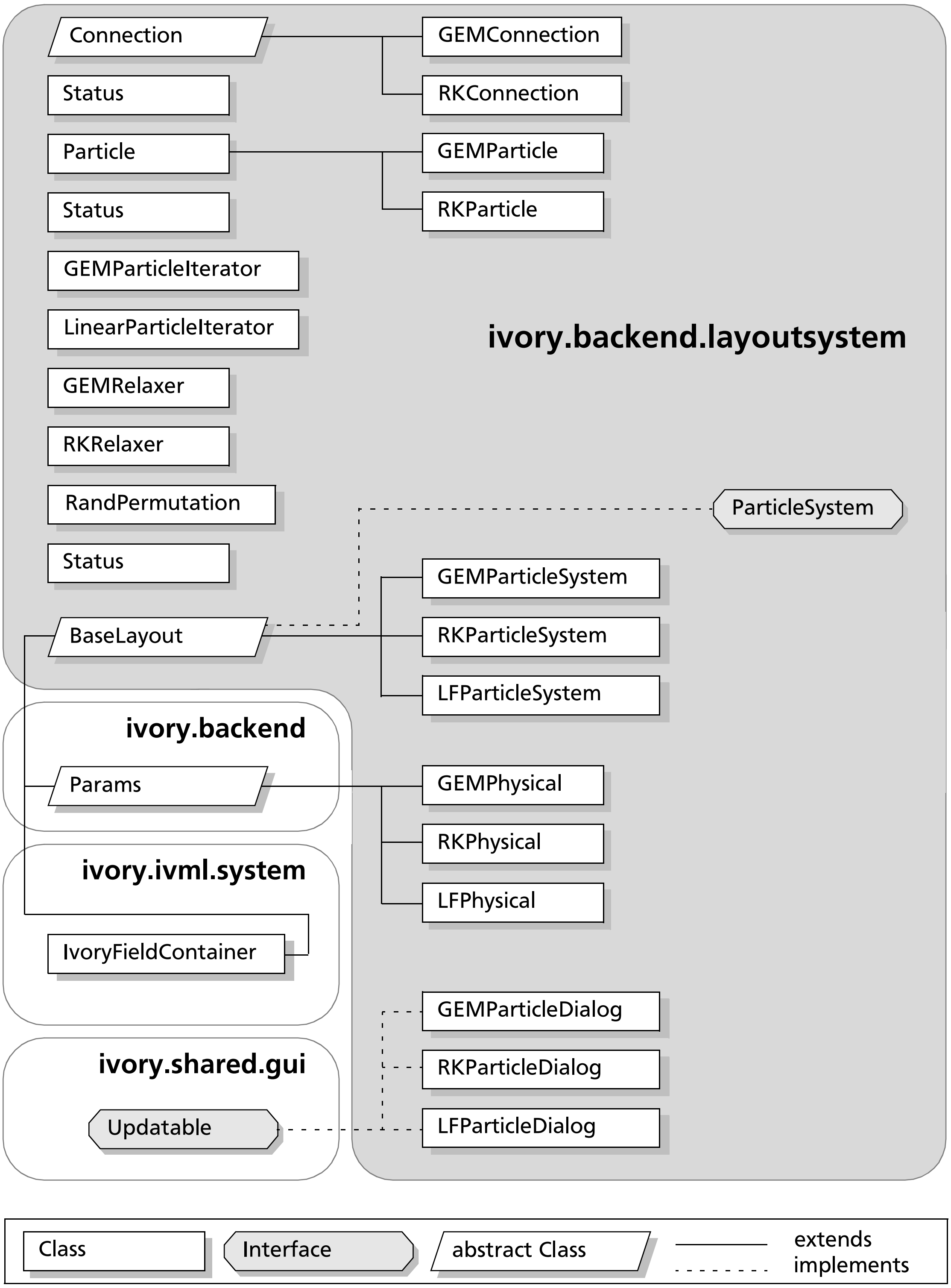


Figure A.3 Layout of package ivory.backend.layoutsystem

A.4 IVORY.IVML.SYSTEM

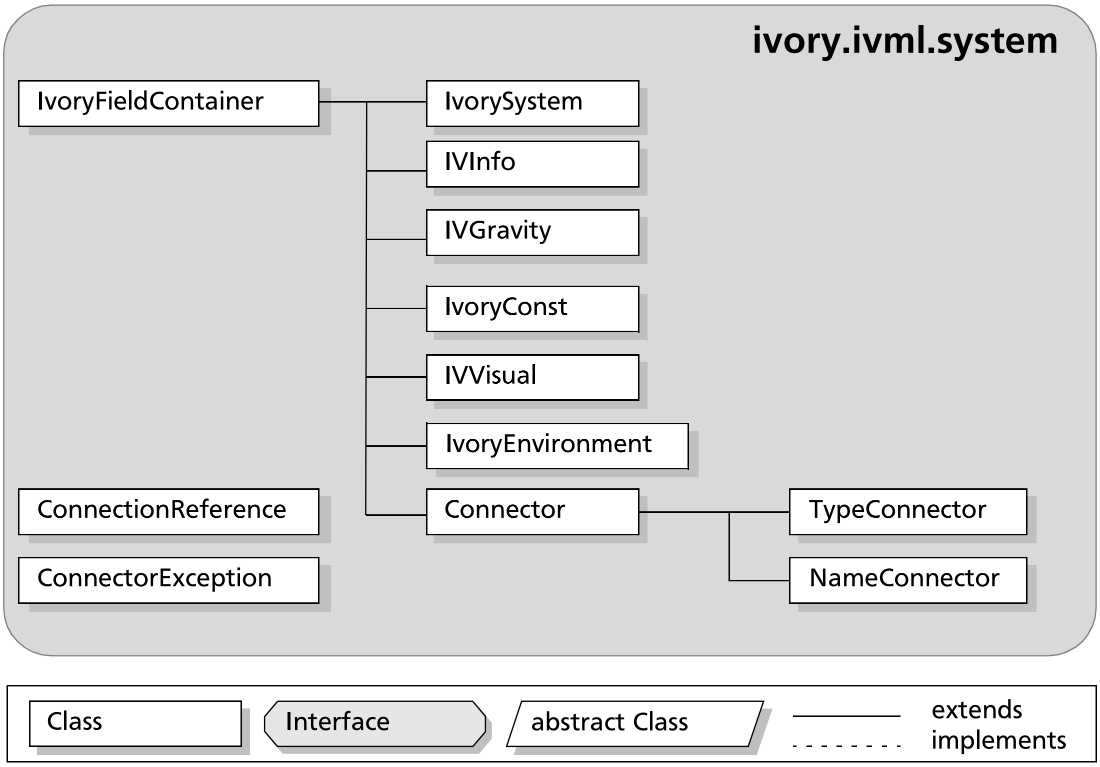


Figure A.4 Layout of packages ivory.ivml.system

A.5 IVORY.PLUGIN

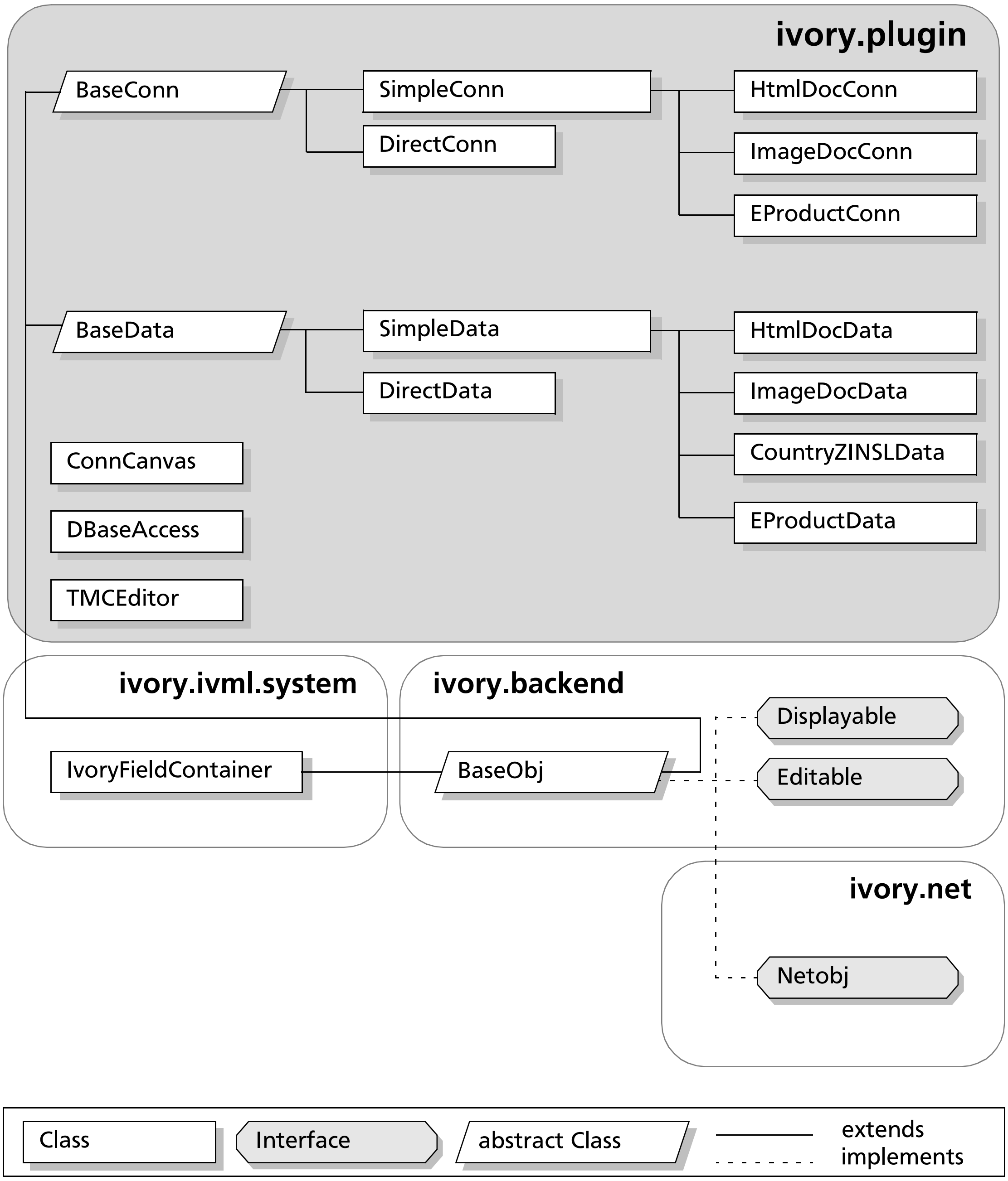


Figure A.5 Layout of package ivory.plugin

A.6 IVORY.IVML

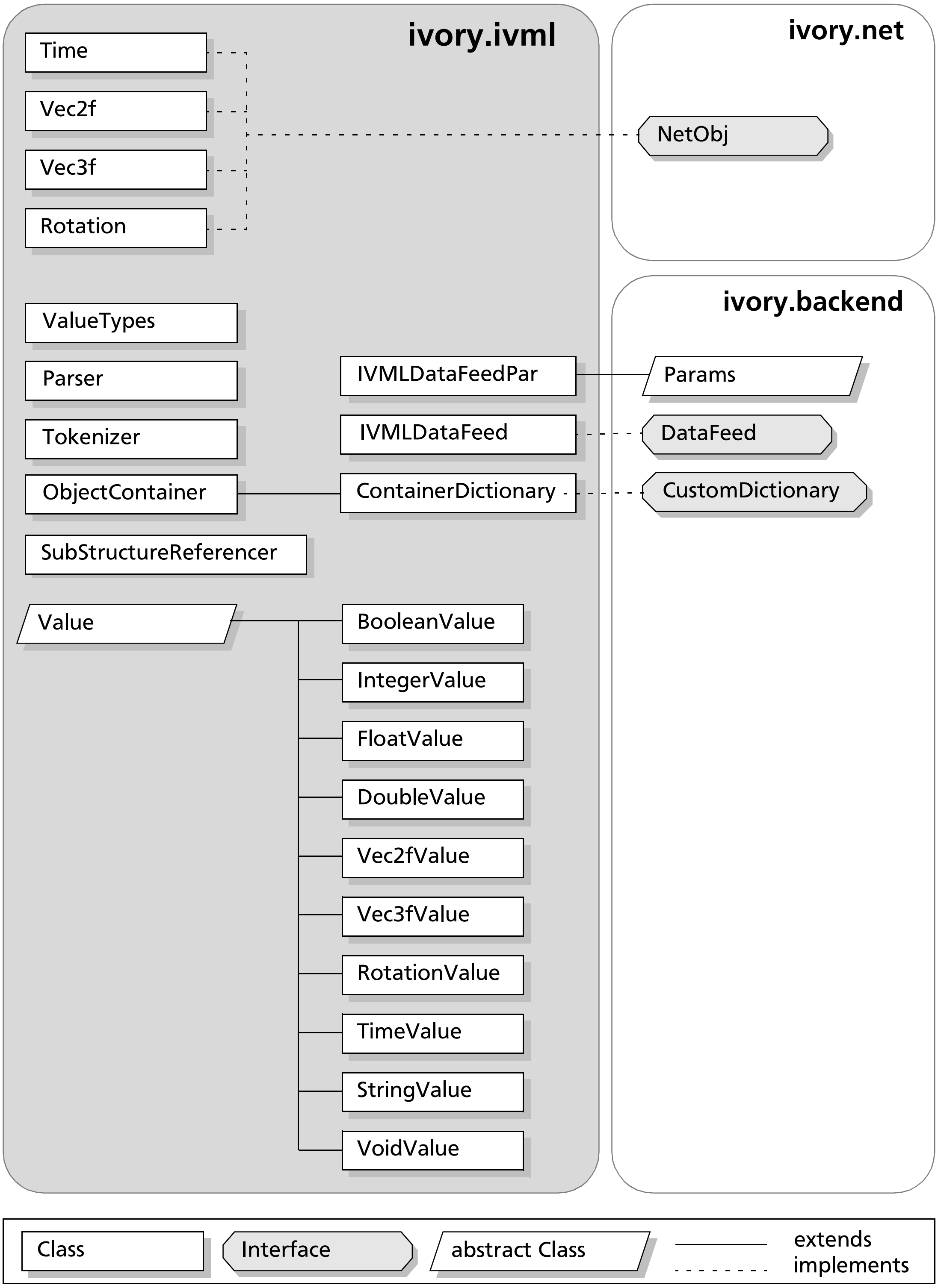


Figure A.6 Layout of package ivory.ivml

THE GRAMMAR OF IVML

This chapter presents a grammar for the *Information Visualization Modeling Language* (IVML). The grammar presented gradually in Chapter 8 was suitable for the purpose of explanation. However, the grammar presented in this chapter uses a formal EBNF notation and can thus serve to implement an IVML parser.

General remarks on the notation of the EBNF and script language.

- Variables are written in *italic*.
- Terminal symbols are written in **bold**.
- Alternations are indicated with “|”.
- Repetitions (zero or more occurrences) are enclosed by “{“ and “}”.
- Options (zero or one occurrences) are enclosed by “[“ and “]”.
- “<“ and “>” stand for: any character from.
- The ‘#’ (0x23) character starts a comment, provided that it occurs outside of a quoted string. The subsequent characters up to the end of the line are treated as whitespaces.
- Whitespaces are: CR (0x0d), LF (0x0a), space (0x20) and tab (0x09) provided that they occur outside of a quoted string.
- In the subsequent EBNF all information about whitespaces are left out in order to increase the legibility. It should be clear by intuition, where whitespaces must happen and where not. For example, it is obvious that between the individual digits of an *integerValues* no whitespaces must happen. On the other hand, between a keyword and one identifier a whitespaces are allowed.

B.1 BASICS

alphaChar := **<abcdefghijklmnopqrstuvwxyz>**
| <ABCDEFGHIJKLMNOPQRSTUVWXYZ>.

digit := <0123456789>.

hexDigit := **<0123456789abcdefABCDEF>**.

alphanumChar := *alphaChar* | *digit.*

escapedChar := “**!**” | < **0x24** - **0x7f** > | “**\n**” | “**\t**” | “**\\”**” | “**\r**” | “ “ | “\\”.

identifier := (*alphaChar* | “_”) { *alphanumChar* | “_” }.

qualident := ((“**me.**” (*identifier* | “**parent**”)) | *identifier)*
{ “.” (*identifier* | “**parent**”) } [“[“ *expression* “]”].

