Effective Retrieval and Visual Analysis in Multimedia Databases

Dissertation zur Erlangung des akademischen Grades des Doktors der Naturwissenschaften an der Universität Konstanz im Fachbereich Informatik und Informationswissenschaft

vorgelegt von

Tobias Schreck

Universität Konstanz

April 2007

Tag der mündlichen Prüfung: 15. Dezember 2006

Referenten: Prof. Dr. Daniel A. Keim, Universität Konstanz

Prof. Dr. Dietmar Saupe, Universität Konstanz

Priv.-Doz. Dr. Umeshwar Dayal, HP Labs, Palo Alto, USA

Für meine Eltern.

Acknowledgments

The last four years during which I worked on this thesis mark a challenging and demanding, yet rewarding period for me. Not only was I introduced to a variety of intriguing research problems in the fields of content-based retrieval and visual analytics. In particular, I had the opportunity to work with a number of inspiring people, each of them sharing insight and motivation.

First of all I like to thank Daniel Keim for his continued supervision and motivation. His approach to defining interesting problems, and then tackling them by appropriate methods has shaped much of my work. I thank Daniel for the wealth of insight provided, and his encouragement to teamwork with colleagues, and to share results with the community. Then, I like to thank Dietmar Saupe for providing supervision and inspiration during continued collaboration. Dietmar shared valuable insight on many occasions; his ready support and help when writing papers is highly appreciated. Some of the ideas investigated in this thesis were developed during two research internships I had the opportunity to spend with Hewlett-Packard Laboratories at Palo Alto, California. I like to thank Umesh Dayal and Ming Hao of HP Labs for their continued support and collaboration, and for a rewarding internship time.

Many thanks go to dear collaborators at the University of Konstanz. Benjamin Bustos is a great colleague always willing to help and to discuss problems and ideas. Christian Panse, Jörn Schneidewind, and Mike Sips have become great colleagues and friend along the way, sharing with me their passion and expertise in visualization and data analysis. I also thank Dejan Vranić for nice collaboration and sharing expertise and data during our joint 3D retrieval project. I thank Marco Pötke of SD&M AG München for valuable discussion on aspects of geometric similarity search and visualization.

I also like to cordially thank my group colleagues Florian Mansmann, Hartmut Ziegler, and Markus Wawryniuk. They all provided specific insights, support, and valuable discussion during joint projects. I also thank students Tilo Nietzschmann, Domink Morent, and Henrico Dolfing for fruitful joint work.

This work was supported by the German Research Foundation (DFG), by the University of Konstanz, by Hewlett-Packard Laboratories, and by the EU Panda and Delos Networks of Excellence.

Parts of this thesis were published in:

1. T. Schreck and C. Panse. A new metaphor for projection-based visual analysis and data exploration. Proceedings of the *IS&T/SPIE Conference on Visualization and Data Analysis (VDA)*, 2007.
2. B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranic. An experimental effectiveness comparison of methods for 3D similarity search. *International Journal on Digital Libraries, Special Issue on Multimedia Contents and Management (IJDL),* 6(1):39-54, Springer, 2006.
3. T. Schreck, D. Keim, and C. Panse. Visual feature space analysis for unsupervised effectiveness estimation and feature engineering. *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, IEEE, 2006.
4. D. Keim, T. Nietzschmann, N. Schelwies, J. Schneidewind, T. Schreck, and H. Ziegler. A spectral visualization system for analyzing financial time series data. *Proceedings of the Eurographics/IEEE-VGTC Symposium on Visualization (EuroVis)*, IEEE, 2006.
5. T. Schreck, D. Keim, and F. Mansmann. Regular Treemap layouts for visual analysis of hierarchical data. *Proceedings of the Spring Conference on Computer Graphics (SCCG),* Comenius University Bratislava, 2006.
6. B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranic. Feature-based similarity search in 3D object databases. *ACM Computing Surveys (CSUR)*, 37:345-387, ACM Press, 2005.
7. U. Dayal, M. Hao, D. Keim, and T. Schreck. Importance driven visualization layouts for large time-series data. *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)*, IEEE, 2005.
8. D. Keim, F. Mansmann, and T. Schreck. MailSom - visual exploration of electronic mail archives using Self-Organizing Maps. *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, Stanford University, 2005 (short paper).
9. B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranic. Using entropy impurity for improved 3D object similarity search. *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME),* IEEE, 2004.
10. B. Bustos, D. Keim, C. Panse, and T. Schreck. 2D maps for visual analysis and retrieval in large multi-feature 3D model databases. *Proceedings of the IEEE Visualization Conference (VIS)*, IEEE, 2004 (poster paper).

Abstract

Based on advances in acquisition, storage, and dissemination technology, increasing amounts of multimedia content such as images, audio, video, or 3D models, become available. The *Feature Vector* (FV) paradigm is one of the most popular approaches for managing multimedia content due to its simplicity and generality. It maps multimedia elements from object space to metric space, allowing to infer object similarity relationships from distances in metric space. The distances in turn are used to implement similarity-based multimedia applications. For a given multimedia data type, many different FV mappings are possible, and the *effectiveness* of a FV mapping can be understood as the degree of resemblance of object space similarity relationships by distances in metric space. The effectiveness of the FV mapping is essential for any application based on it.

Two main ideas motivate this thesis. We first recognize that the FV approach is promising, but needs attention of FV selection and engineering in order to serve as a basis for building effective multimedia applications. Secondly, we believe that visualization can contribute to building powerful user interfaces for analysis of the FV as well as the object space. This thesis focuses on supporting a number of important user tasks in FV-based multimedia databases. Specifically, we propose innovative methods for (a) effective processing of content-based similarity queries, (b) FV space visualization for discrimination analysis, and (c) visualization layout generation for content presentation. The methods are applied and evaluated on a number of specific multimedia data types such as 3D models, images, and time series data, and are expected to be useful in many other multimedia domains.

Effective retrieval in 3D databases (Chapter 2). We review and classify a significant number of recently proposed FV extractors supporting the 3D model domain. Extensive effectiveness evaluation experiments are performed for many FV extractors on a number of benchmarks. Methods for improving retrieval effectiveness by forming static and query-dependent combinations of FVs are researched. Experiments show significant improvements in retrieval precision (quality of the answer sets) to be achievable.

Visual FV space analysis (Chapter 3). We explore the usage of interactive 2D projections for retrieval and organization of multimedia content in a multi-FV 3D retrieval system. Self-organizing maps (SOMs) have shown to be appropriate to this end. We propose a PCA-based visualization method for supervised visual discrimination analysis in FV space. Also, SOM-based techniques are explored for unsupervised estimation of FV space discrimination power. Both visualizations can be used for addressing the FV selection problem, and for fine tuning FV-based multimedia applications.

Layout generation for content presentation (Chapter 4). We investigate visualization layout generation strategies for presentation of multimedia content. Inspired by the popular TreeMap algorithm, we develop space-efficient layout generators providing certain regularity and ordering properties which are useful for presenting multimedia content to the user. The layouts are applied on sets of time series data, and experimentally shown to outperform competing TreeMap algorithms on a number of metrics.

Zusammenfassung

Basierend auf Fortschritten bei der digitalen Erfassung, Speicherung und Übermittlung multimedialer Inhalte werden zunehmend grosse Mengen von Multimedia Objekten wie z.B. Bilder, Audio, Videos, und 3D Modellen verfügbar. Das *Feature Vector* Paradigma ist aufgrund seiner Einfachheit und Allgemeinheit einer der populärsten Ansätze zum Management von Multimedia Inhalten. Es bildet die Elemente eines Multimedia Objektraumes in einen metrischen Raum ab und ermöglicht hierdurch, von den Distanzen im metrischen Raum auf Ähnlichkeitsbeziehungen im Objektraum rückschliessen zu können. Für einen gegebenen Multimedia Datentyp sind prinzipiell viele verschiedene Abbildungen in einen metrischen Raum denkbar. Die *Effektivität* einer gegebenen Abbildung kann als der Grad der Übereinstimmung der Distanzen im metrischen Raum mit dem Grad der Ähnlichkeiten im Objektraum verstanden werden. Die Effektivität der Abbildung mit Feature Vektoren ist von grundlegender Bedeutung für alle auf dieser Abbildung aufsetzenden Anwendungen.

Zwei grundlegende Ideen liegen dieser Arbeit zugrunde. Zum einen stellen wir fest, dass der Feature Vektor Ansatz der geeigneten Auswahl und Konfiguration der Feature Vektoren bedarf, um effektive Anwendungen zu ermöglichen. Zum anderen sind wir überzeugt davon, dass bestimmte Visualisierungstechniken als effektive Schnittstellen für den Feature und den Objektraum geeignet sind. Im Rahmen dieser Arbeit werden innovative Methoden (a) zur effektiven Ausführung von Ähnlichkeits-Suchanfragen, (b) zur visuellen Diskriminierunganalyse, und (c) zur Layouterzeugung für die Präsentation und Analyse von Multimedia Daten entwickelt. Die Nützlichkeit der Methoden wird durch Anwendung auf eine Reihe von verschiedenen Multimedia Datentypen wie 3D Objekte und Zeitreihendaten aufgezeigt.

Effektive Ähnlichkeitssuche in 3D Datenbanken (Kapitel 2). Wir klassifizieren eine signifikante Anzahl von aktuellen Feature Vektor Extraktionsverfahren zur inhaltsbasierten Beschreibung von 3D Modellen. Umfassende Effektivitätsevaluierungen werden durchgeführt, basierend auf verschiedenen Benchmarks. Darüber hinaus werden Methoden zur Verbesserung der Retrieval Effektivität durch Bildung von statischen und dynamischen Kombinationen aus Feature Vektoren vorgestellt.

Visuelle Featureraum Analyse (Kapitel 3). Es werden 2D Projektionen entwickelt die zum Retrieval und zur Organisation von Multimedia Datenbanken dienen. Selbstorganisierende Karten (SOMs) haben sich in diesem Zusammenhang als nützlich erwiesen. Wir untersuchen weiterhin Ansätze zur überwachten und unüberwachten Diskriminierungsanalyse mittels 2D Projektionstechniken. Die Ergebnisse dieser Analyse unterstützen den Datenbankadministrator beim Feature Selektionsproblem und beim Feature Feintuning sowie den Endbenutzer beim Retrieval und der Egebnisanalyse.

Layout Erzeugung für Multimedia Inhalte (Kapitel 4). Basierend auf dem populären TreeMap Algorithmus entwickeln wir neue Strategien zur Darstellung von multimedialen Inhalten. Es werden raumeffiziente Layout Generatoren entworfen, welche bestimmte Ordnungs- und Regularitätseigenschaften aufweisen und deshalb nützlich für die Darstellung von Multimedia Inhalten sind. Untersuchungen zeigen die Überlegenheit der Algorithmen gegenüber klassischen TreeMap Varianten bzgl. einer Reihe von Gütekriterien.

Contents

List of Figures

List of Tables

1 Introduction

Contents

Database technology provides the back end functionality for all kinds of information systems shaping the information society. Whether it be operational information systems supporting processes and work flows in an organization, or innovative applications in E-Commerce, E-Learning, E-Government and so on, all these rely on database technology for storing, querying, manipulating, and sharing information, while providing services such as high performance, high reliability, and concurrency control. The relational approach to data management was extremely successful in offering a simple, powerful model for structuring *factual* information. Yet, due to advances in data acquisition and processing technology, as well as increasing physical storage capacities, large amounts of *multimedia* data are created, collected, and stored.

The term multimedia in the narrow sense describes data compound of multiple media types, e.g., a video stream incorporating an animation sequence with an audio track synchronized to the animation. In a wider sense (which is adopted in this thesis) multimedia data is understood as all kinds of non-standard media information, not necessarily compounding different media at once. Example data types according to this notion include vector graphics, images, 3-dimensional models, audio signals, video sequences, but also trees and graphs of all kinds of applications (such as describing a molecular structure, for example). Multimedia database technology aims at providing useful management facilities for the multimedia data type, much like relational database technology does for the relational data type.

1.1 Database support for multimedia data

In the multimedia database domain, important concepts from relational database technology have to be adapted or show to be less important, while new concepts need to be introduced. E.g., the transaction concept does not play as important a role in managing a database of images than it does in managing a database of flight reservation records, simply because not many conflicting accesses to the database occur. On the other hand, the update operation on an image is not as straightforward as changing one attribute in a given record, but data-specific methods and interfaces for updating the content need to be implemented. One important concept that does not directly translate from relational to multimedia databases is that of *querying*. A relational query can be specified directly on the database schema, requesting the records where the respective attributes satisfy the given selection criteria. By definition, the query is an exact one: Either a record satisfies the query, or it does not. In multimedia databases there usually is no schema structuring the content on the semantic level which the user is typically interested in, and most of the time it is not very useful to perform exact queries on the low-level data representation itself. Rather than that, the user will query for certain data-specific, high-level characteristics of the multimedia data. Consider again an image database. The typical user will not be able to specify a query in the form of requiring certain RGB values in certain regions of the images. She will rather be interested in higher-level features such as e.g., presence or absence of a sunset in the image, or other semantic concepts recognized in the images.

Besides explicitly specifying semantic concepts to query for in a multimedia database, *query-by-example* is an important querying paradigm. Suppose the user has one or a few multimedia objects already at hand, e.g., by preceding browsing of the database. A query-by-example then requests the objects most *similar* to the already available object (the query object). To this end, the multimedia database system has to implement suitable similarity functions scoring the similarity between any two objects of a considered data type in order to produce answer lists of database objects sorted by decreasing degree of similarity for presentations to the user. The *Feature Vector* approach [33] due to its simplicity, performance, and generality is a popular scheme to implement such similarity functions in a multimedia database system. This thesis considers methods for the effective support of several important applications in multimedia databases where one or multiple Feature Vectors are available for each database object. The remainder of this introduction is structured as follows. Section 1.2 recalls the Feature Vector approach to similarity-oriented multimedia data management. Section1.3 describes several prominent database applications, and options for benchmarking FV extractors. In Section 1.4, we describe a number of data types which are of specific interest in this thesis. Finally, Section 1.5 outlines the remainder of this thesis, and summarizes its main contributions.

1.2 Feature Vector approach and effectiveness considerations

The Feature Vector (FV) approach [33] to similarity-based management of multimedia data represents multimedia objects $o \in O$ given in an original object space O by points $\vec{p}_o \in \mathbb{R}^d$ in a d-dimensional vector space. FV extractors are functions $O \rightarrow \mathbb{R}^d$ mapping elements from object space to vectors of real values, numerically describing characteristic objects properties. Suitable extractors should provide that the generated FVs (a) are efficiently extracted and (b) allow to effectively capture object space similarity relationships by appropriate distance functions $d : (\vec{p}_i, \vec{p}_j) \rightarrow \mathbb{R}_0^+$ defined in FV space. The FV approach provides a simple, flexible means to implement important multimedia applications such as content-based retrieval and clustering. Also, the FV approach supports database indexing [10], providing efficient access to database content. For many multimedia data types, description schemes other than FVs exit, e.g., relying on graph-based representations. Also, sophisticated transformation-based matching schemes have been proposed for certain content, where the similarity between two objects is given by the (appropriately defined) cost minimum over all possible transformations $\pi : o_i \rightarrow o_j$ transforming object o_i into object o_j. Yet, due to its simplicity and generality, the FV approach remains highly popular.

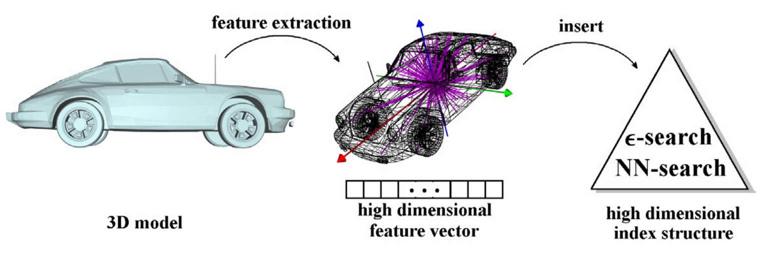
The *effectiveness* of the FVs used to represent multimedia content is of critical importance for any similarity-based application under the FV approach. We understand the effectiveness of a FV extractor as the degree of how accurately distances d in FV space resemble similarity relationships in object space. Generally, FVs are heuristically introduced, and their effectiveness is experimentally evaluated. For a given multimedia data type, many different approaches to extract characteristic features are possible, and a-priori it is not clear what the most important features are, given a data type and application. For many multimedia data types a wealth of competing FV extractors have already been proposed, and even in spite of significant experimental efforts in the multimedia community, often it is not clear what the most effective FV extractors would be for a given data type, far less for a given database. Evidence for the wealth of different FV extractors available are given e.g., in the image [93] and the 3D model (cf. Section 2.3 and [18]) domain, where each several dozens of competing schemes for mapping objects to FV space have been proposed to date.

The reason for such a wealth of FV extractors are (a) the heuristical nature by which the extractors are defined, and (b) the potentially very different similarity notions one can establish for many data types. Consider e.g., the image data type. For different types of images and databases, different types of features seem reasonable. If the user is interested in the distribution of colors present in the images, she could use for example histograms defined in RGB color space as features to describe the images. On the other hand, if color is irrelevant, but rather individual objects contained in the images are of interest, then certainly, features defined over segmented images identifying objects within the images are desired. As another example, consider the 3D model data type. Similarity can be measured based on global or local geometric shape, but it can also be measured based on structural properties of the models. On a wider scale, the similarity between two models can be based on the semantics of the objects in a given application context.

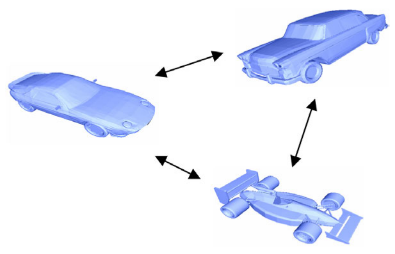
Figure 1.1 (a) illustrates the FV extraction process for a 3D multimedia object. Figure 1.1 (b) illustrates three different 3D models of cars. Depending on the given similarity notion, different degrees of similarity are perceivable (e.g., all models function as means of transportation, and their structure is given by a chassis with four wheels; two models belong to the sports car class, while one car is a limousine; etc.)