wsChar := ‘**0x09**’ | ‘**0x0a**’ | ‘**0x0d**’ | ‘**0x20**’ | (‘#’ { *anyChar* } ‘**0x0a**’).

ws := *wsChar* { *wsChar* }.

B.2 VALUES

integerVal := (*digit* { *digit* })
| (“**0x**” | “**0X**”) *hexDigit* { *hexDigit* }.

floatVal := *digit* “.” { *digit* }.

stringVal := ““““ { *escapedChar* } ““““ .

vec2fVal := “<“ *sum* “,” *sum* “>”.

vec3fVal := “<“ *sum* “,” *sum* “,” *sum* “>”.

rotationVal := “<“ *sum* “,” *sum* “,” *sum* “,” *sum* “>”.

timeVal := *integerVal timeDim*
| *dateVal*
| “IV_TIME_FIRST” | “IV_TIME_LAST”
| “IV_TIME_NULL”.

timeDim := “**ns**” | “**us**” | “**ms**” | “**s**” | “**m**” | “**h**” | “**d**” | “**mon**” | “**y**” | “**f**”.

dateVal := *dayVal* “.” *monthVal* “.” *yearVal*
[“:” *hourVal* [“:” *minuteVal* [“:” *secondVal*]]].

dayVal := *digit* [*digit*].

monthVal := (*digit* [*digit*])
| “**Jan**” | “**Feb**” | “**Mar**” | “**Apr**” | “**Mai**” | “**Jun**”
| “**Jul**” | “**Aug**” | “**Sep**” | “**Oct**” | “**Nov**” | “**Dec**”.

yearVal := *digit digit* [*digit digit*].

hourVal := *digit digit.*

minuteVal := *digit digit.*

secondVal := *digit digit.*

boolVal := “TRUE” | “FALSE”.

nodeVal := “““ (*VRMLSceneGraph* | *fileName*) “““.

VRMLSceneGraph := see syntax of vrml

fileName := “file:” *pathName* | *url.*

routeVal := “““ *identifier* “.” *identifier* “TO” *identifier* “.” *identifier* “““.

B.3 EXPRESSIONS

expression := *sum* [*compOp sum*].

compOp := “=” | “<=” | “>=” | “<“ | “>” | “!=”.

sum := *product* { *sumOp product* }.

sumOp := “+” | “-”.

product := *term* { *prodOp term* }.

prodOp := “*” | “/”.

term := [“-”]
(*constant*
| *function*
| *variable*
| “(“ *expression* “)”).

constant := *rotationVal* | *vec3fVal* | *vec2fVal* | *timeVal* | *floatVal* | *integerVal*
| *boolVal* | *stringVal* | *nodeVal.*

function := *identifier* “(“ [*expression* { “,” *expression* }] “)”.

variable := *qualident.*

B.4 DECLARATION

definition := *namedClassDescr* | *classDescr.*

namedClassDescr := “DEF” *identifier classDescr.*

classDescr := *classIdent* “{“ {*fieldDef*} “}”.

classIdent := *identifier.*

fieldDef := *svFieldDef*
| *mvFieldDef*
| *subFieldDef.*

svFieldDef := *identifier expression.*

mvFieldDef := *identifier* “[“ [*expression* { (“,” | “;”) *expression* }] “]”.

subFieldDef := *identifier classDescr.*

protoDecl := “**PROTO**” *identifier* “{“ { *classDescr* } “}”.

B.5 SCRIPT

ivoryScript := *ivmlHeader*
constSection
systemSection | *inline*
vrmlSection | *inline*
protoSection
defSection
connSection
dsrcSection.

ivmlHeader := “**#IVML 1.0 \n**”

constSection := { *constDef* | *inline* }.

inline := “**INLINE**” “““ *fileName* “““.

constDef := “**DEF**” *identifier* “**IvoryConst**” “{“ { *svFieldDef* } “}”.

systemSection := [“**DEF**” “**system**”] “**IvorySystem**” “{“ { *fieldDef* } “}”.

vrmlSection := [“**DEF**” “**env**”] “**IvoryEnvironment**” “{“ { *fieldDef* } “}”.

protoSection := { *protoDecl* | *inline* }.

defSection := { *definition* | *inline* }.

connSection := { *connDef* | *inline* }.

connDef := (“**NameConnector**” | “**TypeConnector**”) “{“ { *fieldDef* } “}”.

dsrcSection := { *definition* | *inline* }.

C

USABILITY TEST SCENARIO

(IN GERMAN)

This chapter hold a copy of the questionary, that was used for the usability test running from January 31 to February 4, 2000, at the usability lab of the software ergonomics department of UBS. Full particulars concerning the test can be gleaned in Section 9.1 on page 139.

Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Physikalisch-basierte Informationsvisualisierung

Thomas Sprenger

IVORY - Anwenderstudie

31.1.2000 – 4.2.2000
Usability Lab UBS, Zürich

1 Einführung

Im Kontext dieses Projekts entsteht ein plattform-unabhängiges Framework (IVORY) zur physikalisch-basierten Visualisierung multidimensionaler Datenrelationen. Die Implementation ist vollständig in Java 1.2 realisiert. Das Softwaredesign orientiert sich an der Theorie der Operatorklassifikation und der Operator-Frameworks. Individuelle, visuelle Metaphern und Interaktionsparadigmen (Information Drill-down) werden modular durch einen sogenannten Plugin-Mechanismus auf den entsprechenden Operatorlevels ins System eingebunden. Die Konfiguration von IVORY erfolgt über eine Skriptsprache, genannt IVML (Information Visualization Modeling Language).

Die Funktionsweise des Systems basiert auf der Quantifizierung der Ähnlichkeit von in Beziehung stehender Objekte, die wiederum die Parameter eines Masse-Feder-Systems konfigurieren. Weil die Federhärte und die berechneten Ähnlichkeitswerte korrespondieren, konvergiert das System in ein energetisches Minimum, das die inhärent in den Daten enthaltenen Beziehungen und Zusammenhänge als räumliche Nachbarschaften widerspiegelt. Die so berechneten Objektkonfigurationen werden in einer dreidimensionalen Ansicht dem Benutzer präsentiert. Das Framework erlaubt damit eine effiziente (visuell-) explorative Analyse grosser Datenvolumen. Über die definierten Paradigmen kann dabei mit den Daten direkt interagiert werden. Zusätzlich erlaubt das Framework komplexe Szenarien mittels verschiedener Verfahren (Ellipsoid, Blob, H-Blob, etc.) visuell und analytisch zu clustern.

Das Projekt wird in Kooperation mit dem Advanced Engineering Center (ACE) der UBS Basel durchgeführt. Aus diesem Grund liegt das Hauptanwendungsgebiet für das System im Bereich der Finanzdienstleistungen zu denen unter anderem auch das Documentretrieval im Intranet gehört.

Diesen Anwendungsbereich fokusiert auch die vorliegende Anwenderstudie. Am Beispiel des Dokumentretrievals soll eine Verifikation des Frameworks erfolgen. Es wird dabei vorausgesetzt, dass die berechneten Objektkonfigurationen der Visualisierung, die in den Daten enthaltenen Relationen in genügend guter Qualität reflektieren. D.h. die Studie konzentriert sich auf die Analyse und Beurteilung des für viele Benutzer neuen Mensch-Maschinen-Interfaces.

Der grobe Aufbau der Studie besteht aus drei Teilen. In einem ersten Beispiel sollen Sie das Framework und dessen Komponenten ein wenig kennenlernen. Insbesondere können Sie sich mit der Navigation in der dreidimensionalen Szene ein wenig vertraut machen. Im zweiten Teil werden Ihnen verschiedene Szenarien präsentiert, welche Sie mit einer der vorgegebenen Visualisierungstechniken bearbeiten sollen. Zum Schluss können Sie die im dritten Teil der Studie gestellten Aufgaben mit einer von Ihnen frei gewählten Visualisierung lösen.

Einführungsbeispiel

Im IVORY Framework wir zur Visualisierung der Ergebnisse eine 3D-Darstellung verwendet. Diese erlaubt es dem Benutzer, sich frei durch den Ergebnisraum zu bewegen. Es sind dabei Bewegungen in allen 6 Freiheitsgraden möglich (Verschiebungen und Rotationen). Weil die Navigation in solchen virtuellen Räumen für viele Benutzer neu und dementsprechend ungewohnt ist, startet diese Studie mit einem kleinen Einführungsbeispiel. Dabei können Sie sich gleichzeitig mit der Navigation (Maus und Tastatur), wie auch mit den verschiedenen Tastenbelegungen von IVORY vertraut machen.

Öffnen Sie die Datei "Einführungsbeispiel.ivml" und warten Sie bis alle Objekte geladen sind. Ihre Aufgabe ist es nun die Datenobjekte in einer vorgegeben Reihenfolge zu besuchen. Hinter jedem Objekt versteckt sich als Information ein Wort. Werden die einzelnen Worte in der richtigen Reihenfolge aneinander gehängt, so ergibt sich die gesuchte Lösung. Gestartet wird mit der gelben Kugel in der Mitte. Die weitere Folge der Besuche verläuft von dort immer entlang der grünen Verbindungen. Viel Spass!

Antwort/Bemerkungen:

....

....

....

2 Aufgaben

1. Szenario

Situation: Sie bekommen von einem Kunden eine Anfrage. Er hätte von Ihnen gerne den aktuellen Zins auf dem UBS Privatkonto gewusst.

Queryvorschlag: +Zins +Privatkonto

Visualisierungsart: HTML Liste

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7

Antwort/Bemerkungen:

....

2. Szenario

Situation: In Ihrer Abteilung wird ein neues Projekt mit Java gestartet. Weil die Kompetenz auf diesem Gebiet in Ihrer Umgebung noch nicht so gross ist, fragen Sie sich, welche anderen Abteilungen in der UBS sich mit Java beschäftigen. Wenn es eine ausgezeichnete Abteilung gibt, machen Sie mit einem der Verantwortlichen gleich einen Termin ab.

Queryvorschlag: Java

Visualisierungsart: 2D

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7

Antwort/Bemerkungen:

....

3. Szenario

Situation: Für eine Präsentation sollen Sie eine Zusammenstellung der Abteilungen machen, die in der UBS Forschung betreiben. Insbesondere interessiert dabei der Anteil des Bereichs Economical Research?

Queryvorschlag: Forschung Research

Visualisierungsart: 3D

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7

Antwort/Bemerkungen:

....

4. Szenario

Situation: Ein neu zu erstellender Bericht soll über die Verbreitung, der in der UBS eingesetzten Programmiersprachen informieren. Schätzen Sie ab über welche Programmiersprachen in der UBS am meisten publiziert wird?

Queryvorschlag: java c cpp C%2B%2B smalltalk pascal basic cobol

Visualisierungsart: HTML Liste

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Antwort/Bemerkungen:

....

5. Szenario

Situation: Sie sollen einen offiziellen UBS-Fax mit aktuellen Informationen an einen Kunden senden. Deshalb suchen Sie auf dem Intranet nach einem Fax-Template für Word. Zur weiteren Bearbeitung laden Sie dieses auf die lokale Festplatte.

Queryvorschlag: download UBS fax template word

Visualisierungsart: 2D

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)

☐ ☐ ☐ ☐ ☐ ☐ ☐
1 2 3 4 5 6 7

Antwort/Bemerkungen:

....

6. Szenario

Situation:	Sie starten in Java ein neues Projekt. Gerne profitieren Sie von den Erfahrungen anderer Entwickler. Dies ist umso effizienter, wenn gleiche Tools eingesetzt werden. Welches ist in der IT Schweiz die Standardentwicklungsumgebung für Java?
Queryvorschlag:	IDE +Java Standard Entwicklungsumgebung
Visualisierungsart:	3D Szene

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7

Antwort/Bemerkungen:

....

7. Szenario

Situation:	Welche Application Server werden im Projekt WAP (WASP) als Standard empfohlen?
Queryvorschlag:	application server standard wasp
Visualisierungsart:	Nach freier Wahl des Benutzers

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7
Welche Visualisierungsart haben Sie für diese Aufgabe gewählt?	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐ ☐ ☐ ☐ ☐ ☐ ☐ 1 2 3 4 5 6 7

Antwort/Bemerkungen:

....

8. Szenario

Situation: Ein Kollege, welcher erst seit kurzem in der UBS arbeitet, fragt Sie nach einer Abteilung in der UBS, welche Fonts und Templates anbietet?