1.3 Applications and benchmarking in multimedia databases

Many multimedia applications rely on a representation of the original data in an effective FV space to produce meaningful results. In Content-based Database Retrieval, distances between a query object or feature-based query specification, and the database elements are evaluated


feature extraction
insert
3D model
high dimensional feature vector
ϵ-search
NN-search
high dimensional index structure

(a)



(b)

Figure 1.1: Under the Feature Vector approach to retrieval in multimedia databases, the similarity between objects is estimated by the distance between their respective Feature Vector representations (a). The definition of Feature Vectors for a given multimedia data type should reflect an appropriate similarity concept depending on the application, user context, etc. This is illustrated in (b), where the degree of similarity between individual models can be based on various structural or geometric properties.

to produce answer lists sorted by increasing distance to the query point [33]. Distances in FV space are also required in Classification and Clustering [37]. Briefly, in Classification unknown data instances are assigned the class label of the most similar class according to a classifier learned from supervised training data. In Clustering, distances between data instances are used to automatically find clusters of similar elements. Also, in Information Visualization [22] often similarity relationships among the data objects are exploited for effective image generation, e.g., by constructing similarity-driven layouts (cf. also Sections 3.1 and 4.3.3).

While such more user-oriented applications assume the data already to be described in an appropriate FV space, *benchmarking* as a more system-oriented application refers to the process of selecting promising FV spaces from a pool of available FV spaces, for a given database or application. We can distinguish two dimensions of benchmarking: *Supervised* vs. *unsupervised* benchmarking indicates whether supervised information, usually some kind of ground truth, is used in the benchmark, or whether it operates independently of such controlled information. Furthermore, we can distinguish between *numeric* and *visual* benchmarks. We next detail the latter options for benchmarking.

Numeric benchmarking. The effectiveness of a FV space can be numerically benchmarked if a suitable ground truth classification (supervised information) is available. In many data domains, reference benchmarks have been designed containing data and supervised classification information. Example benchmarks are the TREC document collection [68] for text retrieval, the COREL image collections in image retrieval [65], or the Princeton Shape Benchmark for 3D model retrieval [81]. Such retrieval benchmarks can be evaluated using metrics from Information Retrieval such as Precision and Recall [6]. For benchmarking the effectiveness of classification and clustering algorithms, e.g., the UCI Machine Learning Archive [25] provides data sets for machine learning problems from a wealth of application domains. Supervised numeric benchmarking can be problematic due to the costs associated with building and evaluating benchmarks, and also potential instability and ambiguities in the benchmark itself [65]. Therefore, unsupervised FV space benchmarking is desirable, but still a largely unsolved problem. Certain theoretical approaches relying on statistical information calculated in FV space exist [2, 42]. These works are of rather theoretical nature and to the best of our knowledge have not been practically leveraged yet.

Visual benchmarking. Visual benchmarking is an interesting option complementing the numeric benchmarking approach. It aims to support an evaluator in understanding and explaining numeric benchmarking results. It also aims to support the definition and tuning of feature extractors and distance functions employed. For visual benchmarking, projections of the FV space to display space are popular, where the benchmark objects are mapped to and visualized in 2D, sometimes also in 3D. We assume a suitable projection technique exists which manages to transfer the essence of the distance relationships of a data set given in FV (or metric) space to the low-dimensional display space. Then, benchmark-based class distribution characteristics can be analyzed. Important visual benchmarking questions include the discovery of interesting inter-class relationships, the assessment of compactness and discrimination properties of classes of objects, and the comparative evaluation of global discrimination power of different FV (or metric) space representation of a given data set. The visual analysis of such questions can help in selecting and engineering FV spaces to better support a given application. E.g., in a retrieval scenario, badly discriminated classes could be identified, and the FV space then fine-tuned accordingly to improve the discrimination of the identified classes.

1.4 Considered multimedia data types

As indicated above, multimedia as a data type definition is not very specific, but may generally denote non-standard (non-structured, non-relational) data. Many different multimedia data types exist. Throughout this thesis, we develop and discuss FV-based techniques for managing multimedia data along several different data types. We next sketch the specific data types considered later on in this thesis. Figure 1.2 illustrates the data types.

3-Dimensional (3D) Objects 3-Dimensional objects (or models) are used to represent real-world or artificial objects. They are prominently used in CAD-based engineering for modeling e.g., machining parts. They are also used for rendering/visualization purposes in simulation, education, and entertainment. Model formats include boundary representations, e.g., by means of polygon meshes, point clouds, or implicit surfaces. Also, volumetric representations by voxels or composition of primitives in Constructive Solid Geometry (CSG) are used in practice [21]. 3D databases used in this thesis include the Konstanz 3D database [19, 61] and the Princeton Shape Benchmark [81]. As FVs we use a variety of different features defined over curvature, volumetric, statistic, and image-based object properties [18, 97]. The corresponding FV extractors were originally proposed and/or implemented by our colleagues Dietmar Saupe and Dejan Vranic.

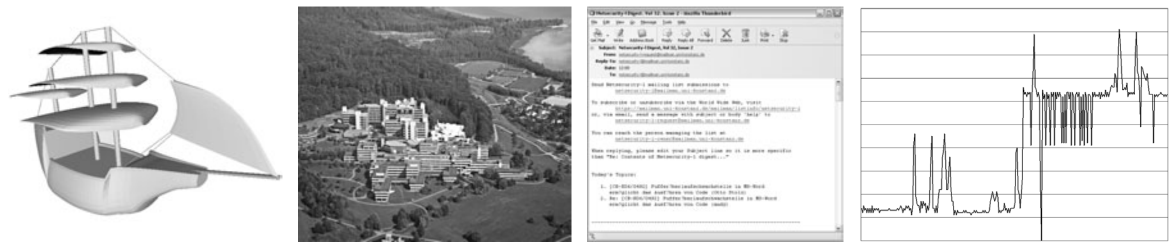
Raster Images Images may very well represent the most popular multimedia data type. Digital image sources include acquisition from the real-world using digitalization (think of the ubiquitous digital cameras; also remote sensing images are obtained from observation satellites); and artificial generation by rendering or manual drawing. Various lossy or lossless schemes for encoding and transmitting raster images exist. Many FV extractors based on characteristics such as color histograms, texture descriptions, or edges distributions have been proposed in context of content-based image retrieval (CBIR). It is also possible to segment images into different partitions prior to feature extraction, possibly identifying semantically meaningful image parts. In this thesis, we use a classified subset of the COREL image database which is often used in CBIR evaluation. The database contains 6.000 images each described by six different FVs based on color histograms, texture, and convolution descriptors, cf. [75]. We obtained the data from Peter Howarth and Stefan Rüger.

Email Documents Text is a highly complex data type. Its degree of structure can vary from completely unstructured text, e.g., a short office note written on a post-it, up to highly structured texts such as a research paper following a sequence of chapters from introduction to conclusion and adhering to certain rules for stating facts, citing other work and so on. The RFC 2822 (Internet Message Format) defines the document structure for E-Mail messages to be transmitted over the Internet. E-Mail consists of fields containing sender and receiver addresses, subject line, and mail body text. A classical descriptor for text documents is the so-called $tf \times idf$ vector, which basically consist of weights rating the importance of given words in a given document with respect to a collection of documents. We have collected about 10.000 E-Mails from our working group, classified as either spam (non-solicited messages) or non-spam. We defined a $tf \times idf$-based FV over these E-Mail, and apply it for database organization and retrieval.

Time Series Time series is also a rich data type of great importance in many application domains ranging from Science and Engineering to Business and Finance. Generally, a time series denotes a sequence of one- or multidimensional measures with associated time stamps that can be equally or non-equally spaced in time. Depending on the application, many aspects can be of importance in a time series, e.g., we can analyze for occurrence of certain predefined patterns like in financial technical analysis, presence or absence of trends, cycles, and seasonal effects in econometric series, or simply extreme values like in many basic monitoring applications, for example. Based on these patterns of interest, many different FVs can be defined. In this thesis, we consider time series from the financial domain (S&P-500 [84] and Fund-based [54] price series).

1.5 Thesis outline and contribution

Two main ideas motivate this thesis. We first recognize that the FV approach to management of multimedia data is promising, but needs attention of FV *selection and combination* in order to serve as a basis for building effective multimedia applications. Secondly, we believe that



(a) 3D object (b) Raster image (c) Email document (d) Time series

Figure 1.2: Four types of multimedia data considered in this thesis.

appropriate *visualization* is suited for designing effective analysis tools and user interfaces for the FV as well as the object space.

In this thesis, we focus on effectively supporting a number of important tasks given in FV-based multimedia databases. Specifically, we propose innovative methods (a) for effective processing of content-based similarity queries, (b) for FV space visualization for database organization and supervised/unsupervised FV discrimination analysis, and (c) for layout generation for content presentation in object space. The methods are applied and evaluated on a number of specific multimedia data types and are expected to be useful in many other multimedia domains. The remainder of this thesis is structured as follows.

- In Chapter 2, we review and classify a significant number of recently proposed FV extractors supporting the 3D model domain. Extensive effectiveness evaluation experiments are performed for many FV extractors on a number of benchmarks. Then, methods for improving retrieval effectiveness by forming static and query-dependent combinations of multiple FVs are researched. Experiments show significant improvements in retrieval quality to be achievable. These aspects contribute to developing effective retrieval systems.

- In Chapter 3, we turn to visualization support for feature-based retrieval systems. We first explore the usage of interactive 2D projections for retrieval and organization of database content in a multi-feature 3D retrieval system. Self-organizing maps (SOMs) have been shown to be appropriate to this end. We then research two SOM-based techniques for unsupervised comparative estimation of the discrimination power provided in multiple FV spaces. The techniques are specifically advocated to support the FV selection process in cases where supervised benchmarking is not possible or too expensive. We finally introduce a simple but effective visualization metaphor supporting supervised visual discrimination analysis in projected FV space. Both lines of FV space visualization can be used for addressing the FV selection problem, for fine tuning FV-based multimedia applications, and for visual FV-space analysis.

- Chapter 4 then deals with layout generation for content presentation in object space. We first review the popular TreeMap family of layout algorithms. Inspired by these works, we develop space-efficient layout generators providing certain regularity and ordering properties which are useful for presenting multimedia content to the user. The layouts are applied on sets of time series data, and experimentally shown to outperform competing TreeMap algorithms on a number of metrics. Specifically, the ID-Map algorithm is developed which replaces the standard slice-and-dice TreeMap strategy with a novel splitting mask-based strategy. Splitting masks are predefined rectangular partitioning schemes supporting regular layouts. In the last Section of Chapter 4, we develop a second alternative TreeMap scheme operating directly on a global regular grid of element cells in order to produce displays of high regularity as required by certain data types such as images or time series data. The Grid-TreeMap is derived, and several rendering methods based on the approach are experimentally examined by application and experiments.

- Chapter 5 finally summarizes the thesis, draws conclusions, and identifies future research directions we regard promising in the context of this work.

The thesis structure follows the three topics FV-based query processing and evaluation (for 3D model FVs), visual analysis of feature space, and layout generation for the object space. These three aspects are rather orthogonal and each complement to multimedia database management and application. Owing to this structure, we have decided against having a separate related work Chapter, but rather, have the discussion of related work where appropriate, interleaved with the flow of the thesis.

Figure 1.3 concludes the Introduction by extending the FV extractor pipeline given in Figure 1.1 (a) to denote the wealth of different multimedia applications possible by the FV approach to multimedia database management. Specifically, on the right-hand-side it illustrates (from top to down) content-based retrieval, projection-based database browsing, and object space layout generation controlled by a feature-based distance function.

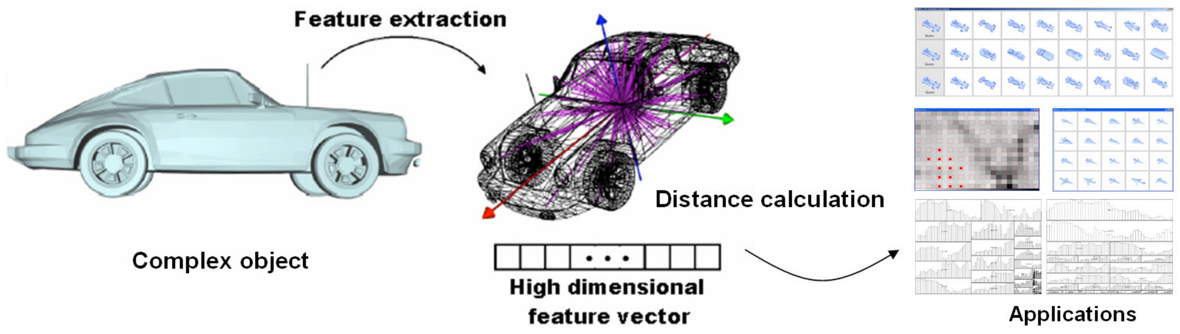


Figure 1.3: Extension of the feature extraction pipeline from Figure 1.1 (a) by including the application layer.

2 Effective feature-based query processing

Contents

Content-based similarity queries are one of the main applications in multimedia databases. For implementing similarity search systems, the usage of Feature Vectors (FVs) is popular due to the simplicity and generality of the approach. On the other hand, setting up an effective content-based retrieval system requires some considerations. Appropriate FV extractors have to be defined for the given multimedia data type to be supported. Questions regarding provision of certain types of invariances, FV resolution and representation, etc. have to be addressed. As it turns out, many different feature extractors are usually possible. While good settings can be found experimentally using single FVs, as will be shown the system can profit from appropriately combining multiple different FVs for query processing.

In this Chapter, Section 2.1 introduces the main concepts of content-based retrieval in multimedia databases. In Section 2.2, we specifically consider the problem of FV extraction in 3D databases, a prominent data type which has only recently been supported by appropriate FV extractors. A process model for 3D feature extraction is proposed. In Section 2.3, the retrieval effectiveness of a range of 3D FV extractors is experimentally benchmarked, identifying strengths and weaknesses of the methods. Motivated by benchmarks comparing individual FV methods, options for building static and query-dependent combinations are researched in Sections 2.4 and 2.5. Both options significantly improve the retrieval precision as shown on several 3D benchmarks.

Parts of this Chapter appeared in [18, 19, 15, 17].

2.1 Content-based retrieval systems

In this Section, we introduce the main concepts associated with content based multimedia retrieval systems. Section 2.1.1 discusses similarity queries in general. Section 2.1.2 introduces the usage of FVs for answering similarity queries, and Section 2.1.3 details methods how the quality in terms of answer precision of a given retrieval system can be practically benchmarked.

2.1.1 Similarity queries and Feature Vectors

A similarity query to a multimedia database requests those objects from the repository which are most *similar* to a given *query specification*. Implicitly or explicitly, a similarity query provides specific attributes which it is referring to, and specifies how similarity is measured on the identified attributes. Under the FV paradigm, objects are represented by vectors of real-valued components which numerically capture important properties (features) of the objects. The similarity between two objects is associated with the *distance* resulting between the objects' FVs under a given metric defined in vector space. Therefore, the user could in principle issue a similarity query by manually setting the component values which are of interest, thereby directly editing the query vector. This approach is only useful if the number of components in the FV space is small and the meaning of the dimensions (properties addressed by the FV components and their respective scales) as well as their influence regarding the distance function employed is known to the user. This is usually not the case in Multimedia databases where the FVs typically compound dozens or even hundreds of dimensions which often are not easily interpretable by the user, e.g., if the features are represented in the frequency domain. Therefore, the *query-by-example* approach is practical. It is based on the user providing the query point by a representative object, the query object. Such a query object can originate from different sources. It can be selected by the user while browsing through the object data base. Also, the user sometimes already has such query objects in her possession, or she can provide a query by a rough sketch or approximation. The latter option is dependent on the data type and can involve drawing a sketch when querying for 2D/3D shape, or humming of a melody in audio retrieval.

Given that a query point is provided, also the similarity function used to answer the query is required. It consists of the required feature(s) to use as well as the distance function to employ. The former is necessary as based on the application or user task, not all of the possibly many features which can describe a given data type are also requested by the user. In multimedia data, typically many different features (aspects) can be perceived in a given application, not all of them relevant to all users. E.g., in an image retrieval scenario, the user may be interested in specific texture patterns occurring in the desired images, but may not care for the specific colors which are involved. This in effect is a feature selection problem which the user has to address. Finally, also the distance function operating in FV space has to be specified. An important family of distance functions in vector spaces is the *Minkowski* (L_s) family of distances, defined as

$$L_s(\vec{x},\vec{y}) = \left(\sum_{1 \le i \le d} |x_i - y_i|^s \right)^{1/s} \in \mathbb{R}_0^+, \quad \vec{x},\vec{y} \in \mathbb{R}^d.$$

Examples of these distance functions are L_1, which is called *Manhattan distance*, L_2, which is the *Euclidean distance*, and $L_\infty = \max_{1 \le i \le d} |x_i - y_i|$, which is called the *maximum distance*. Other more specialized distances from statistics such as Kolmogorov-Smirnow, χ^2 statistics etc are possible. Also, parameterizations are possible for many distance functions, e.g., the Minkowski distances can be weighted to let the user express importance or preferences for certain FV components. But this is usually not considered directly in interactive query specification, as the average user cannot be expected to have an understanding on the outcome of such parameterizations on the retrieval results. Rather, such parameterizations of the distance functions are left for semiautomatic optimization using relevance feedback techniques. There, relevance judgements supplied by the user are leveraged to find distance parameterizations expected to improve sub-sequent search iterations.

If the similarity query (query point q and distance function d) is specified, the query can be processed by the system. To this end, based on the real valued vectors describing the objects in a database, a similarity query for a query object q is usually executed as a k-NN query, returning the k objects whose FVs have the smallest distance to q under distance d, sorted by increasing distance to the query.

Figure 2.1 shows an example of a content-based similarity query in a 3D object database. The first object in the row is the query object (a model of a Formula-1 racing car), and the next objects are the nearest neighbors retrieved by the search system. Objects considered relevant are marked by blue dots in the answer list. Note that the ranking contains several irrelevant objects, and further relevant objects may be encountered at later positions in the ranking.