Queryvorschlag: Vorlagen Templates Schriften Fonts Logo

Visualisierungsart: Nach freier Wahl des Benutzers

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Welche Visualisierungsart haben Sie für diese Aufgabe gewählt?	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7

Antwort/Bemerkungen:

....

9. Szenario

Situation: Für einen Bekannten, der sich sehr für den Bereich Java-Programmierung interessiert, suchen Sie ein internes Stellenangebot für einen Java-Entwickler. Wer bietet in diesem Bereich offene Stellen an?

Queryvorschlag: job offer +java Entwickler Developer

Visualisierungsart: Nach freier Wahl des Benutzers

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Welche Visualisierungsart haben Sie für diese Aufgabe gewählt?	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐☐☐☐☐☐☐ 1 2 3 4 5 6 7

Antwort/Bemerkungen:

....

10. Szenario

Situation: Die UBS bietet ein breites Spektrum an sportlichen Aktivitätsmöglichkeiten. Welche Sportart ist die häufigste in der UBS?

Queryvorschlag: Fussball Schiessen Squash Tennis Laufen Fahrrad

Visualisierungsart: Nach freier Wahl des Benutzers

Was ist Ihr subjektiver Eindruck betreffend der benötigten Zeit zum Lösen der Aufgabe? (1 = sehr kurz, 7 = sehr lange)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Welche Visualisierungsart haben Sie für diese Aufgabe gewählt?	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie gut wurden Sie durch die verwendete Visualisierungsart in Ihrer Aufgabe unterstützt (1 = sehr schlecht, 7 = sehr gut)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Sind Sie mit dem gefundenen Ergebnis zufrieden? (1 = überhaupt nicht, 7 = ausserordentlich)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7
Wie schätzen Sie die Vollständigkeit Ihrer Lösung ein? (1 = mangelhaft, 7 = komplett)	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7

Antwort/Bemerkungen:

....

Besten Dank für Ihre Mitarbeit.

REFERENCES

[AM75] M. A. Arbib, E. G. Manes. *Arrows, Structures, and Functors: The Categorical Imperative.* Academic Press: New York, 1975.

[App85] A. W. Appel. *An Efficient Program for Many-Body Simulation.* Siam, J. Sci. Stat. Comput.6 (1985): 85-103, 1985.

[ATT97] XGobi, a System For Multivariate Data Visualization, *http://www.research.att.com/~andreas/xgobi*, AT&T Labs Research, 1997.

[AVS97] AVS Homepage, *http://www.avs.com*, Advanced Visual Systems Inc., 1997.

[AW95] C. Ahlberg, E. Wistrand. "IVEE: An Information Visualization and Exploration Enviroment". *Proceedings International Symposium on Information Visualization*, Atlanta, GA, pp. 66-73, 1995.

[Ben96] C. L. Bentley. "Animating Multidimensional Scaling to Visualize N-Dimensional Data Sets". *Proceedings of the IEEE Information Visualisation 96*, pp. 72-73, 1996.

[BET+94] G. d. Battista, P. Eades, R. Tamassia and I. G. Tollis "Algorithms for Drawing Graphs: An Annotated Bibliography". *Computational Geometry: Theory and Apllications*, 4(5):235-282, 1994.

[BET+98] G. d. Battista, P. Eades, R. Tamassia and I. G. Tollis. *Graph Drawing.* Prentice-Hall, Inc., 1998.

[BF95] I. Bruss, A. Frick. "Fast Interactive 3-D Graph Visualization". *Proceedings of Graph Drawing 95*, pp. 99-110, Springer Verlag, 1995.

[BFN85] C. Batini, L. Furlani and E. Nardelli. "What is a Good Diagram? A Pragmatic Approach". *Proceedings 4th Intl. Conf. on the Entity Relationship Approach*, 1985.

[BFR98] P. Bradley, U. Fayyad and C. Reina. "Scaling Clustering Algorithms to Large Databases". *4th Int'l conference for Knowledge Discovery and Data Mining*, Menlo Park, CA, 9-15, AAAI Press, 1998.

[BH86] J. Barnes, P. Hut “A Hierarchical O(n log n) Force Calculation Algorithm”. *Nature*, (324):446-449, 1986.

[Bli82] J. F. Blinn “A Generalization of Algebraic Surface Drawing”. *ACM Transactions on Graphics*, pp. 235–256, 1982.

[Blo94] J. Bloomenthal. *An Implicit Surface Polygonizer*. Graphics Gems IV, Academic Press, pp. 324-349, 1994.

[BM76] J. A. Bondy, U. S. R. Murty. *Graph Theory with Applications*. Macmillan, London, 1976.

[But63] J. C. Butcher “Coefficients for the study of Runge-Kutta integration processes”. *Austral. Math. Soc.*, 3, pp. 185-201, 1963.

[But65] J. C. Butcher “On the attainable order of Runge-Kutta methods”. *Math. comp.*, 19:408-417, 1987.

[But87] J. C. Butcher. *The numerical analysis of ordinary differential equations*. Chichester: John Wiley, 1987.

[BW98] D. Baraff, A. Witkin. "Large Steps in Cloth Simulation". *Proceedings SIGGRAPH 98*, pp. 43-54, 1998.

[CEG96] S. Card, S. G. Eick and N. Gershon. "Information Visualization". *SIGGRAPH 96 Course Notes 8*, 1996.

[Cha96] M. Chalmers. "A Linear Time Layout Algorithm for Visualizing High-Dimensional Data". *Proceedings of the IEEE Information Visualization 96*, pp. 127-132, 1996.

[Chr99] Toolkits Offer Fast, Easy-To-Use 3D Data Visualization, *http://java.sun.com/features/1999/08/int.html*, SUN, 1999.

[CK95] J. Carriere, R. Katzman. "Research Report - Interacting with Huge Hierarchies: Beyond Cone Trees". *Proceedings of the IEEE Information Visualization 95*, pp. 74-81, 1995.

[CLR94] T. H. Cormen, C. E. Leiserson and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1994.

[CM97] S. K. Card, J. Mackinlay. "The structure of the information visualization design space". *Proceedings of the IEEE Information Visualization 97*, pp. 92-99, IEEE CS Press, 1997.

[CMS99] S. Card, J. Mackinlay and B. Shneiderman. *Readings in Information Visualization: using vision to think*. Morgan Kaufmann Publishers, Inc., 1999.

[Col66] L. Collatz. *The Numerical Treatment Of Differential Equations*. Springer, 1966.

[Cos97] Cosmo Software Homepage, *http://cosmosoftware.com/products/player*, Silicon Graphics Inc., Mountain View, CA, 1997.

[CR96] M. C. Chuah, S. F. Roth. "On the semantics of interactive visualization". *IEEE Visualization '96*, pp. 29-36, IEEE Press, 1996.

[CR98] E. H. Chi, J. T. Riedl. "An Operator Interaction Framework for Visualization Systems". *Proceedings of the IEEE Symposium of Information Visualization 98*, 1998.

[CRM91] S. K. Card, G. G. Robertson and J. D. Mackinlay. "The Information Visualizer, an Information Workspace". *CHI '91*, 181-188, 1991.

[CS90] P. J. Channell, C. Sovel “Sympletic Integration of Hamiltonian Systems”. *Nonlinearity*, 3:231-259, 1990.

[DBS01] Database System Research Group Homepage, *http://www.dbs.ethz.ch/*, 2001.

[Dem96a] Fast Hierarchical Methods for the N-body Problem, Part 1, *http://www.cs.berkeley.edu:80/~demmel/cs267/lecture26/lecture26.html*, Lecture Notes, U. C. Berkeley, 1996.

[Dem96b] Fast Hierarchical Methods for the N-body Problem, Part 2, *http://www.cs.berkeley.edu:80/~demmel/cs267/lecture26/lecture26.html*, Lecture Notes, U. C. Berkeley, 1996.

[DO74] B. S. Duran, P. L. Odell. *Cluster Analysis: A Survey*. Springer-Verlag, 1974.

[Ead84] P. Eades. "A heuristic for Graph Drawing". *Congressus Numerantium*, pp. 149-160, 1984.

[Eng69] R. England “Error estimates for Runge-Kuta type solutions to systems of ordinary differential equations”. *Comput. J.*, 12:166-170, 1969.

[Epp95] D. Eppstein “Dynamic Euclidean minimum spanning trees and extrema of binary functions”. *Discrete & Computational Geometry*, 13(1):111-122, 1995.

[Ess92] K. Esselink, “The order of Appel's algorithm”, *Information Processing Letters*, Vol. 41, pp. 141-147, 1992.

[Est+96] M. Ester, e. al. “A Density-Based Algorithmfor Discovering Clusters in Large Spatial Databases with Noise”. *Int'l Conference for Knowledge Discovery Databases and Datamining*, Menlo Park, 226-231, AAAI Press, 1996.

[Eul36] L. Euler, “Solutio problematis ad geometriam situs pertinentis”, *Commetarii Academiae Scientiarum Imperialis Petropolitanae*, Vol. 8, pp. 128-140, 1736.

[Feh69] E. Fehlberg “Klassische Runge-Kutta-Formeln fünfter und siebter Ordnung mit Schrittweiten-Kontrolle”. *Computing*, 4:93-106, 1969.

[FLM94] A. Frick, A. Ludwig and H. Mehldau. "A fast adaptive layout algorithm for undirected graphs". *Proceedings of Graph Drawing 94*, Springer Verlag, 1994.

[FR91] T. Fruchterman and E. Reingold. "Graph drawing by force-directed placement". *Softw. - Pract. Exp.*, 21(11):1129-1164, 1991.

[Fuk90] K. Fukunaga. *Introduction to Statistical Pattern Recognition. 2nd Edition*. Academic Press, New York, 1990.

[FVF+90] J. D. Foley, A. vanDam, S. K. Feiner and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.

[GA92] D. M. Gabbay, S. Abramsky, Eds. *Handbook of Logic in Computer Science*. Volume 1, Background: Mathematical Structures, Oxford Science Publications, 1992.

[Gar94] A. Garcia. *Numerical Methods for Physics*. Prentice-Hall, 1994.

[GCW97] S. Gelato, D. F. Chernoff and I. Wassermann “An Adaptive Hierarchical Particle-Mesh code with Isolated Boundary Conditions”. *Accepted for Astrophysical Journal*, 1997.

[GGR99] V. Ganti, J. Gehrke and R. Ramakrishnan “Mining Very Large Databases”. *IEEE Computer*, Volume: 32(8):68-75, 1999.

[GI94] M. R. Girardi, B. Ibrahim. "A Similarity Measure for Retrieving Software Artifacts". *Sixth International Conference on Software Engineering and Knowledge Engineering*, Jurmala, Latvia, pp 478-485, 1994.

[GK95] M. Gross, R. Koch “Visualization of Multidimensional Shape and Texture Features in Laser Range Data using Complex-Valued Gabor Wavelets”. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):44-59, 1995.

[Gla95] A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers,San Francisco, 1995.

[Gra96] N-Body/Particle Simulation Methods, *http://www.maths.napier.ac.uk:80/gavin/nbody.html*, 1996.

[Gre90] L. Greengard “The Numerical Solution of the N-Body Problem”. *Computers in Physics*, 5:142-152, 1990.

[GRG+99] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell and J. French. "Clustering Large Datasets in Arbitary Matric Spaces". *15th International Conference for Data Engineering*, Los Alamito, CA, 502-511, IEEE CS Press, 1999.

[Gro94] M. H. Gross. *Visual Computing - The Integration of Computer Graphics, Visual Perception and Imaging*. Springer-Verlag, 1994.

[GRS98] S. Guha, R. Rastogi and K. Shim. "CURE: An Efficient Clustering Algorithm for Large Databases". *Int'l Conference for Management of Data*, New York, 73-84, ACM Press, 1998.

[GRS98] S. Guha, R. Rastogi and K. Shim. "CURE: An efficient clustering algorithm for large databases". *Proceedings of ACM SIGMOD International Conference on Management of Data*, New York, pp. 73-84, 1998.

[GS93] M. H. Gross, F. Seibert “Visualization of Multidimensional Data Sets using a Neural Network”, 145-159, 1993.

[GSF97] M. H. Gross, T. C. Sprenger and J. Finger. "Visualizing Information on a Sphere". *Proceedings of the IEEE Information Visualization 97*, Phoenix AZ, USA, pp. 11-16, 1997.

[GT88] H. Gould, J. Tobochnik. *An Introduction to Computer Simulation Methods*. Addison Wesley, 1988.

[GT98] M. T. Goodrich, R. Tamassia. *Data Structures and Algorithms in JAVA*. John Wiley & Sons, Inc., 1998.

[Har72] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1972.

[HD95] Visualisation of complex systems, *http://www.cs.bham.ac.uk/~rjh/aig.html*, R. J. Hendley, N. S. Drew, 1995.

[HDH+00] M. Hao, U. Dayal, M. Hsu, T. C. Sprenger and M. H. Gross "Visualization of Directed Associations in E-Commerce Transaction Data". HP Laboratories Technical Report, 2000.

[HDH+01] M. C. Hao, U. Dayal, M. Hsu, T. Sprenger and M. H. Gross. "Visualization of directed associations in e-commerce transaction data". *Proceedings of VisSym'01*, Ascona, Switzerland, 2001.

[HE88] R. W. Hockney, J. W. Eastwood. *Computer Simulation Using Particles*. Adam Hilger, New York, 1988.

[Hel+99] J. M. Hellerstein, e. al. “Interactive Data Analysis: The Control Project”. *IEEE Computer*, 32(8):51-59, 1999.

[Hen+95] R. Hendley, e. al. "Case Study - Narcissus: Visualizing Information". *Proceedings of the IEEE Information Visualization 95*, pp. 90-96, 1995.

[HH91] T. R. Henry, S. E. Hudson. "Interactive Graph Layout". *Proceedings of the ACM SIGGRAPH Symposium on UI Software*, 1991.

[HH98] B. Heckel, B. Hamann. *Visualization of cluster hierarchies*. Visual Data Exploration and Analysis V, SPIE. R. F. a. P. Erbacher, A., eds. SPIE -- The International Society for Optical Engineering, Bellingham, WashingtonVol. 3298: pp. 162-171, 1998.

[HHD+01] M. C. Hao, M. Hsu, U. Dayal, S. F. Wei, T. C. Sprenger and T. Holenstein "Market Basket Analysis Visualization On A Spherical Surface". Hewlett Packard Research Laboratories, HPL-2001-3, 2001.

[HLN99] J. Han, L. V. S. Lakshmanan and R. T. Ng “Constraint-Based Multidimensional Data Mining”. *IEEE Computer*, 32(8):46-50, 1999.

[HMM00] I. Herman, G. Melancon and M. S. Marshall “Graph Visualization and Navigation in Information Visuzalization: A Survey”. *IEEE Transactions in Visualization and Computer Graphics*, vol. 6(no. 1), 2000.

[HNW91] E. Hairer, S. P. Nørsett and G. Wanner. *Solving Ordinary Differential Equations I - Nonstiff Problems*. Springer Verlag, 1991.

[HO92] L. Hernquist, J. Ostriker “A Self-Consistent Field Method for Galactic Dynamics”. *Astrophysical Journal*, 386:375-397, 1992.

[Hop96] H. Hoppe. "Progressive meshes". *Proceedings SIGGRAPH 96*, pp. 99-108, 1996.

[HW96] E. Hairer, G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer Verlag, 1996.

[HWB95] M. Harada, A. Witkin and D. Baraff. "Interactive Physically-Based Manipulation of Discrete/Continuous Models". *Proceedings SIGGRAPH 95*, pp. 199-208, 1995.

[IBM91] "Data Explorer Reference Manual, IBM Corporation". Armonk, NY, 1991.

[IBM97] Home of the latest technologies from IBM, *http://www.alphaWorks.ibm.com*, IBM Corporation, 1997.

[ID90] A. Inselberg, B. Dimsdale. "Parallel Coordinates: A Tool for Visuallizing Multi-Dimensional Geometry". *Proceedings of IEEE Visualization 90*, Los Alamitos, CA, pp. 361-375, 1990.

[J3D97] Java3D API Homepage, *http://java.sun.com/products/java-media/3D*, SUN Microsystems Inc., Palo Alto, CA, 1997.

[JavaBeans96] JavaBeans Homepage, *http://java.sun.com/products/javabeans*, SUN Microsystems Inc., Palo Alto, CA, 1996.

[JD88] A. K. Jain, R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[JP86] J. G. Jernigan, D. H. Porter “A tree code with logarithmic reduction of force terms, hierarchical regularization of all variables, and explicit accuracy controls”. *Astrophysical Journal, Supp. Series*, 71:pp. 871-893, 1989.

[Kam88] T. Kamada "On Visualization of Abstract Objects and Relations". Department of Information Science. University of Tokyo, 1988.

[Kam89a] T. Kamada "Symmetric Graph Drawing by a Spring Algorithm and its Applications to Radial Drawing". Department of Information Science, University of Tokyo, 1988.

[Kam89b] T. Kamada. *Visualizing Abstract Objects and Relations*, 1989.

[Kei97a] D. A. Keim. "Visual Techniques for Exploring Databases". *Invited Tutorial, Int. Conference on Knowledge Discovery in Databases (KDD'97)*, Newport Beach, CA, 1997.

[Kei97b] D. A. Keim. "Visual Data Mining". *Tutorial, Int. Conf. on Very Large Databases (VLDB'97)*, Athen, Greece, 1997.

[KHK99] G. Karypis, E. S. Han and V. Kumar “Chameleon: Hierarchical Clustering Using Dynamic Modeling”. *IEEE Computer*, Volume: 32(8):68-75, 1999.

[KK89] T. Kamada, S. Kawai. *An Algorithm for Drawing General Unidirected Graphs*. Inform. Process. 31:7-15, 1989.

[Knu63] D. E. Knuth. “Computer drawn flowcharts”. *Communications of the ACM*, 6, 1963.

[Koh95] T. Kohonen. *Self-Organizing Maps Second Extended Edition*. Springer, Berlin, Heidelberg, New York, 1995.

[LG95] J. P. Lee, G. G. Greenstein. "An architecture for retaining and analyzing visual explorations of databases". *Proceedings of IEEE Visualization 95*, pp. 101-108, IEEE Press, 195.

[LL00] L. J. Latecki, R. Lakämper “Shape Similarity Measure Based on Correspondence of Visual Parts”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10), 2000.

[LVC95] S. Li, O. d. Vel and D. Coomans "Comparative performance analysis of non-linear dimensionality reduction methods". James Cook University, 1995.

[Mac86] J. Mackinlay “Automating the design of graphical presentation of relational information”. *ACM Transactions on Graphics*, 5(2):110-141, 1986.

[Mag97] The Magician OpenGL Interface, *http://www.hermetica.com/technologia/java/magician*, Arcane Technologies Ltd., 1997.

[MDB87] B. H. McCormick, T. A. DeFanti and M. D. B. Ed. “Visualization in Scientific Computing (ViSC)”. *In Computer Graphics, ACM SIGGRAPH*, Vol. 21(Number 6), 1987.

[MHC+96]T. Munzner, E. Hoffmann, K. Claffy and B. Fenner. "Visualizing the Global Topology of the mbone". *Proceedings Symposium on Information Visualization 96*, 1996.