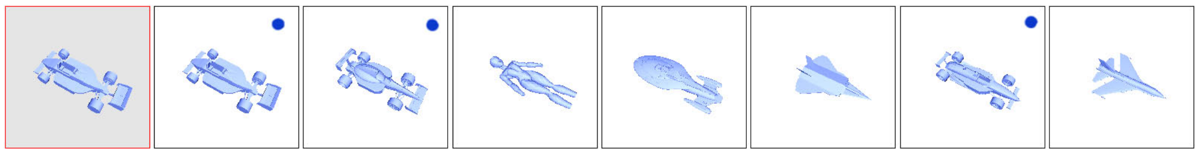


Figure 2.1: A query-by-example in a 3D multimedia database. The leftmost object is the query example. On the right, the nearest neighbors to the query are displayed. Relevant answer objects are marked by a dot.

2.1.2 Effectiveness and efficiency considerations

Several desirable properties can be identified in FV-based search systems. Generally, the system should support *effective* and *efficient* retrieval. Efficiency refers to the consumption of resources needed for storage and retrieval of the multimedia objects and can be measured by metrics such as response times and disk utilization. Effectiveness of the system relates to the usability of the query interface provided. It also relates to the quality of the answer objects returned by the search system, and this point is of primary interest here. Quality of the answers measures the degree of relevance among the answer list with respect to the query object. An effective retrieval system is supposed to return the most relevant objects from the database on the first positions from the k-NN query, and to hold back irrelevant objects from this ranking.

Effectiveness and efficiency in a FV-based search system are determined primarily by the implemented FVs. This leads to the following requirements for good FVs. Regarding efficiency, we can identify:

1. Efficiency of FV extraction.
2. Efficiency of FV representation.
3. Embedded multi-resolution property.

Fast extraction makes it possible to perform database inserts on the fly, where FVs are calculated for any new object to be inserted in real time. Efficiency of representations requires the vectors to consume minimal space in terms of number of vector components and number of bits used to encode the component values. Short FVs reduce the amount of disk/memory space required to store the FVs, and speed up distance calculations and access to the vectors. Specifically, the performance of vector space index structures deteriorates quickly if the dimensionality of the indexed data grows [10]. Often, there is a typical tradeoff between resolution (size) of the FVs, and the provided discrimination power (cf. Section 2.3.4), in that higher dimensionality leads to better retrieval precision. Therefore, the embedded multi-resolution property is desirable. FVs with this property encode progressively more object information inside a given FV, meaning that by considering subsets of dimensions in embedded multi resolution FVs allows to chose the level of detail of the object description. This is a clear advantage, as for non multi-resolution FVs, one FV has to be stored for each resolution level to be supported by the system.

Regarding effectiveness, the following FV properties are desirable:

4. Sufficient discrimination power.
5. Desired invariance properties.
6. Robustness properties.
7. Interpretability of components.

Discrimination power requires that an appropriate distance function defined in FV space effectively captures the similarity relationships present in object space by distances in FV space. Therefore, discriminating object features have to be selected on which the FVs are based. Next, depending on the multimedia data type and application, certain invariances of the search may be desired, meaning that distances in FV space should be invariant with respect to certain application-dependent object transformations which are considered leaving similarity relationships unchanged. E.g., in many cases the resolution of two raster images should not affect the similarity measure between them. Robustness is another effectiveness criterion often demanded, implying that small variations in the multimedia objects, e.g., caused by noise, should not dramatically alter the resulting distance between the objects in FV space. Finally, also interpretability of the components can be seen as an effectiveness criteria. If the user can understand the meaning of the FV components, she can directly configure the distance function employed, e.g., by specifying component weights to be used.

2.1.3 Benchmarking effectiveness in a retrieval system

The effectiveness of a similarity search system can be assessed by different approaches. Under the *user oriented* approach, a number of users are to perform similarity search tasks using the algorithms (FVs) under concern, and then certain measures of user satisfaction are aggregated. While this approach can reflect user satisfaction in real-world application settings, such experiments usually are not quantitatively reproducible and need careful definition of user tasks and user groups, therefore they are expensive. Objective and reproducible effectiveness evaluations are possible if there exist suitable *ground-truth* classified data sets (benchmarks) on which similarity search methods can be evaluated. Examples of multimedia benchmark data collections include the TREC text archives for information retrieval [68], or the UCI machine learning repository [25] for data mining research. In content-based image retrieval (CBIR), collections of images published by Corel Corporation are prominently used [65]. In time series retrieval, the UCR time series repository maintained by Eamonn Keogh is a reference

[57]. In the 3D model domain, also several benchmarks exist, most prominently to date the Princeton Shape Benchmark [81], among others (cf. Section 2.3.2). A ground truth benchmark is a data set of objects with an associated classification scheme defining membership of objects in similarity classes. Such classifications can be disjoint or overlapping, and hierarchic or flat. The classification is usually manually created according to common and intuitive understanding of similarity. On the benchmark, queries are performed for benchmark objects, and then, statistics on the precision of the produced answer lists are calculated which allow to compare the quality of different search algorithms under the benchmark.

In Information Retrieval, different measures for the quality of answer lists given a reference classification are possible. A nice overview is given e.g., in [89, 6], and also in [81]. In this thesis, we consider the widely used *precision-recall* metrics for comparing the effectiveness of the search algorithms. We consider benchmark classifications where the similarity is given in binary form: Given a query object q, an answer object o is either relevant to the query, or irrelevant, and relevance does not differ between two objets which are both relevant (or irrelevant) to the query object. Then, for a given answer list, *precision* (P) is the fraction of the retrieved objects which are relevant to q, and *recall* (R) is the fraction of the complete set of relevant objects in the benchmark, which are contained in A. That is, if N is the number of objects relevant to the query, A is the number of objects retrieved and R_A is the number of relevant objects in the result set, then

$$P = \frac{R_A}{A}, \text{ and } R = \frac{R_A}{N}.$$

Precision-recall diagrams plot the precision values obtained for answer sets for increasingly higher recall levels. They can be normalized to the eleven standard recall levels (0%, 10%, . . . , 100%) [6]. Given a set of benchmark queries, the precision-recall diagrams obtained for each query can be averaged to obtain aggregated precision-recall results for a whole benchmark, or for a subset of the benchmark, e.g., a specific benchmark query class.

In addition to precision at multiple recall points, we also consider the *R-precision* measure [6] (also known as *first tier* [81]) for each query, which is defined as the precision when retrieving the first (nearest) N objects. The R-precision gives a single number to rate the performance of a retrieval algorithm:

$$\textit{R-precision} = \frac{R_N}{N}$$

Precision-recall statistics can be used in so-called one-shot experiments, where each query object from the benchmark is queried using a given search algorithm. Then, the benchmark can be evaluated fully automatically. In systems which employ relevance feedback trying to improve search results during consecutive iterations involving the user giving relevance scores to answer lists, then also the change (delta) in the precision-recall statistics is interesting. Systems which need fewer relevance feedback iterations to achieve a given level of precision, or where the rate of improvement is larger on an average relevance feedback iteration, are considered better.

Figure 2.2 illustrates the comparison of four different retrieval methods using a benchmark query. A F-1 car model (leftmost column) is queried in a 3D benchmark; racing cars are considered relevant, while all other objects are considered irrelevant to the query. The respective answer lists A_1 to A_4 of size 15 are given in the four rows. The precision of the answer sets amounts to $P_1 = \frac{6}{15}, P_2 = \frac{8}{15}, P_3 = \frac{5}{15}$, and $P_4 = \frac{8}{15}$, respectively (the relevant objects are marked by dots). Regarding the full answer lists of length 15, we consider methods 2 and 4 best in this example, as both retrieve 8 relevant answer objects. If we consider smaller prefixes of the answer lists, we see that method 4 manages to report the relevant objects at earlier positions in the rankings, so if short answer lists are demanded by the system, then this algorithm is the best choice among the four algorithms.

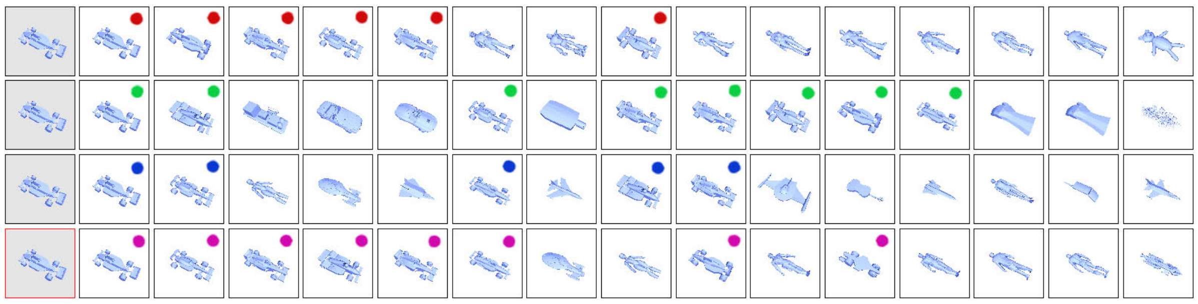


Figure 2.2: Results of a query using the DBF, SIL, and CPX FVs (cf. Section 2.3) are shown in the first three rows, respectively (answer list A_1, A_2, and A_3). The last line (answer list A_4) shows the answers using a static unweighed combination of all three FVs. Objects relevant to the query object are marked by dots.

2.2 Modeling FV extraction for 3D objects

2.2.1 3D objects data type

3D objects are an important multimedia data type with many application possibilities. 3D models can represent complex information, and the problem of searching for similar 3D objects arises in a number of fields. Example application domains include Computer Aided Design/Computer Aided Manufacturing (CAD/CAM), Virtual Reality (VR), Medicine, Molecular Biology, Military applications, Entertainment, and so on. E.g., in Medicine the detection of similar organ deformations can be used for diagnostic purposes [51]. 3D object databases are also used to support CAD tools which have many applications in industrial design and manufacturing. For example, standard parts used in manufacturing processes can be modeled as 3D objects. When a new product is designed, it can be composed by many individual parts which fit together to form the product. If some of these parts are similar to one of already existing standard parts, then the re-usage of standard parts can lead to a reduction of production costs, as compared to the re-designing of the requires parts. As another application, movie and video game producers make heavy usage of 3D models to enhance realism in entertainment applications. Re-usage and adaptation of 3D objects by similarity search in existing databases is a promising approach to reduce production costs.

As 3D objects are used in diverse application domains, different forms for object representation, manipulation, and presentation have been developed. In the CAD domain, objects are often built by merging patches of parameterized surfaces, which are edited by technical personnel. Also, constructive solid geometry techniques are often employed, where complex objects are modeled by composing primitives. 3D acquisition devices usually produce voxelized object representations or clouds of 3D points. Other representations like swept volumes or 3D grammars exist. Probably the most widely used representation to date is to approximate a 3D object by a mesh of polygons, usually triangles (cf. Figure 2.3 for a 3D mesh and a rendered image of it.) For a survey on important representation forms, see [21]. For 3D retrieval, basically all of these formats may serve as input to a query-by-example. Where available, information other than pure geometry data can be exploited, e.g., structural data that may be included in a Virtual Markup Language (VRML) representation. Many similarity search methods that are presented in the literature up to date rely on triangulations, but could easily be extended to other representation forms. Of course, it is always possible to convert or approximate from one representation to another one.

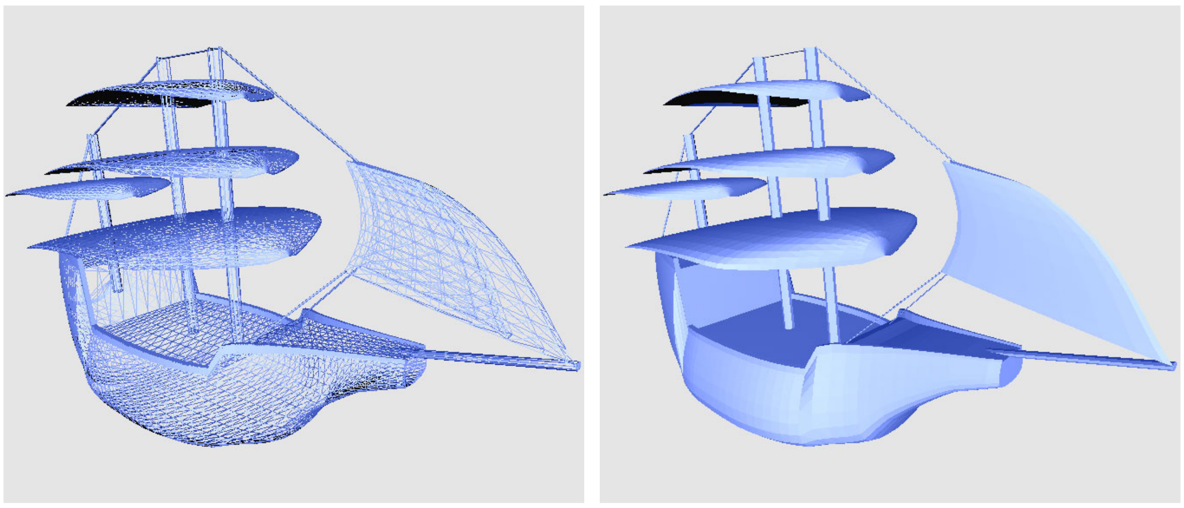

(a) Polygon mesh (b) Rendered image

Figure 2.3: Visualization of a polygon mesh (a) and a typical rendering result (b).

2.2.2 A new process model for classification of 3D FV extractors

Candidate features for usage in a 3D FV extractor depend on the specific format in which the models in the considered database are given. Often, information about properties of surface (e.g., texture or reflection properties), volumetric aspects (e.g., mass density), or structure (e.g., hierarchical containment relationships defined on parts of the model) cannot be assumed in a model representation. A property common to most representation forms is geometry, and consequentially, 3D FV extractors usually rely on geometry information to generate 3D FVs.

In this Section, we next propose a model of the 3D descriptor definition process, followed by a discussion of important requirements typically demanded for 3D FVs.

3D FV extraction process model

The extraction of shape descriptors generally can be regarded as a multistage process like illustrated in Figure 2.4. In this process, a given 3D object, usually represented by a polygonal mesh, is first preprocessed to achieve required invariance and robustness properties. Then, the object is transformed so that its character is either of surface type, or volumetric, or captured by one or several 2D images. Then, a numerical analysis of the shape takes place, from the result of which finally the feature descriptors are extracted. We briefly sketch these basic steps in the following.

1. *Preprocessing.* Several requirements that suitable methods for 3D similarity search should fulfill can be identified. The methods should be *invariant* with respect to changes in rotation, translation, and scale of 3D models in their reference coordinate frame. Ideally, an arbitrary combination of translation, rotation and scaling operations applied to one object should not affect its similarity measure with respect to another object. In other words, the features comprising the shape descriptor ideally should not depend on the arbitrary coordinate frames that the authors of 3D models have chosen. Suitable methods should also be *robust* with respect to variations of the level-of-detail, and to small variations of the geometry and topology of the models. In some applications, invariance with respect to anisotropic scaling may also be desirable.

2. *Type of object abstraction.* A polygonal mesh can be seen in different ways. We may regard it as an ideal mathematical surface, infinitely thin, with precisely defined properties of differentiability. Alternatively, we may look at it as a thickened surface that occupies some portion of volume in 3D space, or for watertight models as a boundary of a solid volumetric object. The transformation of a mesh into one of these forms is typically called voxelization. Statistics of the curvature of the object surface is an example of a descriptor based directly on a surface, while measures for the 3D distribution of object mass, e.g., using moment-based descriptors, belong to the volumetric type of object abstraction. A third way to capture the character of a mesh would be to project it onto one or several image planes producing renderings, corresponding depth maps, silhouettes, and so on, from which descriptors can be derived.

3. *Numerical transformation.* The main features of meshes in one of the types of object abstractions outlined before can be captured numerically using one of various methods. Voxel grids and image arrays can be Fourier or Wavelet transformed, and surfaces can be adaptively sampled. This yields a numerical representation of the underlying object. It is not required that the numerical representation allows the complete reconstruction of the 3D object. However, these numerical representations are set up to readily extract the mesh shape descriptors in the final phase of the process.

4. *Descriptor generation.* We propose to group the descriptors for 3D shape in three main categories based on their form.

a) *Feature vectors*, or *FVs*, consist of elements in a vector space equipped with a suitable metric. Usually, the Euclidean vector space is taken with dimensions that may easily reach several hundreds. Such Feature Vectors may describe conceptually different types of shape information, such as spatial extent, visual expression, surface curvature, and so forth.

b) In *statistical approaches*, 3D objects are inspected for specific features, which are summarized usually in the form of a histogram. For example, in simple cases this amounts to the summed up surface area in specified volumetric regions, or, more complex, it may collect statistics about distances of point pairs randomly selected from the 3D object.

c) The third category is better suited for *structural 3D object shape description* that can be represented in the form of a graph [85, 41]. A graph can more easily represent the structure of an object that is made up of or can be decomposed into several meaningful parts, such as the body and the limbs of objects modeling animals. However, finding a good similarity measure for graphs is not as straightforward as for Feature Vectors, and, moreover, small changes in the 3D object may lead to large changes in the corresponding structural graph, which is not ideal for solving the retrieval problem.

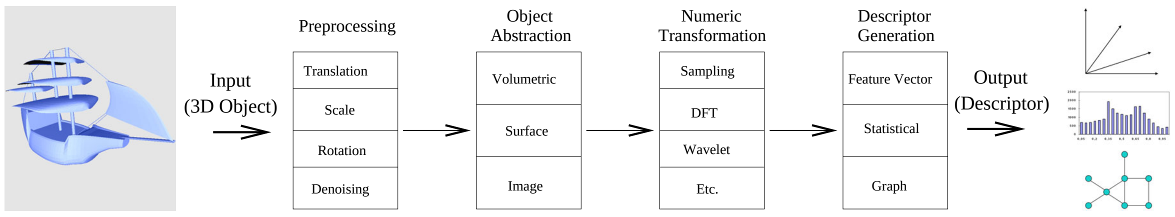


Figure 2.4: 3D FV extraction process model.

For a classification of 3D object retrieval methods we use the type of object abstraction from the second stage of the extraction pipeline as the primary category. Thus, we ask whether the descriptor used in the respective method is derived directly from the surface, or whether it is based on an intermediate volumetric or image type of abstraction. For a second level of differentiation we propose to look at the form of descriptors (Feature Vector, statistical, or structural). Therefore, we adopt a classification based on the abstraction setting and the form of descriptors rather than the semantics behind them. Other classifications for shape description and analysis methods are possible, see for example the survey of Tangelder and Veltcamp [87] or Loncaric [64]. The methods in the Feature Vector class are efficient, robust, easy to implement, and provide some of the best approaches [87, 49, 97]. Therefore, these are the most popular ones that are explored in the literature. Also in this work, we restrict to this case as the currently dominant framework for 3D retrieval systems. We do not imply, however, that the other methods may be inferior and should therefore be discarded from future research. Most of these methods have their particular strengths and may well be the ideal candidate for a specific application.

Invariance and robustness requirements

The most important invariance properties of searching for 3D content refer to invariance with respect to translation, scale, and orientation of the models. It is also commonly demanded that the search is robust with respect to changes in the level of detail of the models, e.g., the resolution of the mesh which is modeling an object. These invariance and robustness properties can be achieved in different ways. If only relative object properties are used to define the descriptor, then the invariance is not a problem, e.g., as in [70]. These methods are typically found in the class of statistical methods. Invariance with respect to rotation can be achieved with energy summation in certain frequency bands of spectral representations of suitable spherical functions [34, 49]. In a generalization of this method to volumetric representations one may achieve rotational invariance by an appropriate combination of Zernike moments [67]. The invariance with respect to translation and to scale must be achieved in these methods by an a-priori normalization step, i.e., by translating the center of mass of the 3D object to the origin and by scaling the objects so that they can be compared at that same scale.

Otherwise, the invariance properties can be obtained approximately by an additional preprocessing normalization step, which transforms the objects so that they are represented in a canonical reference frame. In comparison to the above mentioned works, besides the translation of the coordinate origin and the definition of a canonical scale, also a rotational transformation must be applied in order to complete the normalization. In such a reference frame, directions and distances are comparable between different models. The predominant method for finding this reference coordinate frame is pose estimation by principal components analysis (PCA) [73, 98], also known as Karhunen-Loeve transformation. The basic idea is to align a model by considering its center of mass as the coordinate system origin, and its principal axes as the coordinate axes. An extension to normalizing (isotropic) scale is to factor out also anisotropic scale [50] so that the variance of the object along any direction is unity. This is achieved by scaling the object along its principal axes by the inverses of the corresponding eigenvalues. The three eigenvalues can be appended to the Feature Vector of the re-scaled object, and with an appropriate distance metric one may either completely disregard the anisotropy of the model or assign an arbitrary importance to it, depending on the application or user preferences [50].

While the majority of proposed methods employs PCA in some form or another, several authors have stability concerns with respect to the PCA as a tool for 3D retrieval. On the other hand, omitting orientation information also omits valuable object information. Thus, there is a tradeoff between achieving intrinsic rotation invariance without rotating the object in a canonical orientation, and the discrimination power that can additionally be attained by using an (approximated) canonical orientation. A detailed thorough empirical analysis would have to compare both cases to the retrieval performance achievable by optimal pairwise object alignment. This is a hard to do experiment and still outstanding. For a more detailed discussion see [34, 96, 66, 86].

2.3 Query processing using single Feature Vectors

2.3.1 Classification of studied FV extractors using the process model

In this Section, we experimentally evaluate the retrieval performance of a number of 3D feature extractors. These FV extractors were designed and/or implemented by our colleagues Dietmar Saupe and Dejan Vranic and detailed in [97]. The contribution of this Section is (a) to provide a comparative evaluation of a significant number of 3D FV extractors, and (b) to serve as a reference base line for measuring the improvement potential achievable by building static and dynamic combinations of FVs in Sections 2.4 and 2.5. We briefly sketch the principles of the used FV extractors used in this study in the following. The ordering is done by the object abstraction underlying the FV extractor, according to the descriptor extraction process model introduced in Section 2.2.2. For a more detailed discussion of the methods, as well as additional evaluation results please refer to [18, 19, 16, 97].

Volumetric methods. These methods perform a partitioning of 3D space and consider aggregates of model surface in each of the partitions. The *Rotational Invariant* (RIN) [47] method sums up sampled model surface points belonging to certain equivalence classes defined on the cells of a voxel grid inside which the models are scaled. The *Voxel* (VOX) [97] method considers the fraction of model surface occupying the cells of a voxel grid around the model. The *3DDFT* (3DDFT) [97] method considers the same samples as VOX, but represented in the frequency domain. The *Volume* (VOL) [97] FV is constructed by measures of model volume located inside a pyramid-like partitioning of the model's bounding box. The last method in this category is the *Harmonics 3D* (H3D) [34] FV, which considers a set of concentric binary functions defined on the voxel grid and transformed to the frequency domain using the Spherical Harmonics transform. Note that by design this FV provides rotation invariance by discarding rotation information from the description.

Surface-based methods. Methods in this class consider properties obtained from abstracting the 3D models to their surface. The *Moments* [PMOM] [73] describes a 3D model by enumerating certain moments calculated from the 3D coordinates of the centroids of all model faces (triangles, polygons). The *Ray Moments* [RMOM] [97] FV also considers a collection of moments, where the moments are calculated by sampling the model surface using equally-spaced rays emitted from the model centroid. The *Cords* [COR] [73] FV combines histograms over length and angles obtained from lines connecting the model centroid with the centroids of all model faces. *Shape Distribution SD2* [SD2] [70] describes a 3D model by a histogram of distances calculated for pairs of points randomly sampled on the model surface. The *Shape Spectrum* descriptor [SSD] [105] is a histogram over model surface curvature.

Image-based methods. These approaches extract features from one or several 2D projections of the 3D models. The *Silhouette FV* [SIL] [97] is a Fourier descriptor considering centroid-based distance samples of 2D renderings of the models. The *Depth Buffer* [DBF] [97] FV is also image-based, but considers grey-scale depth images rendered from the mod-

Table 2.1: Studied 3D FV extractors.

Descriptor name	Abbr.	Source	Preproc.	Object abstr.	Num. transf.	Type
Rot. Inv.	RIN	[47]	RTS	Volumetric	None	Histo
Voxel	VOX	[97]	RTS	Volumetric	None	Histo
3DDFT	3DDFT	[97]	RTS	Volumetric	3D DFT	FV
Volume	VOL	[97]	RTS	Volumetric	None	FV
Harmonics 3D	H3D	[34]	TS	Volumetric	Sph. Harm.	FV
Moments	PMOM	[73]	RTS	Surface	Sampling	FV
Ray Moments	RMOM	[97]	RTS	Surface	Sampling	FV
Cords	COR	[73]	RT	Surface	Sampling	Histo
Shape Dist. D2	SD2	[70]	None	Surface	Sampling	Histo
Shape Spectrum	SSD	[105]	None	Surface	Curve fitting	Histo
Silhouette	SIL	[97]	RTS	Image	Sampling + DFT	FV
Depth Buffer	DBF	[97]	RTS	Image	2D DFT	FV
Ray-Based	RAY	[97]	RTS	Image	Sampling	FV
Rays-SH	RSH	[97]	RTS	Image	Sampling + Sph. Harm.	FV
Shading-SH	SHA	[97]	RTS	Image	Sampling + Sph. Harm.	FV
Complex-SH	CPX	[97]	RTS	Image	Sampling + Sph. Harm.	FV

els. The *Ray-based* [RAY] [97] method can be regarded as a real-valued depth image in spherical coordinates. The *Rays-SH* [RSH] [97] considers the Spherical Harmonics transform of the Ray-based depth image. *Shading-SH* [SHA] [97] performs a spherical projection of shading information calculated from surface normal vectors. Finally, the *Complex-SH* [CPX] [97] FV combines the RSH and SHA features in a complex-valued measure represented in the frequency domain.

Please refer to table 2.1 for an overview over the methods in the order as listed above. The *preprocessing* field lists the explicitly supported invariances (R: rotation; T: translation; S: scale). The *object abstraction* field refers to the abstraction type from the process model discussed in Section 2.2.2. *Numeric transform* gives the method applied on the model samples such as Discrete Fourier or Spherical Harmonics transform, Sampling, or fitting of curves. Finally, the *type* field indicates wether the vectors are histograms or arbitrary real-valued vectors. Please note that the table also includes *abbreviations* for the individual methods. We will make repeated use of these abbreviations to refer to the individual 3D FV extractors throughout the remainder of this thesis.

All these FV extractors were heuristically introduced. They were inspired by various Computer Graphics (e.g., DBF or SIL), Geometry (e.g., SSD), or Signal Processing (e.g., the idea of representing features in the frequency domain) techniques. Up to special cases it is a-priori quite unclear which of these features should be preferred for addressing the general 3D retrieval problem. Each of the descriptors captures specific model information, and their suitability for effective retrieval needs to be experimentally evaluated.

2.3.2 3D retrieval benchmark design

The effectiveness of the heuristically introduced 3D FV extractors can be experimentally evaluated using the benchmarking approach on a suitable ground truth database as described in Section 2.1.3. The availability of commonly accepted, high-quality reference benchmarks is a beneficial factor in finding effective FV extractors in many multimedia applications. In the content-based 3D retrieval field, which is not yet as established as other retrieval domains, there are not too many benchmarks available. Probably the most prominent 3D benchmark was released in 2004 by the Princeton Shape Retrieval and Analysis Group: The *Princeton Shape Benchmark* (PSB) [81]. This benchmark consists of a carefully compiled set of 1,814 3D models in polygon mesh representation that were harvested from the Internet. The benchmark also includes object partitioning schemes on several levels of abstraction, that is, several definitions of disjoint classes of objects, where all objects within the same class are to be considered similar. The benchmark is partitioned in one *Training* and one *Test* set, each containing half of the models. As to the types of objects considered, the PSB consists of models representing object classes that are familiar from the real world, such as animals, plants, vehicles, tools, or accessories. Not included are model classes from specialized application domains, e.g., CAD engineering or molecular biology. Of the different PSB classification schemes defined, the *PSB-Base* classification represents the most selective classification granularity, grouping objects strictly by function (semantic concept) as well as global shape. For our subsequent effectiveness evaluations, we consider this base classification.

In our own work, we also compiled a 3D benchmark database for evaluation purposes: The *Konstanz 3D Database* (KN-DB) [19, 61]. The KN-DB contains $1,838$ 3D objects which we harvested from the Internet, and from which we subsequently manually classified 472 objects by global shape and function into 55 different model classes (the remaining models were left as “unclassified”). Table A.1 in the Appendix lists the individual query classes defined in the KN-DB benchmark, along with the number of member objects per class. Similar to the Princeton Shape Benchmark, the KN-DB benchmark contains mesh models representing “real-world” objects such as humans, animals, vehicles, and so on. Please refer to Figure A.1 in the Appendix for an illustration of member objects from some of the query classes contained in the KN-DB benchmark.

Comparing model types and classification philosophy in the PSB-Base and the KN-DB, we find that the partitioning of models into similarity classes was done in the same spirit, and both databases contain similar classes of objects. Having this in mind, the following evaluation, which is based on these two benchmarks, is valid for these ’real-world’ 3D objects. Supposing that these model types form a significant part of the models freely available today on the Internet, the results give hints for selecting algorithms for building general-purpose 3D Internet search engines.

We evaluated the implemented FVs using different levels of resolution (dimensionalities), from 3 up to 512 dimensions, testing many different resolution settings as allowed by the individual methods. For object preprocessing prior to FV extraction, we apply the continuous principal component analysis (CPCA) [97] for those descriptors that require pose nor-

Table 2.2: Average R-precision of the 3D descriptors (KN-DB).

Descriptor	Abbr.	Best dim.	Avg. R-prec.
Depth Buffer	DBF	366	0.3220
Voxel	VOX	343	0.3026
Complex	CPX	196	0.2974
Rays-SH	RSH	105	0.2815
Silhouette	SIL	375	0.2736
3DDFT	3DDFT	365	0.2622
Shading-SH	SHA	136	0.2386
Ray-Based	RAY	42	0.2331
Rotation Invariant	RIN	406	0.2265
Harmonics 3D	H3D	112	0.2219
Shape Distribution D2	SD2	188	0.1930
Ray Moments	RMOM	363	0.1922
Cords	COR	120	0.1728
Moments	PMOM	31	0.1648
Volume	VOL	486	0.1443
Shape Spectrum	SSD	432	0.1119

malization. Fixing a FV, we obtain benchmark-average retrieval performance values by first executing queries for all classified objects defined in the benchmark, and then averaging the query-specific precision and recall statistics. In a similar way, we obtain class-average results by averaging only over the queries belonging to a given class. We obtain the retrieval performance statistics by considering the binary relevance status of answer objects according to class membership, and applying the formulas recalled in Section 2.1.3.

2.3.3 Benchmark-global effectiveness results

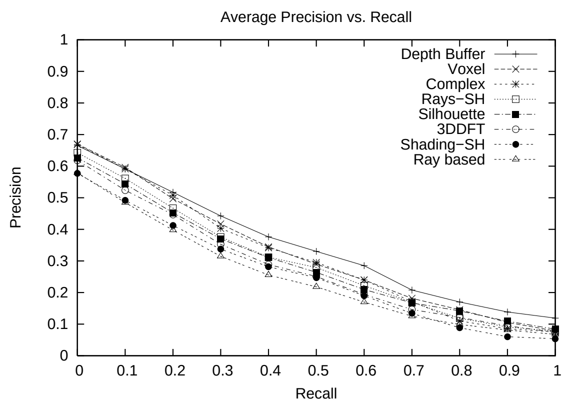
Database-average effectiveness results

Table 2.2 shows the best average R-precision values obtained for all implemented descriptors over all queries from the *KN-DB* benchmark, and their corresponding best dimensionality settings. The most effective FV according to this measure is the image-based Depth Buffer FV set to 366 dimensions.

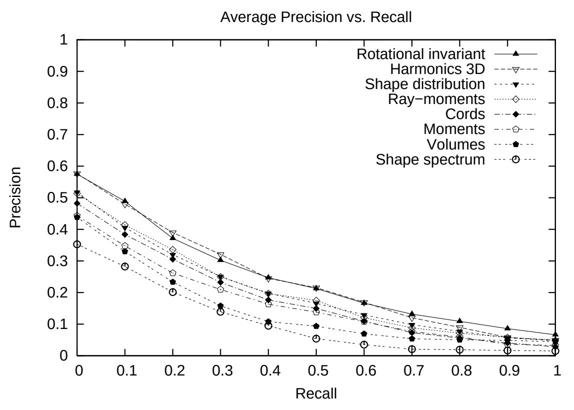
Figures 2.5 (a) and (b) show the precision vs. recall figures for all the implemented descriptors, evaluated on the KN-DB. The difference of the average R-precision values between the best performing descriptors is small, which implies that in practice these FVs should all be suited equally well for retrieval of “general-purpose” polygonal 3D objects. As a contrast, the effectiveness difference between the best and the least performing descriptor is significant (up to a factor of 3). We observed that descriptors which rely on consistent polygon orientation like Shape Spectrum or Volume exhibit low retrieval rates, as consistent orientation is not guaranteed for many of the models retrieved from the Internet. Also, the moment-based descriptors in this test seem to offer only limited discrimination capabilities.

Figures 2.6 (a) and (b) give the query-average precision vs. recall curves for the *PSB-Test* database when using the FV resolution providing the best average R-precision for this database (we include database-specific optimal dimensionality setting and achieved R-precision numbers in the legend). It is interesting to note that the results from the PSB-Test are quite similar to the ones obtained with the KN-DB. Despite the two databases having differences in size, models, and classification, the ranking of descriptors by retrieval performance, as well as the absolute performance figures are well comparable. Comparing the descriptor rankings from the KN-DB and the PSB-Test, there occur certain switches in the rankings, but all switches take place on roughly the same R-precision level. The two best performing descriptors and the four least performing descriptors retain their positions. We attribute the similarity of the retrieval performance results to the fact that both databases contain a comparable distribution of models, and manual classification was done in a comparable manner, namely, according to function and shape.

We also evaluated the descriptors' retrieval performance on the *PSB-Train* database. While the absolute retrieval performance level using the PSB-Train (as measured by R-precision) is slightly higher than on the PSB-Test (about two percentage points), the descriptor rankings by retrieval performance are the same on both PSB partitions, except for one adjacent rank switch occurring between the eighth and ninth position in the ranking. This is not surprising, considering the construction of the PSB Training and Test partitions [81].



(a)

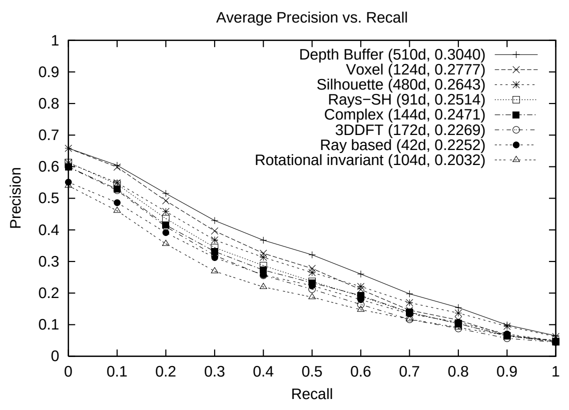


(b)

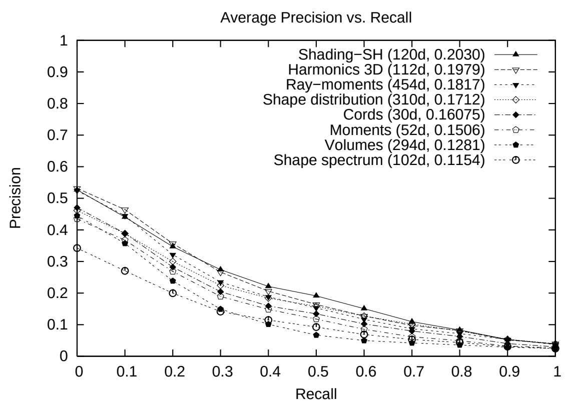
Figure 2.5: Average precision vs. recall with best dimensionality settings (KN-DB).