[Mil56] G. A. Miller “The magical number seven, plus or minus two: Some limits on our capacity for processing information”. *Psychological Review*, 63:81-97, 1956.

[MMG+98]J. E. Moreira, S. P. Midkiff, M. Gupta and R. D. Lawrence "Parallel Data Mining in Java". IBM T. J. Watson Research Center. Yorktown Heights, NY 10598-0218:18, 1998.

[MRC91] J. D. Mackinlay, G. G. Robertson and S. K. Card. "The Perspective Wall: Detail and Context Smoothly Integrated". *CHI '91*, 173-179, 1991.

[MS97] Dimension X Liquid Reality, *http://www.microsoft.com/dimensionx/lr*, Microsoft Corporation, Redmond, WA, 1997.

[MTS91] T. Mihalisin, J. Timlin and J. Schwegler “Visualizing Multivariate Functions, Data, and Distributions”. *IEEE Computer Graphics and Applications*, 11 (13):28-35, 1991.

[Mun97] T. Munzner. "H3: Laying out large directed graphs in 3D hyperbolic space". *Proceedings of IEEE Symposium on Information Visualization*, pp. 2-10, 1997.

[Mur91] S. Muraki. "Volumetric shape description of range data using “Blobby Model”". *Proceedings SIGGRAPH 91*, pp. 227–235, 1991.

[NH91] J. Nievergelt, K. Hinrichs. *Algorithms and Data Structures with Applications to Graphics and Geometry*. Englewood Cliffs: Prentice Hall, 1993.

[NH94] R. T. Ng, J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining". *20th Int'l Conference for Very Large Data Bases*, San Francisco, 144-155, Morgan Kaufmann, 1994.

[PCJ96] H. C. Purchase, R. F. Cohen and M. James. *Validating Graph Drawing Aestetics*. Graph Drawing (Proc. GD 95). e. F. J. Brandenburg. Springer-Verlagvol. 1027 of Lecture Notes Comput. Sci.: pp. 435-336, 1996.

[PFT+88] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1988.

[Pre86] W. H. Press. *The Use of Supercomputers in Stellar Dynamics*. Springer Verlag, New York, 1986.

[PSS97] J. D. Pinter, W. J. H. Stortelder and J. J. B. d. Swart "Computation of Elliptic Fekete Point Sets". CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands, 1997.

[RC94] R. Rao, S. K. Card. "The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information". *CHI '94*, 318-322, 1994.

[RCM89] G. G. Robertson, S. K. Card and J. D. Mackinlay. "The Cognitive Co-processor for Interactive User Interfaces". *Proceedings of UIST 89, ACM Symposium on User Interface Software and Technology*, pp. 10-18, 1989.

[RG87] V. Rokhlin, L. Greengard “A Fast Algorithm for Particle Simulations”. *Journal of Computational Physics*, v. 73:pp. 325-348, 1987.

[RG99] N. Ramakrishnan, A. Y. Grama “Data Mining: From Serendipity to Science”. *IEEE Computer*, 32(8):34-37, 1999.

[RMC91] G. G. Robertson, J. D. Mackinlay and S. K. Card. "Cone Trees: Animated 3D Visualization of Hierarchical Information". *Proceedings Human Factors in Computing Systems CHI '91 Conference*, New Orleans, LA, pp. 189-194, 1991.

[Rok85] V. Rokhlin "Rapid Solution of Integral Equations of Classical Potential Theory". *J. Comp. Phys.*, v. 60, 1985.

[RSP+93] D. M. Russell, M. J. Stefik, P. Pirolli and S. K. Card. "The Cost Structure of Sensemaking". *Proceedings of INTERCHI '93*, Amsterdam, pp. 269-276, ACM, New York, 1993.

[RT81] E. M. Reingold, J. S. Tilford "Tidier Drawing of Trees". *IEEE Transactions on Software Eingineering*, 7(2):pp. 223-228, 1981.

[SBG00] T. C. Sprenger, R. Brunella and M. H. Gross. "H-BLOB: A Hierarchical Visual Clustering Method Using Implicit Surfaces". *Proceedings of IEEE Visualization 00*, Salt Lake City, IEEE Press, 2000.

[Sch93] H. R. Schwarz. *Numerische Mathematik*. B. G. Teubner Stuttgart, 1993.

[SCZ98] G. Sheikholeslami, S. Chatterjee and A. Zhang. "WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases". *24th Int'l Conference for Very Large Data Bases*, San Francisco, 428-439, Morgan Kaufmann, 1998.

[SGB98] T. C. Sprenger, M. H. Gross, D. Bielser and T. Strasser. "IVORY - An Object-Oriented Framework for Physics-Based Information Visualization in Java". *Proceedings of IEEE Information Visualization 98*, Research Triangle Park, NC, USA, pp. 79-86, 1998.

[SGE+97] T. C. Sprenger, M. H. Gross, A. Eggenberger and M. Kaufmann. "A Framework for Physically-Based Information Visualization". *Eight EuroGraphics-Workshop on Visualization in Scientific Computing*, France, pp. 77-86, 1997.

[SGI91] "IRIS Explorer Users Guide, Silicon Graphics Inc.". Mountain View, CA, 1991.

[SGI97] MineSet: Silicon Graphics data mining and visualization product, *http://www.sgi.com/Products/software/MineSet*, Silicon Graphics Inc., Mountain View, CA, 1997.

[SM86] R. Stepp, R. Michalski "Conceptual clustering of structured objects: A goal-oriented approach". *Artificial Intelligence*, (28):43-69, 1986.

[SML96] W. J. Schroeder, K. M. Martin and W. E. Lorensen. "The Design and Implementation Of An Object-Oriented Toolkit For 3D Graphics And Visualization". *Proceedings of IEEE Visualization 96*, pp. 93-100, 1996.

[SO95] M. Stricker, M. Orengo. "Similarity of Color Images". *Proceedings of Storage and Retrieval for Image and Video Databases, SPIE*, San Jose, CA, 1995.

[SR96] Splinter, Randall "A Nested Grid Particle-Mesh Code for High Resolution Simulations of Gravitational Instability in Cosmology". *MNRAS*, 281, 1996.

[STT81] K. Sugiyama, S. Tagawa and M. Toda "Methods for Visual Understanding of Hierarchical Systems". *IEEE Transactions Syst. Man Cybern.*, SMC-11(2):109-125, 1981.

[SUN97] The source for Java Technology, *http://java.sun.com*, SUN Microsystems Inc., Palo Alto, CA, 1997.

[TLA90] A. M. Tannenbaum, Y. Langsam and M. J. Augenstein. *Data Structures Using C*. Prentice-Hall, 1990.

[Tuf83] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1983.

[VD98] Homepage of Visual Decision, *http://www.visualdecision.com/*, Visual Decision, Québec, Canada, 1998.

[VI97] Homepage of Visual Insights, *http://www.visualinsight.com/*, Visual Insights, Naperville, IL 60563-8495, 1997.

[VRM97] ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97), *http://www.vrml.org/Specifications/VRML97*, VRML Consortium Inc., 1997.

[W3C97] Web Naming and Addressing Overview: Uniform Resource Locator (URL), *http://www.w3.org/Addressing/URL*, W3C Consortium, 1997.

[WB95] P. C. Wong, R. D. Bergeron. "30 Years of Multidimensional Multivariate Visualization". *Proc. Workshop on Scientific Visualization*, IEEE Computer Society Press, 1995.

[WBK95] A. Witkin, D. Baraff and M. Kass. "Physically-Based Modeling". *SIGGRAPH 95 Tutorial course Notes No. 34*, 1995.

[Wis+95] J. Wise, e. al. "Visualizing the Non-Visual: Spatial analysis and Interaction with Information from text Documents". *Proceedings of the IEEE Information Visualization 95*, pp. 51-58, 1995.

[Wit96] A. Witkin. "Particle System Dynamics". *SIGGRAPH 96 Course Notes 34*, pp. C1-C12, 1996.

[Woo95] A. Wood, e. al. "HyperSpace: Web Browsing with Visualisation". *Third International World-Wide Web Conference, Poster Proceedings*, Darmstadt, Germany, pp. 21-25, 1995.

[WPL01] Chariot - Similarity Search in Large Image Databases, *http://mercator.inf.ethz.ch/Chariot/*, 2001.

[Xu96] A New Parallel N-body Gravity Solver: TPM, *http://zeus.ncsa.uiuc.edu:8080/gc3/gc306/abstract.html*, 1996.

[You87] F. Young. *Multidimensional scaling: history, theory, and applications*. Lawrence Erlbaum associates, Hillsdale, New Jersey, 1987.

[You96] P. Young "Three Dimensional Information Visualization". *Visualisation Research Group, Center for Software Maintenance, Department of Computer Science, University of Durham. Durham*, 1996.

[YR91] F. Young, P. Rheingans "Visualizing Strucutre in High-Dimensional Multivariate Data". *IBM Journal of Research and Development*, 35(1/2):97-107, 1991.

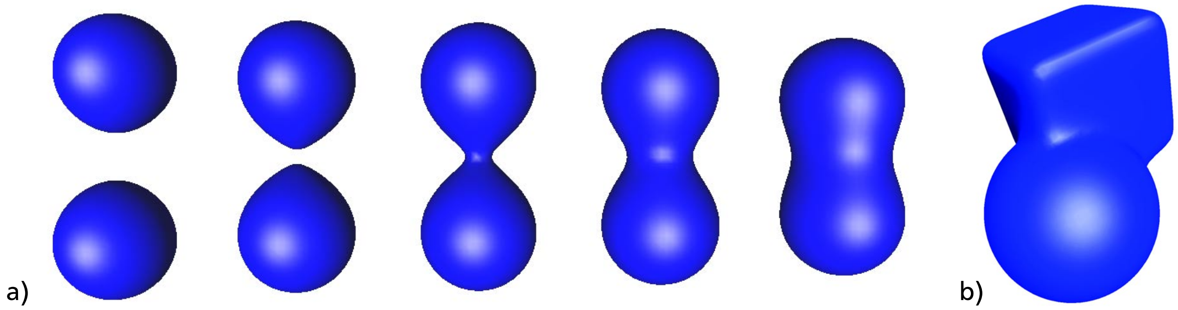
[Zon64] J. A. Zonneveld. "Automatic numerical integration". *Math. Centra Tracts*, Mathematisch Centrum Amsterdam, 1964.