Specific query classes

The preceding experimental results compared the benchmark-average retrieval performance of the different FV extractors. The Depth Buffer was found to outperform the other methods on average over all queries. It is interesting to also compare the retrieval performance for individual query classes and ask whether the FV ranking obtained on average also holds on the



(a)



(b)

Figure 2.6: Average precision vs. recall with best dimensionality settings (PSB-Test).

class level. To this end, we find that a good number of individual query classes from all three benchmarks reflect the benchmark-average rankings, while discrepancies on the subsequent ranks may occur. Figures 2.7 to 2.10 illustrate two query classes and corresponding retrieval results from the KN-DB, namely one class with planes and one class with swords. The charts give the effectiveness results obtained with the descriptors for these query classes. Both in the planes and swords classes, Depth Buffer scores first. While the FV performance ranking for the planes class resembles the benchmark-average ranking quite closely, for the swords class, there is significant disagreement with the global ranking on the next ranks. Specifically, the RIN and SD2 descriptors are among the top-4 descriptors in this class, while they are ranked on positions 9 and 11 in the benchmark-average ranking, respectively.

Another interesting observation on the class level can be made with respect to the Shape Spectrum descriptor. While this descriptor performs the least on benchmark average, it achieves the best retrieval result in a KN-DB query class number 55 containing models of humans, yielding on average 34% R-precision in this query class. As this descriptor considers the distribution of local curvature, it is able to retrieve human models that have different postures, while the other descriptors retrieve only those models where model posture is roughly the same (see Figure 2.11 for an illustration).

While these results are illustrative in nature, they indicate that choosing the descriptor based on benchmark average results may not be the optimal choice for answering each and every query that may be submitted to the retrieval system. Specifically, these results motivate the use of static as well as dynamic combinations of individual feature vectors, as will be further developed in Sections 2.4 and 2.5. Please refer to Table 2.3 in Section 2.4.1 for a statistic regarding the ranking of the different descriptors on the class level.

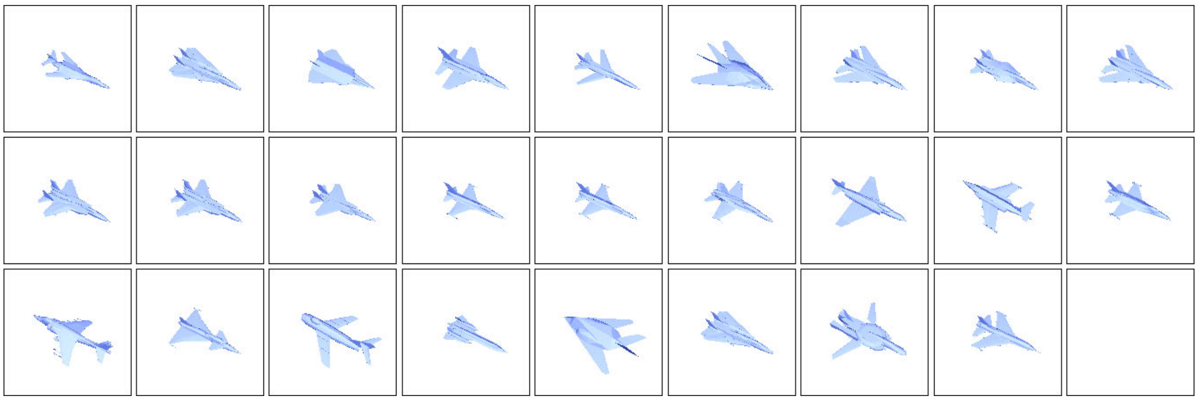
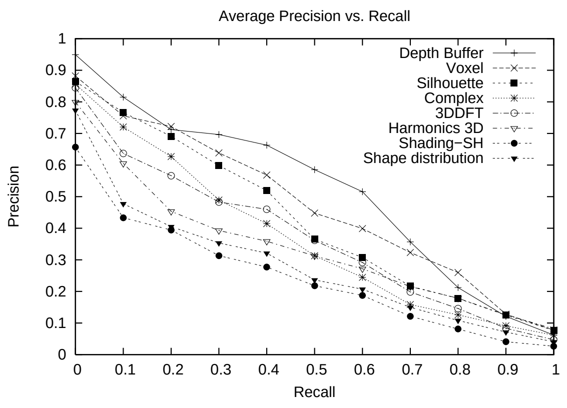
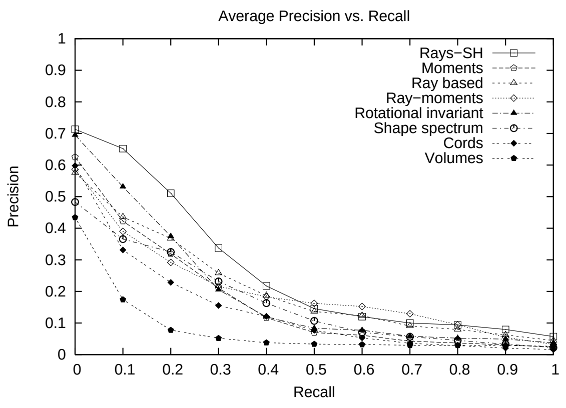


Figure 2.7: The models from the planes model class (KN-DB).



(a)



(b)

Figure 2.8: Average precision vs. recall, *planes* model class (KN-DB).

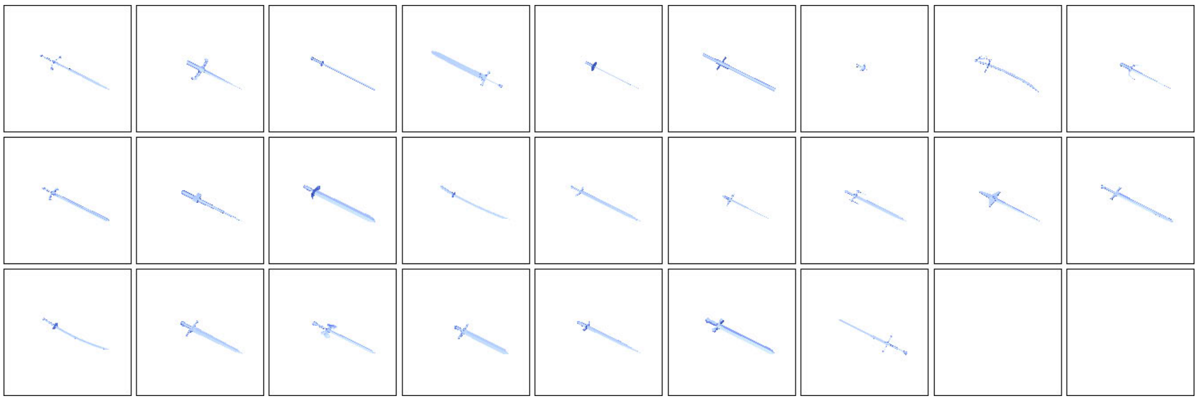
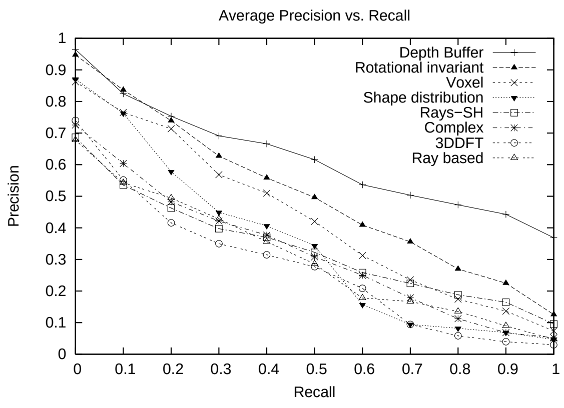
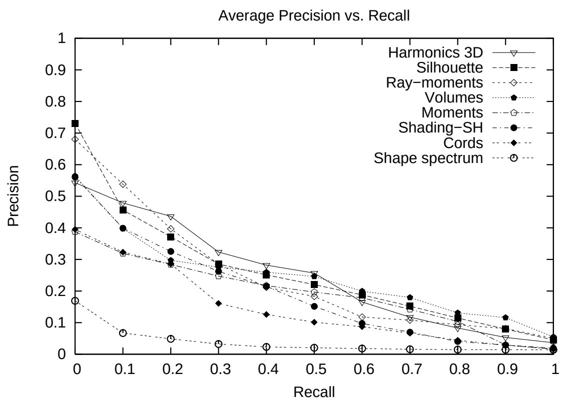


Figure 2.9: The models from the *swords* model class (KN-DB).



(a)



(b)

Figure 2.10: Average precision vs. recall, swords model class (KN-DB).

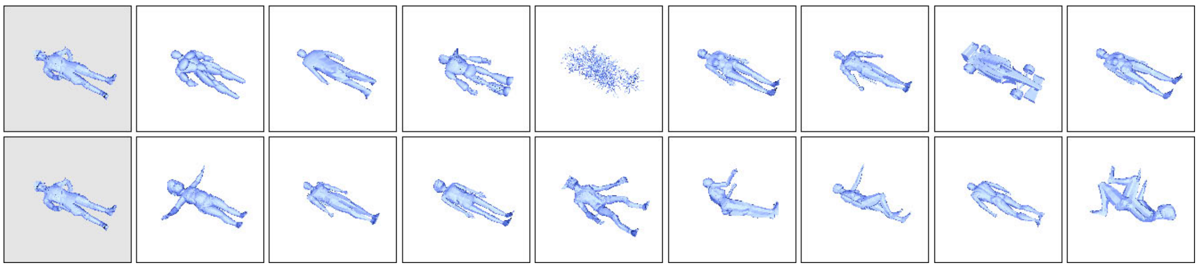
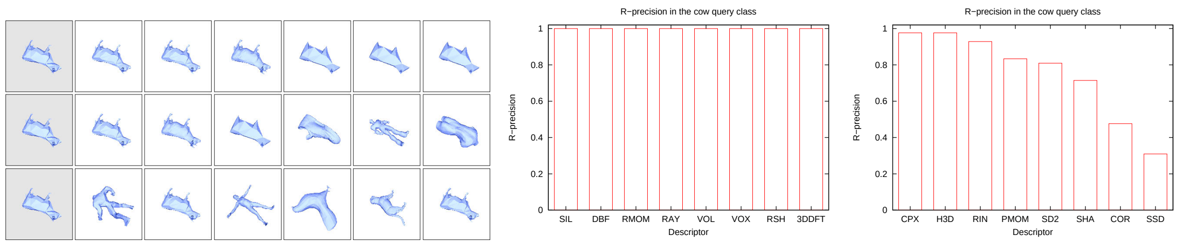


Figure 2.11: Example query in the humans class (KN-DB). The first and second rows show the eight nearest neighbors using the Depth Buffer and the Shape Spectrum FVs, respectively.

2.3.4 Sensitivities and extraction complexity

Level-of-detail

Robustness of the retrieval with respect to the level-of-detail in which models are given in a database is an important descriptor property. We test for this property using a query class from the KN-DB that contains 7 different versions of the same model, in varying levels of resolution (specifically, models of a cow with 88 up to 5,804 polygons). Except Shape Spectrum and Cords, all descriptors manage to achieve perfect or near-perfect retrieval results. Figure 2.12 (a) shows one example query in this class for 3 descriptors, and Figure 2.12 (b) gives the average R-precision numbers for all descriptors in this query class.



(a) (b) (c)

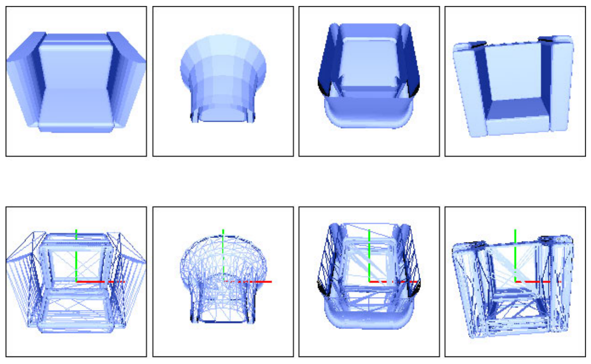
Figure 2.12: (a): Retrieval results for one example cow query object (KN-DB). The descriptors used are Depth Buffer, Cords, and Shape Spectrum from the first to the third query row, respectively. (b-c): R-precision values for the cows model class, and for all evaluated FVs.

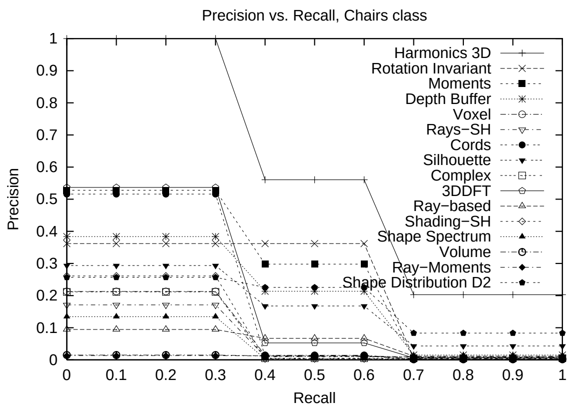
Principal axes

PCA normalization is required by most descriptor methods. For certain model classes, the PCA gives alignment results that are not in accordance with the alignment a user would intuitively expect based on semantic knowledge of the objects. For example, in the KN-DB we have defined a query class with 4 arm chairs (see Figure 2.13 (a)). In this class, PCA results are counterintuitive. While we cannot give an in-depth discussion of the PCA here, we note that in this query class an inherently rotational-invariant descriptor (harmonics 3D) provides the best class-specific retrieval performance (see Figure 2.13 (b)).

Effectiveness as a function of the dimensionality of the descriptor

It is possible to calculate FVs at different resolutions, e.g., by specifying the number of rays with which to scan the objects, by specifying the number of Fourier coefficients to consider, etc. We are therefore interested in assessing the effect of descriptor resolution on retrieval effectiveness. Figure 2.14 (a) and (b) show the effect of the descriptor dimensionality on the query-average effectiveness in the KN-DB. Figures 2.15 (a) and (b) show the same charts for the PSB-Test benchmark. Again, the descriptors' retrieval performance behaves similarly

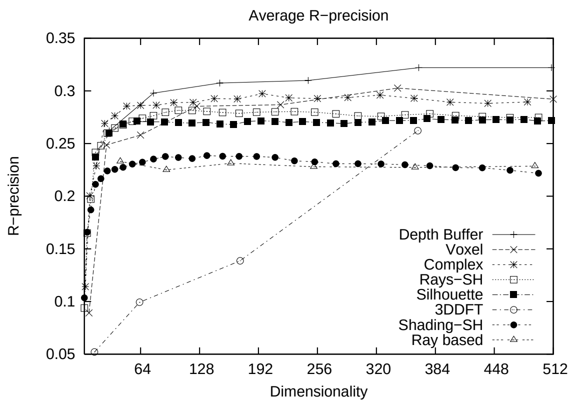

(a)



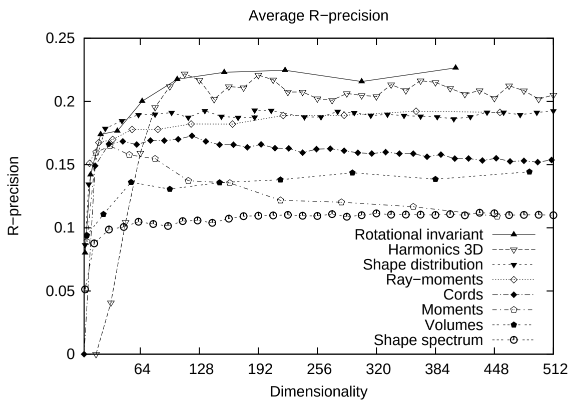
(b)

Figure 2.13: Alignment problems of PCA in some classes (a). All objects are rendered with the camera looking at the center of mass along the least important principal axis. The rotation-invariant FV Harmonics 3D shows the best retrieval performance in this query class (b).

for both benchmarks. The figures show that the precision improvements are negligible for roughly more than 64 dimensions for most FVs, which means that it is not possible to improve the effectiveness of the search system by increasing the resolution of the FV over some dimensionality. It is interesting to note that this saturation effect is reached for most descriptors at roughly the same dimensionality level. This is an unexpected result, considering that different FVs describe different characteristics of 3D objects.

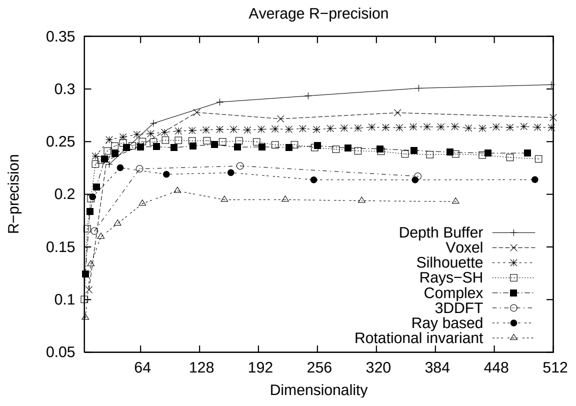


(a)

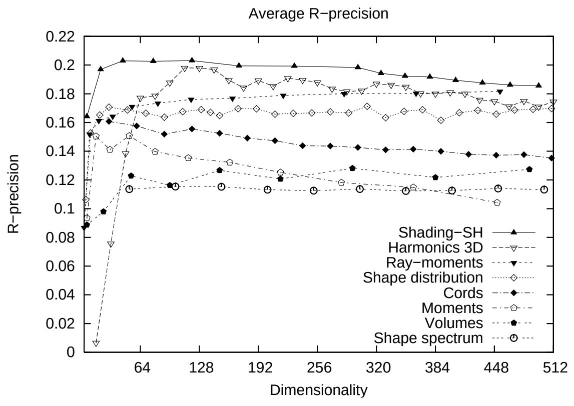


(b)

Figure 2.14: Dimensionality vs. R-precision (KN-DB)



(a)



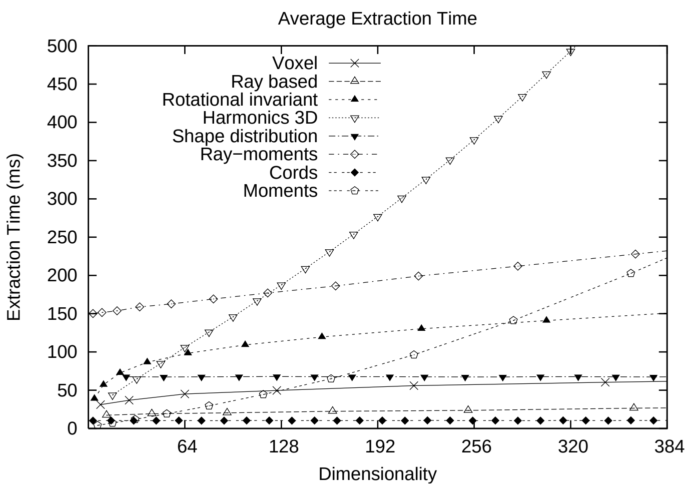
(b)

Figure 2.15: Dimensionality vs. R-precision (PSB-Test)

Computational complexity

We also compared the computational complexity of the 16 implemented descriptors. Typically, the computational cost of feature extraction is not of primary concern as extraction needs to be done only once for a database, while additional extraction must be performed only for those objects that are to be inserted into the database, or when a user submits a query object that is not yet indexed by the database. Still, computational complexity may be a bottleneck if many objects are to be imported on the fly in a productive system. We therefore present some efficiency measures taken on an Intel P4 2.4 GHz platform with 1 GB of main memory, running Microsoft Windows, when extracting FVs from the KN-DB database. We observed that in general FV extraction is quite fast for most of the methods and 3D objects. Shape Spectrum is an exception. Due to the approximation of local curvature from polygonal data by fitting of quadratic surface patches to all object polygons, this method is rather expensive. In general, PCA object preprocessing only constitutes a minor fraction of total extraction cost, as on average the PCA cost was only 3.59 seconds for the complete database of $1,838$ objects (1.95 milliseconds per object on average).

Figure 2.16 (a) shows the average extraction time per model as a function of the dimensionality of a descriptor. We did not include in this chart some of the descriptors that posses the multi-resolution property (because we computed those descriptors only once, using the maximum possible dimensionality), and we also discarded the curves for Shape Spectrum (almost constant and one order of magnitude higher than the others) and Volume (a constant value for all possible dimensions, 387 milliseconds). It follows that the extraction complexity depends on the implemented descriptor. For example, one of them has constant extraction complexity (Shape Distribution), others produce sub-linear curves (e.g., Rotation Invariant and Cords), others produce linear curves (e.g., Ray-Moments), and the rest produce super-linear curves (e.g., Harmonics3D and Moments). Figure 2.16 (b) summarizes the average extraction time per model (milliseconds) for all examined descriptors using their optimal dimensionality (cf. Table 2.2).



(a)

Feature Vector	Avg. time (ms)
Depth Buffer	249
Voxel	60
Complex	166
Rays-SH	162
Silhouette	50
3DDFT	1.545
Shading-SH	166
Ray-based	19
Rotation invariant	153
Harmonics 3D	167
Shape distribution	68
Ray-moments	228
Cords-based	10
Moments	12
Volume	388
Shape spectrum	6.439

(b)

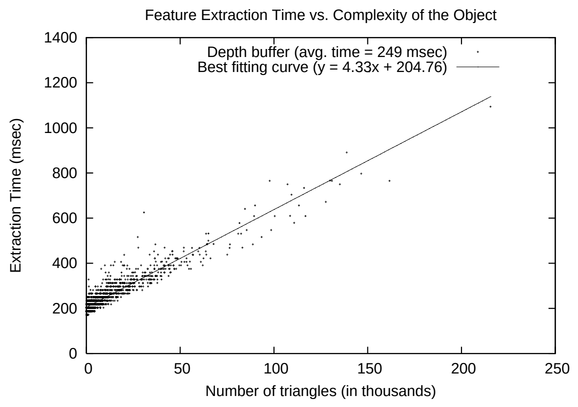
Figure 2.16: Average extraction time for some of the descriptors while varying their corresponding dimensionality (a). Descriptor computation complexity for fixed (optimal) dimensionality (b).

If the dimensionality of the descriptor is fixed, then it is possible to produce a point cloud visualizing the extraction time as a function of the number of triangles of the 3D object. Using this point cloud, we computed the best fitting linear curve by performing a linear regression. Figures 2.17 (a) and (b) show two examples of best fitting curves for the Depth Buffer and Harmonics 3D descriptors respectively, using their optimal dimensionality setting (cf. Table 2.2).

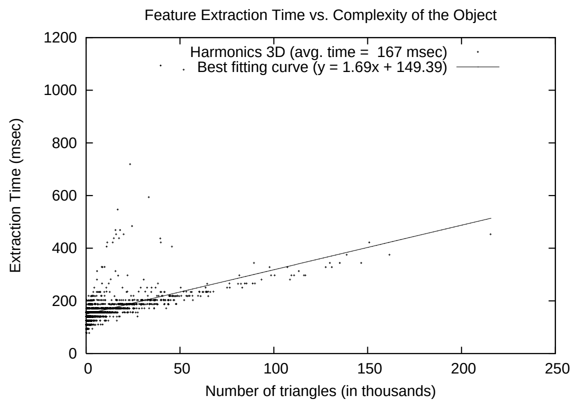
2.4 Query processing using static combinations

2.4.1 Building combinations of Feature Vectors

The retrieval performance analysis in Section 2.3 suggests that there exist a number of FVs that achieve good average retrieval performance on the majority of query classes, but that there is no clear winner among them which delivers optimal retrieval precision for all possible query classes. Instead, the individual FVs have different advantages and disadvantages. Sometimes, FVs which perform bad on benchmark average prove valuable when certain specific classes must be retrieved. E.g., the Shape Spectrum provides good retrieval results on the *humans* query class, but performs poor for most of the other benchmark classes. For many pairs of query classes, there is a significant disagreement of the FV performance rankings found. E.g., for the KN-DB classes *F-1* and *Bottles*, the respective FV rankings by retrieval precision for a number of FVs from our setup differed significantly (c.f. Figure 2.18). E.g, Depth Buffer scores 1^{st} in the F-1 class, while it performs only 4^{th} in the bottles class. As another example, the Rays-SH FV scores worst in F-1, but second best in the bottles class.

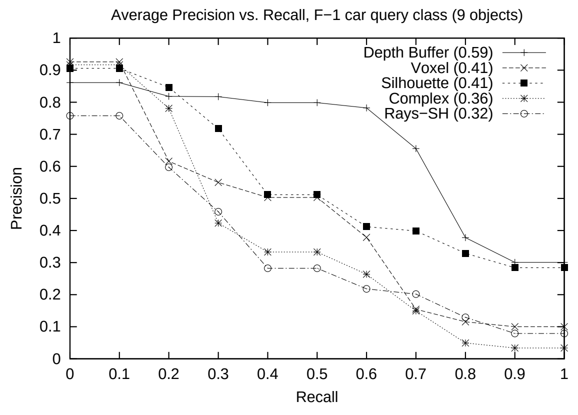


(a)

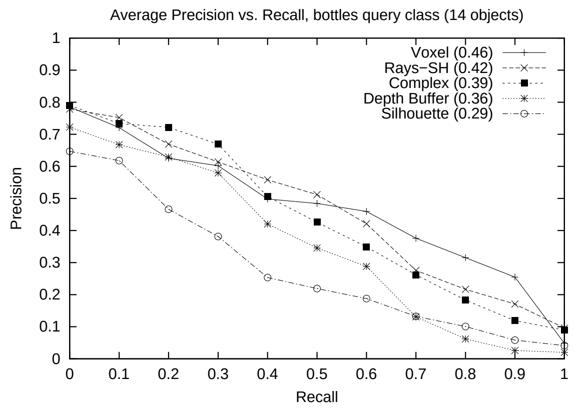


(b)

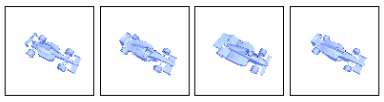
Figure 2.17: Best fitting curve for the extraction time of Depth Buffer (a) and Harmonics 3D (b).



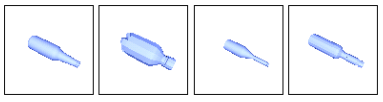
(a) F-1 results



(b) Bottle results



(c) F-1 models



(d) Bottle models

Figure 2.18: Retrieval results for the two KN-DB query classes F-1 cars and bottles. The performance rankings of the individually benchmarkes FVs do not agree for these two classes.

These class-specific ranking results are, similar to the per-class results given in Section 2.3.3, illustrative in nature. More systematically, we looked at the distribution of rankings of the methods throughout the 55 query classes of the KN-DB benchmark. Table 2.3 shows the number of query classes a FV scored in the top 1, 2, or 3 positions according to the benchmark-average R-precision results. Interestingly, the Depth Buffer FV, which achieves best retrieval performance on benchmark average, is the best method only for 12 of the query classes, while Silhouette and Voxel win in 14 and 13 classes, respectively.

Table 2.3: Number of query classes where the FVs manage to score top-*n* (KN-DB).

FV	$n = 1$	$n = 2$	$n = 3$
Silhouette	14	24	35
Voxel	13	18	26
Depth Buffer	12	22	29
Harmonics 3D	9	20	27
Ray-SH	6	22	37
Complex-SH	1	4	11

Sometimes, the suitability of a given feature extractor for retrieval of a specific query class can be theoretically estimated by the extractor definition. E.g., the Shape Spectrum descriptor performs good in the humans query class as it considers surface-relative properties which allow retrieval of models with similar surface properties but different articulations (poses) of the limbs. For many other FVs, such theoretical estimations are much harder to do. What we can note is that the considered FVs represent different, often complementary information regarding the description of 3D objects. Some of them consider surface-based information, some consider extension-based information, and still others consider information extracted from 2D projections like silhouettes. As the different FVs capture different aspects and characteristics of the models, we are interested to test whether *combining* FVs leads to improved retrieval effectiveness as compared to database-average selection of a certain FV to use by default (e.g., sticking to the Depth Buffer FV for all queries). We expect appropriate combinations of FVs to produce more robust and precise retrieval results, as we avoid the disadvantages of using just one single FV which is often not the best choice for a given query class. Combining multiple structurally different FVs leads to a more complete object description capturing more object aspects. Combining FVs can be regarded as a voting scheme, where the individual FVs contribute preferences for retrieval results (rankings) towards an aggregated ranking. Intuitively, if a minority of FVs performs bad (ranks irrelevant objects high) and a majority of FVs contributes good rankings, we expect the good FVs to compensate the bad FVs in an appropriate ranking combination scheme.

Building of combinations of FVs of different types has previously been shown to be beneficial in content-based image retrieval, e.g., in [44, 39]. Also, the combination of different classifiers using so-called *ensembles* is a technique researched for improving classification accuracy in data mining problems. So, how can different FVs be practically combined in a retrieval system? There exist two main approaches for building *unweighed* (equal-important, homogeneous) combinations of multiple FVs, which we discuss next.

Feature Concatenation. Here, the FVs are combined by simply concatenating the respective vectors to form so-called hybrid FVs. Two issues arise in this context. First, appropriate normalization of individual FVs must be considered if each FV is to exert the same (homogeneous) influence for sub-sequent distance calculations: (a) the FVs should have the same length (number of dimensions), and (b) the intervals (spread) of the component values should be comparable, if standard L_p distances are to be engaged for the distance calculation. Second, if efficiency is a concern, then the overall dimensionality of the resulting hybrid FV has to be taken care of, as concatenation of high-dimensional FVs leads of course to FVs of even higher dimensionality. Dimensionality reduction [37] is an option to reduce the concatenated FVs to reasonable sizes.

An advantage of the concatenation approach is that at query time, the resulting distance can be immediately calculated, while in the ranking-aggregation approaches, combination has to be done at query time. On the other hand, concatenated FVs lead to less flexibility in terms of weighting of individual FVs which might be desirable in a search system (cf. Section 2.5).

Result Aggregation. With this scheme, the system retains the individual FVs which are to be combined. At query time, the system produces sorted lists (rankings) for each of the considered FVs, and then forms the final query result by aggregating the individual rankings as yielded by the FVs. The aggregation can be done by several strategies. *Distance-based* aggregation produces the final ranking by calculating the distance between a query object and each database object as a sum of normalized distances as returned by each individual FV. Normalization is necessary to compensate for different scales which are possible for the individual FVs - a FV of high dimensionality and with large variance in component values would likely dominate smaller sized FVs with less variance in a sum of distances, as the resulting distances yielded by the former FV would be much higher.

Rank-based (positional) aggregation ignores distances in the rankings but considers just the ranks of objects in the answer lists returned by the individual FVs. So-called Borda-rules assign scores (votes) to the rank of each object in each ranking, and produce the final ranking by the sorted lists over the summed-up votes. The Median aggregation rule forms the final ranking by assigning each answer object the median of the ranks given to it by each of the individual FVs, a rule which has been shown to posses interesting robustness properties [31]. Many more positional aggregation methods are possible, including complex algorithms minimizing certain distance functions between individual rankings and the aggregation, or satisfying properties such as the *Condorcet* criterion known in Social Choice theory.

Distance and rank-based aggregation methods have to retain and engage the individual FVs which are to be used in the combinations. Sorted lists have to be produced and aggregated until the final result can be determined. On the other hand, these aggregation schemes are flexible to easily integrate weights to the individual FVs/ranks to be aggregated.

Concatenation of FVs has been studied in [97] for the 3D setting. We here focus on distance-based combination approaches. In section 2.4.2, we experimentally evaluate the improvement potential of selected distance-based aggregation schemes in our 3D retrieval system, and in Section 2.4.3, we consider two prominent positional aggregation schemes.

2.4.2 Results for static distance-based combinations

Combined rankings for a set F of FVs can be produced by sorting the answer objects according to the sum of normalized distances $dist_i(q,o)$ between query object q and candidate objects o, where $i \in F$:

$$dist_{combined}(q,o) = \sum_{i=1}^{|F|} \frac{1}{norm_i} dist_i(q,o)$$

The normalization factor $norm_i$ is needed to avoid that FVs which yield comparably larger distances suppress the contributions of other FVs yielding smaller distances. Several normalization schemes are possible. The *maximum* normalization uses:

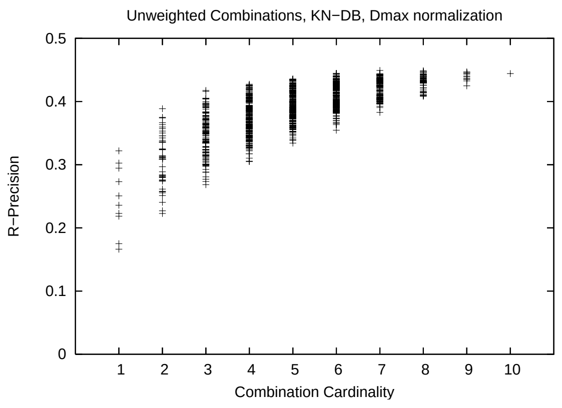
$$norm_i = \max_{o \in DB} dist_i(q,o)$$

It uses the maximum occurring distance between the query object and any candidate object. It guarantees that $dist_i \in [0,1]$ but may be susceptible to outliers: If an outlier object returns a very large distance as compared to the rest of the database objects, then most of the remaining object distances will be scaled down drastically, under-emphasizing their role in forming the combined ranking. Therefore, it is worth also including more outlier-robust normalization factors in the study. Specifically, we consider normalization factors based on *mean*, *variance*, and *median* of the distances resulting under each FV space contributing to the combination.

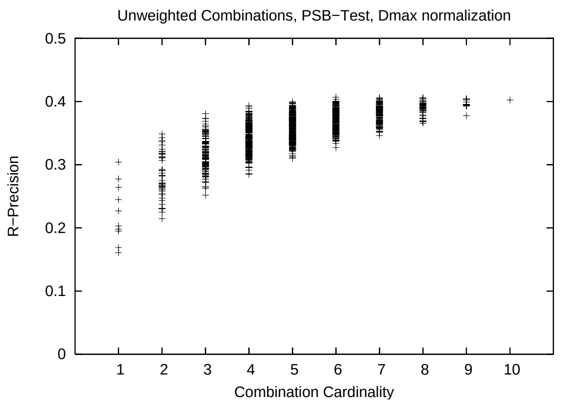
We next present retrieval precision results using fixed combinations of distance-based aggregation of multiple FVs on a number of benchmarks. The goal is to compare the performance of the combinations with the performance of single FVs. To make the experiments computationally tractable, we selected a set of 10 FVs from the available 3D FVs (cf. Section 2.3.1). For this set, we built all $2^{10} - 1 = 1023$ possible FV combinations of cardinalities 1 up to 10. Specifically, we selected the 3DDFT, COR, CPX, DBF, SHA, H3D, RIN, SD2, SIL, and VOX FVs, which belong to the best and most robust FVs from our implementation. For each possible combination, we ran batch experiments benchmarking the average retrieval performance on the KN-DB as well as the PSB-Train and Test benchmarks.

Maximum distance-based combinations

We first evaluated usage of static combinations of FVs with the d_{max} normalization factor. Figure 2.19 (a) and (b) give the resulting R-precision figures in a scatter plot for all combination cardinalities for the KN-DB and the PSB-Test benchmarks. Cardinality 1 corresponds to the usage of single FVs, and the R-precision values range between 16.6% for the worst FV



(a) KN-DB



(b) PSB-Test

Figure 2.19: Average R-precision scores for unweighed d_{max}-based combinations of up to 10 FVs, for the KN-DB and PSB-Test benchmarks.

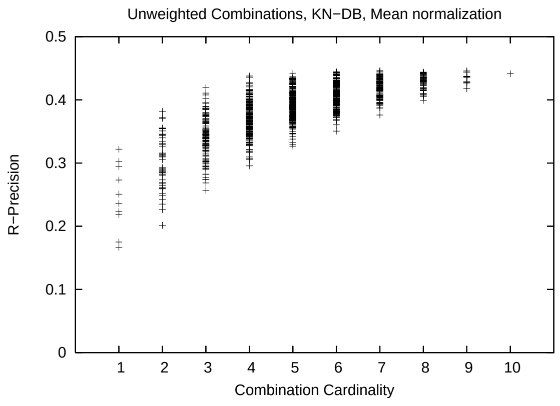
(COR), and 32.2% for the best FV (DBF) on the KN-DB benchmark. Looking at the KN-DB results, the R-precision rates indicate that there indeed exist combinations of FVs which yield a significant improvement in retrieval performance. The best combination of cardinality 2 is composed of the CPX and VOX FVs and yields 38.9% R-precision, which corresponds to an improvement over the single best FV (DBF) of 20%, which is significant. The best overall combination under the KN-DB benchmark is of cardinality 7 and compounds all FVs except for COR, RIN, and SD2 (note that these belong to the worst performing FVs on average). The combined R-precision amounts to 44.9% and corresponds to an improvement of about 39% over the single best FV (DBF). Using more than 7 FVs in any combination, the best R-precision rates start to decline. We note that the largest performance improvement leap is achieved by just 2 FVs in the combination, while inclusion of additional FVs does not impact the retrieval quality as much as going from single FV usage to the best combination of cardinality 2. The results under the PSB-Test benchmark are a few percentage points below the results of the KN-DB benchmark, but resemble the same performance pattern.