[ZRL96] T. Zhang, R. Ramakrishnan and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, pp. 103-114, ACM Press, 1996.

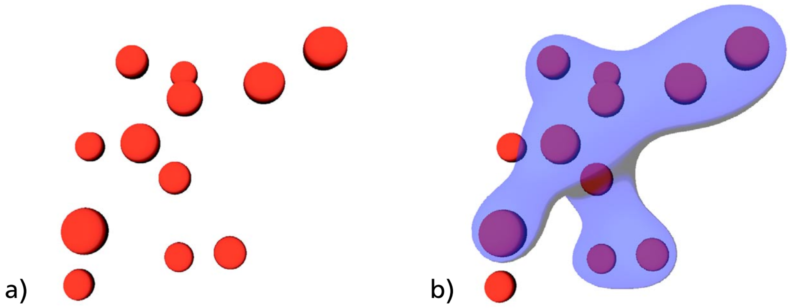
[Zup82] J. Zupan. *Clustering of Large Data Sets*. Research Studies Press, 1982.

E

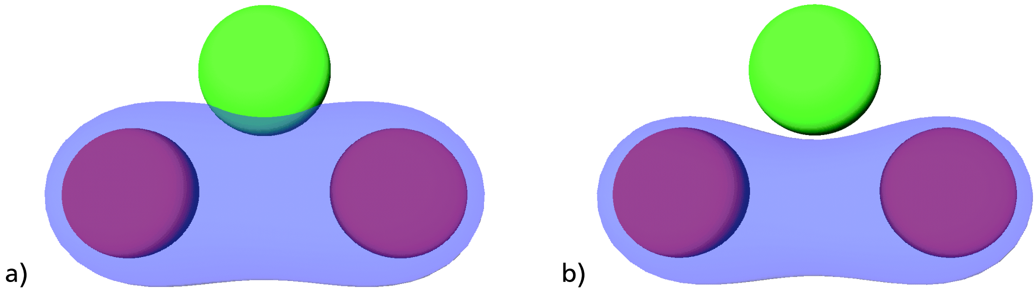
COLOR PLATES



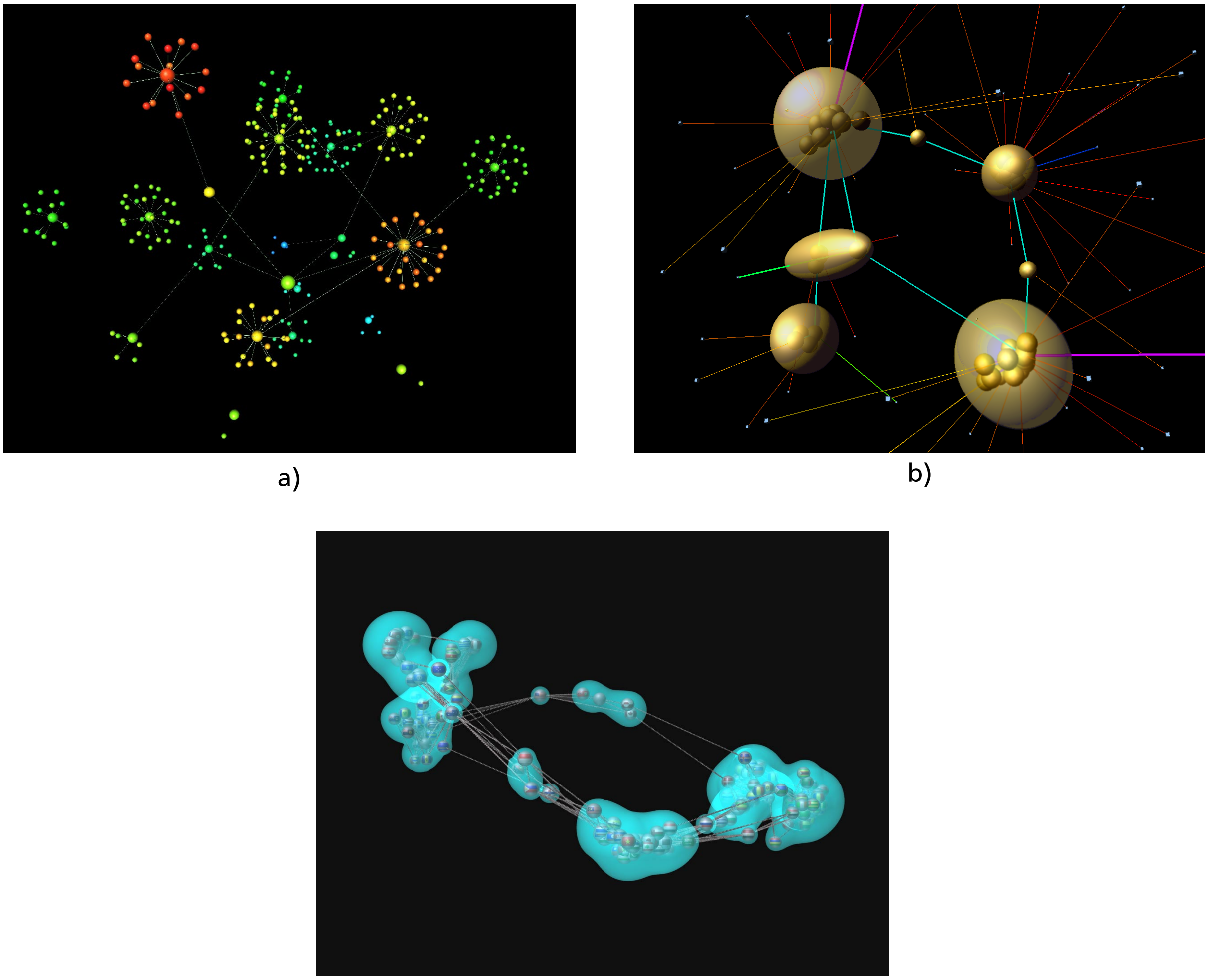
Color Plate 1 a) Illustration of implicit surfaces obtained from two field functions as their centroids are approaching. b) Modification of the shape by setting $\nu = \mu = 0.25$ for the upper field function (see Figure 7.8 on page 106).



Color Plate 2 Clustering of a subset of objects scattered in 3D space: a) initial configuration, b) clustering isosurface as a transparent hull wrapped around the desired objects (see Figure 7.10 on page 107).



Color Plate 3 a) Undesired intersection of object and hull. b) repulse of the clustering surface from the object by assigning a negative value for b (see Figure 7.11 on page 108).

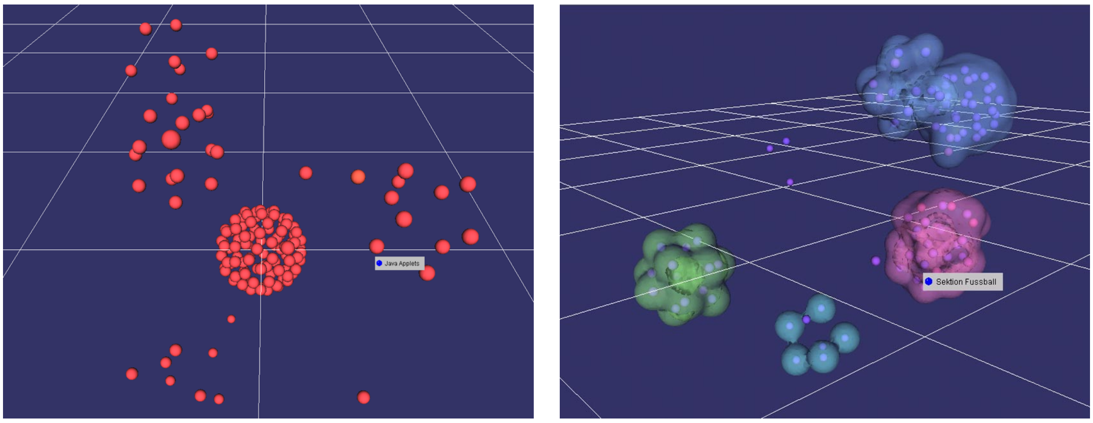


a)

b)

c)

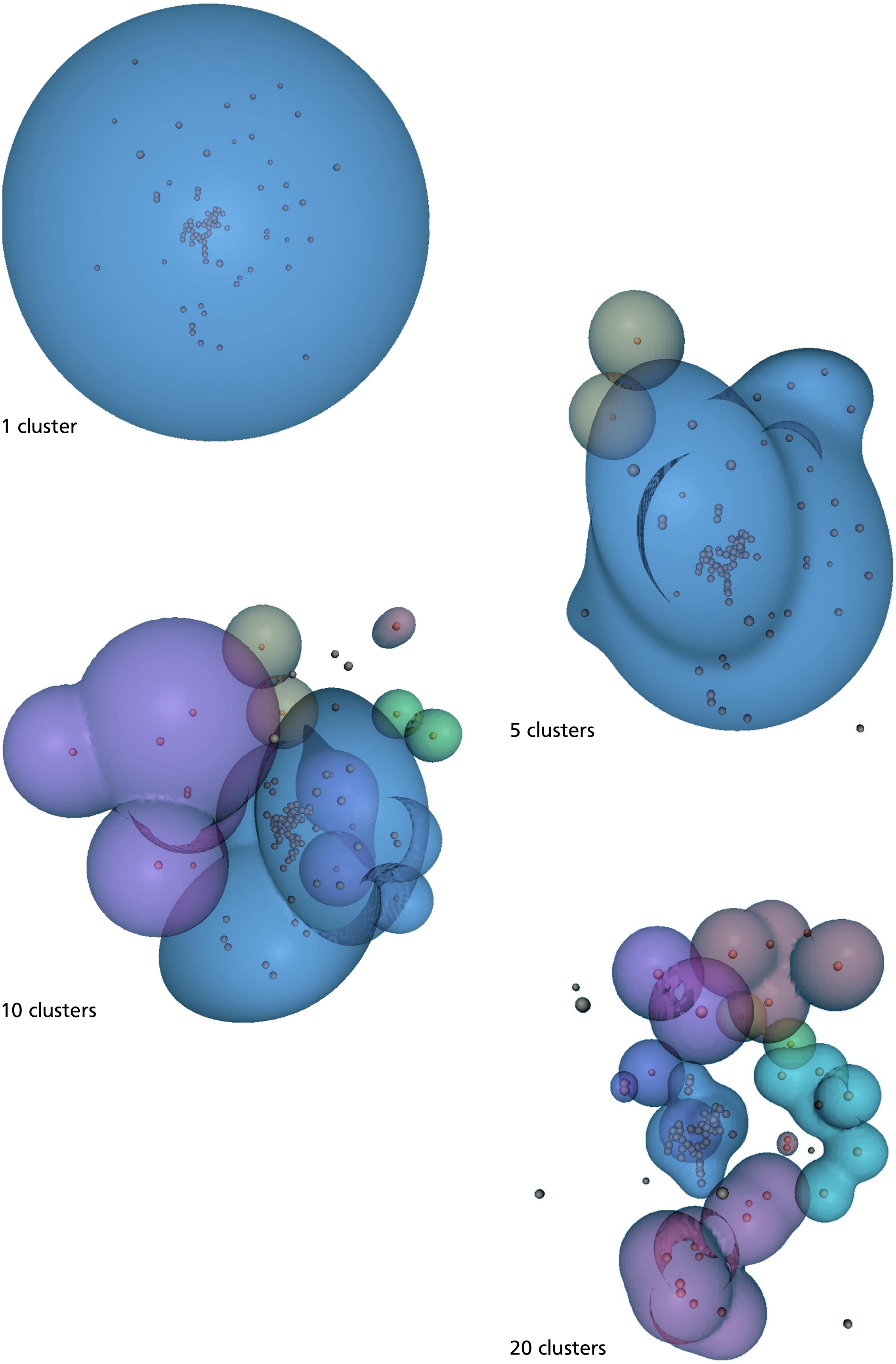
Color Plate 4 Different techniques to visualize clusters of data objects. a) cluster represented by a cluttered group of single objects b) visualization with ellipsoidal surfaces wrapped around clusters c) objects visually combined by a BLOB surface (see Figure 7.6 on page 101).



a)

b)

Color Plate 5 Two typical IVORY scenes from this test showing clustered Intranet documents arranged according to subject (see Appendix 11). a) Shows the result of scenario 2, no visual clustering, 150 documents b) Presents a snapshot of scenario 10, BLOB clustering, 100 documents (see Figure 9.3 on page 141).



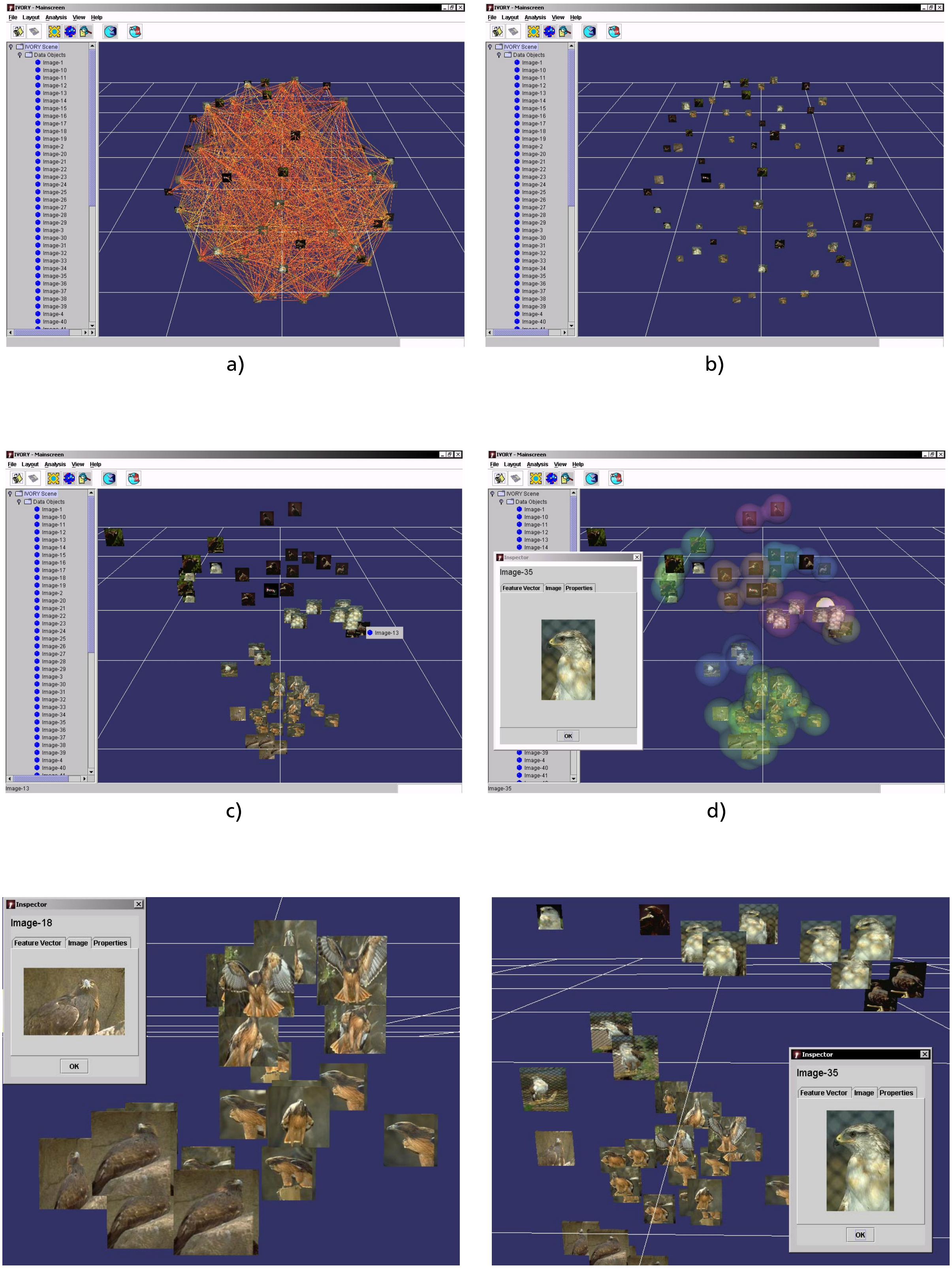
1 cluster

5 clusters

10 clusters

20 clusters

Color Plate 6 Document Retrieval Visualization using H-BLOB clustering. Cluster hierarchies are shown with 20, 10, 5 and 1 cluster (see Figure 9.9 on page 146).



a)

b)

c)

d)

e)

f)

Color Plate 7 Screen shots from the image retrieval example: a, b) Initial arrangement on a virtual spherical surface c-f) Different views of resulting object layout automatically arranged by the IVORY framework (see Figure 9.11 on page 148).

F

COPYRIGHTS

[HP©]	VisMine Research Project © 1999 Hewlett Packard Research Laboratories, Palo Alto, USA.

G

CURRICULUM

Name: Sprenger
Firstname: Thomas Carl
Date of birth: 06/05/1970
Place of birth: Basel, Switzerland
Marital status: single
Address: 707, Bounty Dr. #201W
Foster City, CA 94404
USA
Phone private +1-650-345 7402
work +1-650-252 9322 ext. 12
E-Mail tom@sprengers.ch

G.1 SCHOLAR EDUCATION

April 1977 – April 1983	Primary school, Rorschach (SG), Switzerland
April 1983 – April 1985	Secondary school, Rorschach (SG), Switzerland
April 1985 – January 1990	Highschool (Typus C), Heerbrugg (SG), Switzerland
October 1990 – February 1996	Swiss Federal Institute of Technology (ETH) Zürich Studies in Computer Science (major) and Biomedical Engineering Degrees: - 1992 Bachelor of Computer Science - 1996 Master of Computer Science (Dipl. Informatik Ing. ETH)
April 1996 – October 2000	Ph.D. on the topic of Information Visualization and Computer Graphics at ETH Zürich

G.2 PRACTICAL EXPERIENCE

February	1993 –	May	1993	GRETAG Imaging, Regensdorf, Switzerland, Practical training, Computer Vision
August	1995 –	October	1995	Swissbank IT-Camp, Basel, Switzerland, Practical training, Virtual Bank Project
July	1999 –	October	1999	HP Laboratories, Palo Alto, USA, Seed exchange, Information Visualization
April	1996 –	October	2000	ETH Zürich, Department of Scientific Computing, Computer Graphics, Switzerland, Research Assistant, Object-oriented information visualization framework design

G.3 KNOWLEDGE/ABILITIES/INTERESTS

Languages	German:	native language
	English:	fluently speaking and writing
	French:	speaking and writing
Hobbies	Volleyball, Beachvolleyball, Skwaling, Snowboarding, Graphical Design, Attending concerts and movie presentations	