Table 2.4 gives the maximum, minimum, and average R-precision scores for the different combination cardinalities for the KN-DB and the PSB-Test benchmarks. Interestingly, starting with combination cardinality of 5, for both benchmarks even the worst performing combinations yield a retrieval precision which is better than the best performing single FV (DBF). This is true for both benchmarks. This is interesting because it is not always possible to perform supervised benchmarks to determine the best combination to implement in a practical retrieval system. The results indicate that there are combination cardinalities for which any combination outperforms all single FVs, thereby dominating in terms of achievable retrieval quality. It means that even if we randomly select one specific combination of sufficient cardinality, we are guaranteed to improve in terms of retrieval precision, as compared to the single FV usage case.

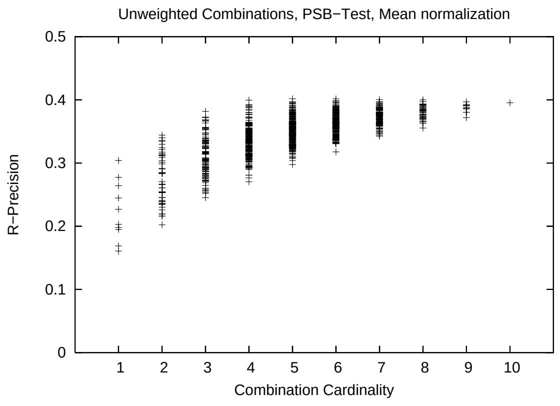
Table 2.4: R-precision statistics for the d_{max} normalized combinations.

Benchmark		1	2	3	4	5	6	7	8	9	10
KN-DB	min	16.6	22.3	26.8	30.5	33.4	35.5	38.3	40.8	42.5	44.4
KN-DB	mean	24.6	31.0	34.7	37.3	39.4	40.9	42.1	43.1	43.9	44.4
KN-DB	max	32.2	38.9	41.8	42.8	43.6	44.4	44.9	44.9	44.7	44.4
PSB-Test	min	16.1	21.5	25.2	28.5	31.0	32.7	34.6	36.6	37.7	40.2
PSB-Test	mean	22.4	28.2	31.7	34.1	35.8	37.0	38.0	38.8	39.6	40.2
PSB-Test	max	30.4	34.9	38.1	39.3	40.0	40.7	40.6	40.6	40.5	40.2

Mean-based combinations



(a) KN-DB



(b) PSB-Test

Figure 2.20: Average R-precision scores for unweighed mean-based combinations of up to 10 FVs, for the KN-DB and PSB-Test benchmarks.

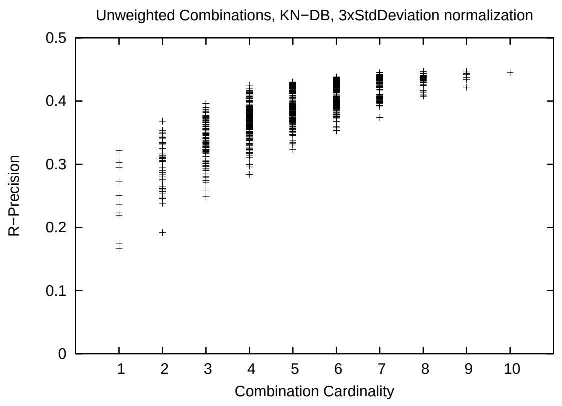
Instead of d_{max}, normalization of distances is possible using the *average* distance between the query object and all the ranked candidate objects. This factor is expected to be less susceptible to the presence of outlier objects with very large distance to the query object. Figure 2.20 presents the scatter plots of the achieved R-precision values for the combinations under the KN-DB and PSB-Test benchmarks. The performance characteristics are qualitatively and quantitatively comparable to the d_{max} normalization (cf. also Table 2.5). Considering the best combinations possible, there is a maximal performance leap improving from the best single FV (DBF, 32.2%) to 38.1% R-precision for two FVs (CPX and VOX), to 42% for three FVs, and reaching a peak performance of 44.5% for 6 FVs, before peak performance starts to drop. The peak performance is slightly worse than under d_{max} normalization (44.6% (7 FVs) compared to 44.9% (7 FVs) under the KN-DB – for PSB-Test the same situation holds. Also, for 5 and more FVs, the worst performing combinations consistently outperforms the single best FV (DBF), making it safe to use any combination of such cardinalities and still improve over the DBF single usage mode.

While the overall performance pattern of mean normalization tightly resembles the d_{max} normalization, the absolute R-precision figures are somewhat smaller, loosing between 1.0 and 0.5 percentage points for the min, max, and mean R-precision scores for most of the combinations on both benchmarks. An exception is the weakest performing combination which is 2 percentage points weaker than the d_{max}-scheme. Contrarily, the optimal combinations of cardinality 4 and 5 are about 1.0 and 0.5 percentage points better than under the d_{max} scheme (both benchmarks). Our general impression is that as judged by the extreme and the mean R-precision rates, there is no significant and systematic difference between the performance of both normalization schemes (cf. also Section 2.4.4 which includes the distribution of individual combination differences).

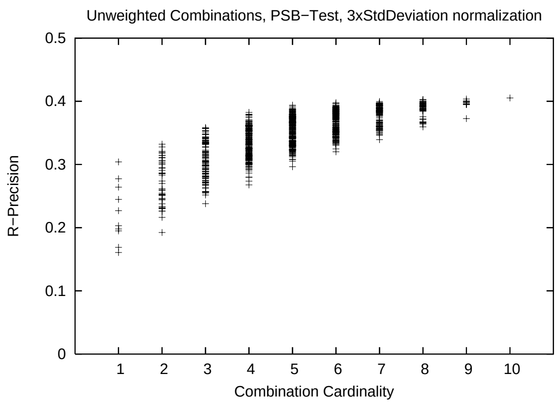
Table 2.5: R-precision statistics for the mean normalized combinations.

Benchmark		1	2	3	4	5	6	7	8	9	10
KN-DB	min	16.6	20.1	25.7	29.6	32.7	35.0	37.6	39.9	41.8	44.1
KN-DB	mean	24.6	30.1	34.3	37.1	39.2	40.7	41.9	42.8	43.4	44.1
KN-DB	max	32.2	38.1	41.9	43.8	44.2	44.4	44.6	44.3	44.6	44.1
PSB-Test	min	16.1	20.2	24.5	27.0	29.8	31.8	34.3	35.5	37.2	39.5
PSB-Test	mean	22.4	27.6	31.0	33.4	35.1	36.3	37.3	38.0	38.6	39.5
PSB-Test	max	30.4	34.4	38.2	39.9	40.2	40.2	40.0	40.0	39.7	39.5

Variance-based combinations



(a) KN-DB



(b) PSB-Test

Figure 2.21: Average R-precision scores for unweighed 3-standard variance-based combinations of up to 10 FVs, for the KN-DB and PSB-Test benchmarks.

Besides d_{max} and mean normalization, we can normalize by a factor based on the *variance* of the distances returned by each FV to be combined. In one experiment, we set the normalization factor $norm_i = 3 \times \sigma_i(q, DB)$, where $\sigma_i(q, DB)$ is the standard variance of the distances between query object q and all candidate objects from the database DB under the FV i. This is a rule-of-thumb often used in data normalization preprocessing. While it does not bound the maximum normalized distance, it is expected to be more robust with respect to outliers.

Figure 2.21 gives the scatter plot of R-precision results achieved by the variance-normalized combinations, and Table 2.6 gives the extreme and the mean R-precision results achieved. Basically, the same performance pattern like for d_{max} and mean normalization is given. The combinations improve significantly over the single usage of FVs, with a performance peak at about 6 to 7 FVs in the combinations. The largest performance leap occurs when going from the DBF single FV to a combination of DBF and CPX. With 5 (6) FVs, the worst performing combination still outperforms the DBF single FV on the KN-DB (PSB-Test) benchmark.

The overall performance readings are again somewhat below the results yielded by d_{max} aggregation. Minimum and maximum performance results for smaller cardinalities are between 1 and 3 percentage points below the d_{max} results, while the performance gap narrows for larger cardinalities, and the mean results. Generally, the performance results are also slightly below the results achieved by the mean normalization scheme.

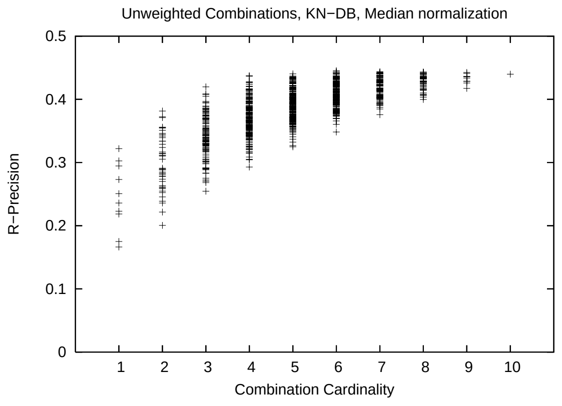
Table 2.6: R-precision statistics for the 3x standard deviation normalized combinations.

Benchmark		1	2	3	4	5	6	7	8	9	10
KN-DB	min	16.6	19.2	24.9	28.4	32.3	35.3	37.4	40.8	42.2	44.5
KN-DB	mean	24.6	29.7	33.9	36.9	39.1	40.8	42.1	43.2	44.0	44.5
KN-DB	max	32.2	36.8	39.6	42.5	43.2	43.8	44.5	44.7	44.7	44.5
PSB-Test	min	16.1	19.2	23.8	26.8	29.6	32.0	33.9	36.0	37.3	40.5
PSB-Test	mean	22.4	27.1	30.7	33.3	35.2	36.7	37.8	38.7	39.8	40.5
PSB-Test	max	30.4	33.2	35.9	38.2	39.4	39.8	40.0	40.3	40.4	40.5

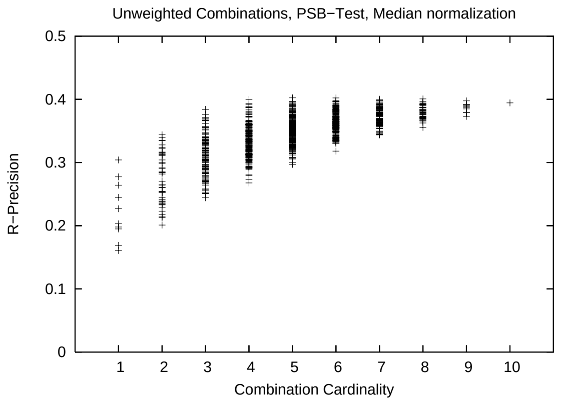
Median-based combinations

We finally evaluated the results of normalizing the distances by the median of all distances occurring for a given query objects under a given FV space. The median is examined as another outlier-robust normalization scale often used in other data processing applications. Figure 2.22 and Table 2.7 present the R-precision results obtained by the batch experiments.

The overall performance pattern again resembles the three previous normalization schemes: Significant improvements over the single best FV, decreasing performance leaps with a saturation cardinality, and certain minimum cardinalities after which even the worst combinations outperform the best single FV. The absolute performance values are slightly below the d_{max} normalization results, and on a level comparable to the mean normalization, and better than the variance-based normalization. Also, for larger cardinalities, the performance detriment regarding d_{max} is decreasing.



(a) KN-DB



(b) PSB-Test

Figure 2.22: Average R-precision scores for unweighed median-based combinations of up to 10 FVs, for the KN-DB and PSB-Test benchmarks.

Table 2.7: R-precision statistics for the median normalized combinations.

Benchmark		1	2	3	4	5	6	7	8	9	10
KN-DB	min	16.6	20.1	25.5	29.3	32.5	34.8	37.6	40.0	41.7	44.0
KN-DB	mean	24.6	30.0	34.2	37.1	39.1	40.7	41.8	42.6	43.4	44.0
KN-DB	max	32.2	38.2	42.0	43.7	44.0	44.5	44.4	44.3	44.2	44.0
PSB-Test	min	16.1	20.1	24.4	26.8	29.7	31.8	34.4	35.6	37.3	39.4
PSB-Test	mean	22.4	27.5	31.0	33.3	35.0	36.3	37.2	38.0	38.6	39.4
PSB-Test	max	30.4	34.4	38.4	40.0	40.2	40.2	40.0	40.1	39.8	39.4

2.4.3 Results for static rank-based combinations

It is also possible to build combinations based only on the *ranks* which are given to the candidate objects by the FVs in the combination. Rank-based aggregation considers only the positions of candidate elements in the rankings and ignores any numeric scales which might be present. The advantage of this scheme is that it requires less information for building combined rankings. Rank aggregation is popular for building meta search engines for the web [31]. These aggregate the sorted lists retrieved from different search engines into a combined list of web pages. The combination algorithm does not need to know internals of the different search engines but generically operates on the outputted lists, considering the different search engines as black boxes. On the other hand, relying just on ranks ignores information. There is no chance to exploit information on how close or distant two adjacently ranked objects are, which is possible using distance-based aggregation. We compare the previous results by considering two rank-based aggregation schemes.

The *Borda Aggregation* (or voting) scheme transforms rank positions into votes which are sub-sequently summed up in order to obtain the final ranking based on these votes. The dependency between rank and number of votes awarded has to be specified. In the most simple case, for a ranking $R = [o_{(1)}, \ldots, o_{(n)}]$ of n objects $o \in O$, we award $v(i) = n - i$ votes for object $o_{(i)}$ at rank i, $i \in [1, \ldots, n]$. This scheme realizes a linear dependency between rank and number of votes; also, non-linear schemes are possible.

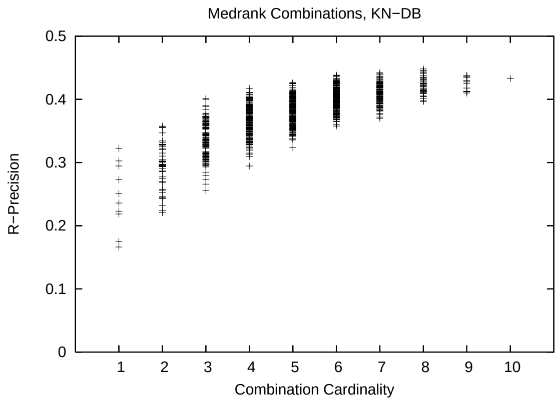
The *median Aggregation* scheme is another simple but interesting aggregation scheme. It forms the aggregated ranking by sorting the candidate objects according to the median of the ranks given to them by each of the FVs contributing to the combination. Median rank aggregation has been shown to approximate the optimal aggregation solution based on the *Footrule* rank distance measure, and to provide robust aggregation [31]. Also, an efficient implementation exists for median rank aggregation, called Medrank [32].

Medrank combinations

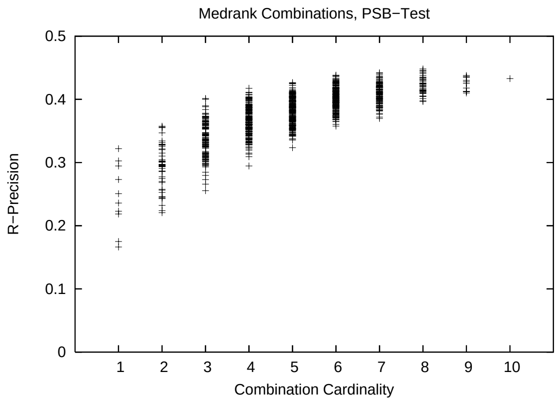
Figure 2.23 and Table 2.8 give the R-precision results for the Medrank-based combinations. The overall performance pattern considering the extreme and the average R-precision resembles that of the distance-based aggregation: Significant improvements over the single usage mode are achieved. The optimal R-precision (KN-DB) is reached with 8 FVs (all except for RIN and SD2) and amounts to 44.8%, which is only 0.1 percentage point worse than the optimal d_{max}-based combination (7 FVs yielding 44.9% R-precision). Other than this peak performance, the cardinality-dependent minimum, average, and maximum performance readings are consistently below the d_{max}-based results, yielding each about 1 to 2 percentage points less R-precision than the d_{max} scheme.

Borda combinations

We finally tested the simple Borda aggregation scheme, awarding the candidate objects votes inversely proportional to their ranks as given under the individual FV spaces. The final ranking is produced by summing up the votes for all candidate objects and sorting the final list. Figure 2.24 and Table 2.9 present the obained R-precision scores. Once again, combinations pay



(a) KN-DB



(b) PSB-Test

Figure 2.23: Average R-precision scores for Medrank-based combinations of up to 10 FVs, for the KN-DB and PSB-Test benchmarks.

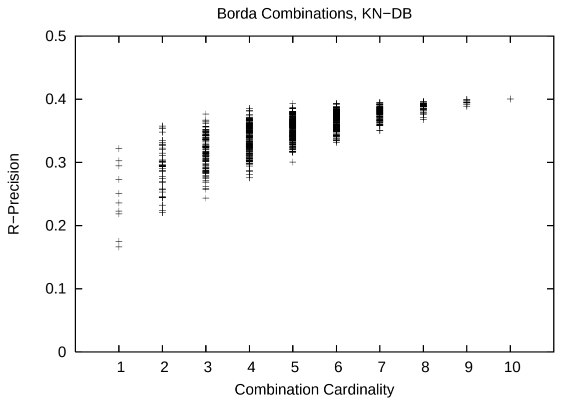
Table 2.8: R-precision statistics for the Medrank-based combinations.

Benchmark		1	2	3	4	5	6	7	8	9	10
KN-DB	min	16.6	22.1	25.5	29.5	32.3	35.8	37.0	39.7	41.0	43.3
KN-DB	mean	24.6	29.3	33.7	36.5	38.2	40.1	40.9	42.2	42.4	43.3
KN-DB	max	32.2	35.8	40.1	41.8	42.6	43.8	44.2	44.8	43.8	43.3
PSB-Test	min	16.1	19.8	23.3	26.5	29.1	31.8	33.3	35.8	37.1	39.2
PSB-Test	mean	22.4	26.3	29.9	32.6	34.0	35.9	36.4	37.8	38.1	39.2
PSB-Test	max	30.2	32.1	36.3	37.3	38.3	38.8	39.9	40.0	39.2	39.2

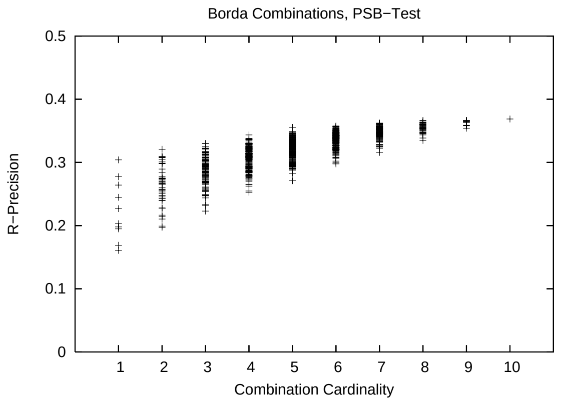
Table 2.9: R-precision statistics for the borda rank-based combinations.

Benchmark		1	2	3	4	5	6	7	8	9	10
KN-DB	min	16.6	22.1	24.4	27.6	30.0	33.2	35.0	36.8	38.9	40.0
KN-DB	mean	24.6	29.3	31.8	33.8	35.3	36.7	37.8	38.7	39.5	40.0
KN-DB	max	32.2	35.7	37.6	38.5	39.3	39.3	39.5	39.6	40.0	40.0
PSB-Test	min	16.1	19.7	22.3	25.3	27.1	29.8	31.6	33.5	35.4	36.9
PSB-Test	mean	22.4	26.3	28.8	30.6	32.2	33.5	34.6	35.5	36.3	36.9
PSB-Test	max	30.4	32.1	33.0	34.3	35.5	35.8	36.2	36.7	36.7	36.9

off by improved retrieval precision. Comparing with the previous combination schemes, we see that the R-precision statistics are significantly below the Medrank and the distance-based aggregation schemes. The minimum, average, and maximum performance for each combination cardinality is between 3 to 5 percentage points below the respective results achieved by the d_{max} scheme, which is a significant loss in precision. Also, we see that there is no saturation point reached for the best-performing combinations in the combination cardinalities considered here.



(a) KN-DB



(b) PSB-Test

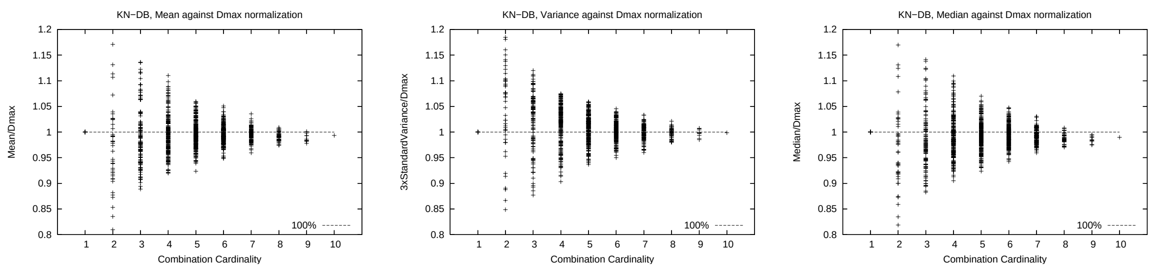
Figure 2.24: Average R-precision scores for Borda-based combinations of up to 10 FVs, for the KN-DB and PSB-Test benchmarks.

2.4.4 Analysis of the results

The presented results are now analyzed for insight into how to build good static combinations. We compare the effect of normalization methods in distance-based combinations, and aggregation schemes in rank-based aggregation. The combination structure is analyzed to give recommendation for cases where optimal combinations cannot be determined by exhaustive benchmarking. Stability with respect to the different benchmarks is discussed. Finally, the results are summarized.

Performance assessment of distance-normalization methods

For the distance-based aggregation approach, we have tried different distance normalization schemes based on maximum, mean, median, and variance-based scaling factors. A-priori it is not clear which normalization should be engaged, as all of them can be theoretically justified. Regarding the comparison of the lowest, highest, and average R-precision results achieved by the different combinations, surprisingly the d_{max}-based normalization, which in principle is susceptible to outlier distances, dominated the other normalization schemes for almost all combination cardinalities in terms of the best, the worst, and the average performing combinations. The mean and median-based normalization schemes performed comparably and slightly worse than d_{max}, and the variance-based scheme yielded the lowest retrieval performance results. We note that these comparisons are done for an aggregation of performance results, namely, the extreme and average R-precision results aggregated over all combinations of a given cardinality.



(a) Mean/d_{max} (b) Variance/d_{max} (c) Median/d_{max}

Figure 2.25: Average R-precision on the KN-DB benchmark of mean, variance, and median normalized unweighed combinations in fraction of respective performance of d_{max} normalized combination. There is no significant difference in the performance of the normalization methods.

If we instead compare the normalization schemes along all possible combinations, we observe that d_{max} is not superior for all the combinations. Rather, there exist combinations where one or several of the other normalization schemes outperform d_{max}. Figure 2.25 shows a scatter plot of the mean, variance, and median-based combinations' performance relative to the d_{max} scores, under the KN-DB benchmark. Clearly, about half of the combinations for each normalization scheme actually *improves* over d_{max}. But it is the total effect of all improvements/deteriorations which leads to the aggregate results found previously: While about half of the combinations improve over d_{max}, the rate of improvement is outweighed by the rate of deterioration occurring for the other half of combinations.

We can practically use these findings in the following ways. If benchmarking is possible and we are free to form any combination with respect to cardinality and contributing FVs, then we will select the optimum combination using the d_{max} normalization. If we are restricted in the usage of FVs, meaning that it is not possible to form any combination of FVs available for some reason, and exhaustive benchmarking is possible, then we may find one of the other normalization schemes to provide superior results, and consequently, use this combination. If exhaustive benchmarking is not possible, then we recommend to default to the d_{max} aggregation scheme, using a combination of large cardinality, expecting to yield best performance on average.

Performance assessment of rank aggregation methods

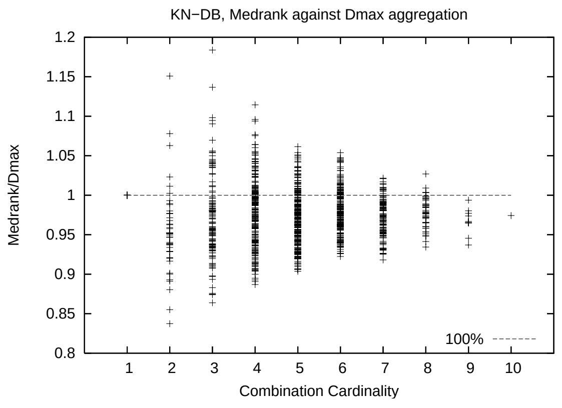
The rank-aggregation schemes Borda and Medrank were found to be outperformed by the d_{max}-based distance aggregation scheme for the minimum, maximum, and average benchmark scores for any combination cardinality. The Medrank scheme performs comparable to the mean and variance-based distance aggregation schemes. The linear Borda scheme performed worst under this analysis, being consistently outperformed between 3 and 5 percentage points in R-precision by d_{max}.

If we consider the rank aggregation performance results relative to the d_{max} scheme (cf. Figure 2.26), we see that contrary to the situation for the distance-based schemes, there are cardinalities where most or all individual combinations are outperformed by d_{max} aggregation. This is especially evident for the Borda scheme (cf. Figure 2.26 (b)), where the mass of combinations yields only around 90% of the respective d_{max} performance. For cardinalities 2 and 3, a minority of Borda combinations improves over d_{max}, while starting with cardinality 4, except for one outlier all Borda combinations are outperformed by d_{max}. For Medrank, the comparison with d_{max} is not as bad as for Borda, still the trend is that for the larger-sized combinations (6 FVs and above), the mass of combinations is outperformed by d_{max}.

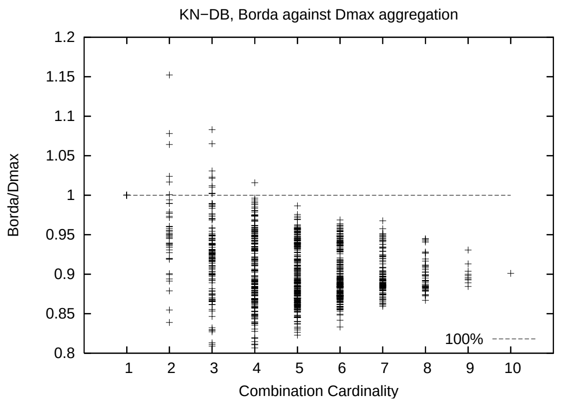
We conclude that if we are free to form any combination of FVs, and exhaustive benchmarking is possible, we will choose the d_{max} scheme, as it outperforms the optimal Borda and Medrank combinations. If we are restricted in the combination of FVs to select, depending on the restriction it might be the case that Medrank results in best performance (probably for low cardinalities), and to a lesser degree of probability, the Borda scheme (up to cardinalities 3). If exhaustive benchmarking is not possible, then we recommend using the d_{max} combination on a large number of FVs as default.

We attribute the loss in performance by the rank aggregation schemes to the fact that distance information is suppressed, while distance actually carries meaningful information which the distance-based schemes manage to exploit. Nevertheless, positional aggregation schemes have advantages. Positional aggregation must be used in cases where distance information is not available (such as when building a meta search engine treating individual search algorithms as black boxes). Also, an efficient implementation of Medrank is available [32], while efficiently implementing query processing using multiple FVs is not straightforward and involves building of complex index structures [20].

We finally note that Medrank has been attributed so-called spam-fighting abilities, referring to its robustness with respect to outlier rankings giving undeserved high ranks to certain candidate objects (as judged by a majority of additional rankings) [31]. This is a useful property when combining rankings for a World Wide Web meta search engine, where individual engines might be spammed by manipulated content. We tested this property by considering all FV combinations which included the worst performing single FV (COR) from our selection of FVs. If COR is actually acting as a spammer, Medrank could be expected to give better retrieval results than other non spam-resistent aggregation schemes for the combinations including COR. We looked at the R-precision results for these combinations but could not find any systematic improvement of Medrank on these combinations over the d_{max} scheme. This does not, however, imply that Medrank does not have anti-spam properties, but most probably that COR cannot be regarded a spammer (its retrieval results are not arbitrary enough). A more systematic experiment testing the anti-spam property of Medrank in this retrieval setup would have to involve systematically perturbing/noising a small number of FV rankings.



(a) Medrank/d_{max}



(b) Borda/d_{max}

Figure 2.26: Average R-precision on the KN-DB benchmark of Medrank and Borda aggregation, as a fraction of performance of d_{max} normalized combinations.

Role of the combination structure

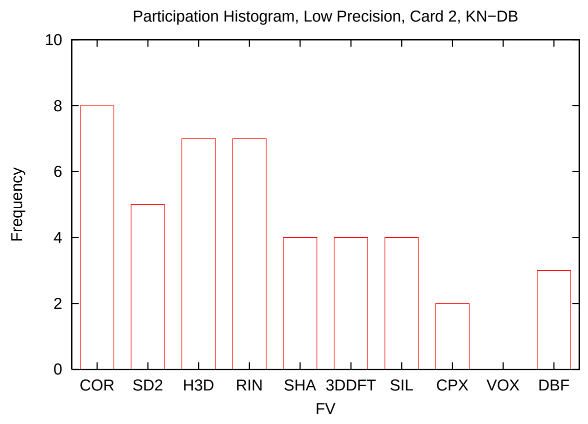
Given that for each combination cardinality, we have many options for combining subsets of the available FVs, and given that we observed a bandwidth of resulting retrieval precision rates, it is interesting to look at the structure of the combinations. Are there specific characteristics that make up good combinations? To this end, we constructed *participation histograms* for the combinations. For the set of all possible combinations of a fixed cardinality, we partition the combinations into two groups *low precision* and *high precision*, according to the average R-precision scores resulting for the combinations (low contains the lowest 50%, high contains the highest 50% of combinations). Then, for both groups and for each of the 10 FVs, we calculate the respective participation frequencies. For a group and a FV, the participation frequency gives the number of combinations in which the considered FV participates.

Figure 2.27 shows the participation histograms for combinations of cardinality 2 (first row) and 4 (second row). The histograms were calculated for d_{max} normalized combinations under the KN-DB benchmark. The left charts give the participation histograms for the low performing combinations, while the right charts give the histograms for the high performing combinations. The FVs were sorted by increasing single benchmark scores. We observe that the histograms of the low performance combinations are skewed to the left, implying that the badly performing FVs, according to single benchmark scores, are participating more frequently in bad-performing combinations than in high-performing combinations. Conversely, in the group of good combinations, the histograms are skewed to the right, meaning that these combinations consist more frequently of FVs which also perform good as benchmarked individually. A visualization of the participation histograms for the cardinalities 2 and 3 for the KN-DB benchmark is included in the Appendix Figures A.4 and A.5.

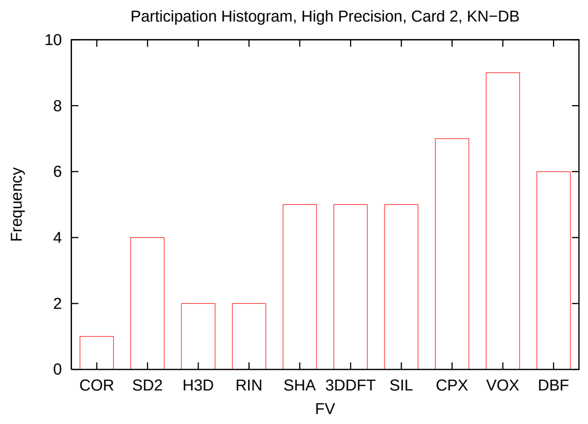
We observed this dependency clearly for participation histograms for combination cardinalities of small up to medium size (up to about cardinality 6). For higher cardinalities 7 to 9, the skewness effect was also present, but diminished (the corresponding histograms approach a uniform-like shape). This can be attributed to the fact that for increasing cardinalities and a fixed number of FVs, each FV participates more often in a combination, thereby leveling-out the skewness effect. This becomes clear if we consider the case of 10 FVs and the 10 possible combinations of cardinality 9: Each FV participates in 90% of all possible combinations, and the corresponding participation histograms are approximately uniform. We also observed the skewness effect for the other benchmarks PSB-Train and Test, as well as the other distance-based combination schemes. Taking together these observations, we conclude that FVs which perform good in single usage are also good candidates for high-performance combinations. As a rule-of-thumb for cases where exhaustive benchmarking is not possible to determine the optimal combination, we therefore recommend to combine a set of best performing single FVs.

Sensitivity regarding the benchmark

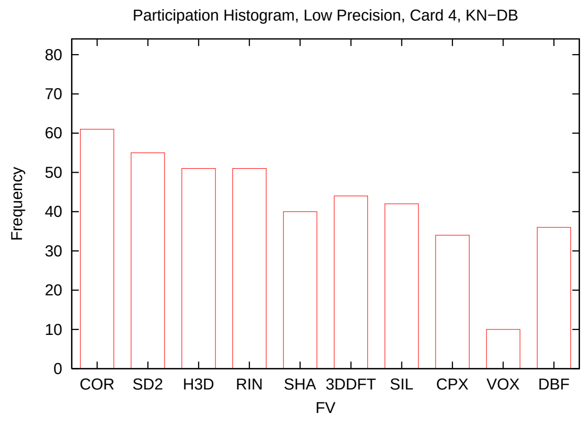
We also asked whether the performance results of the combinations are dependent on the 3D benchmark used. Considering that benchmarking is expensive, and database content may dynamically evolve away from the benchmark characteristics for which a combination was initially optimized, it is interesting to assess whether the retrieval performance of a specific combination remains stable if database content changes. To approach this question, we studied the correlation between R-precision results for the combinations under the three 3D benchmark used here. Figure 2.28 gives the scatter plots of observed R-precision values for the 1023 possible combinations of the 10 FVs, for all three pairs of benchmarks, and using the d_{max}-based aggregation. There is a significant linear correlation between the R-precision results of the combinations for all pairs of benchmarks. The squared correlation coefficients amount to more than 93%. We conclude that the combinations' retrieval precision rates are stable over the three benchmarks, implying that we can safely determine the optimal combination using any of the three benchmarks, expecting to achieve the optimal or near-optimal combination result even in case the target database (application) is not identical to the benchmark. This conclusion is valid for the three benchmarks considered here, which are build over a comparable distribution of 3D models (cf. Section 2.3.2). We note that the correlation analysis also includes the cardinality 1 combinations, which correspond to the single usage of FVs. Here also the correlation is clear, which indicates benchmark-based selection of single FVs



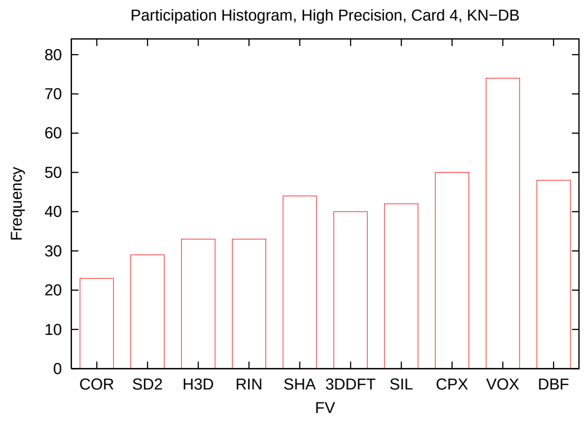
(a) Card. 2, Low Precision Combinations



(b) Card. 2, High Precision Combinations



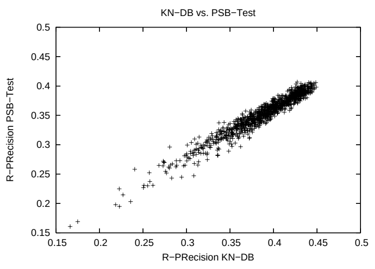
(c) Card. 4, Low Precision Combinations



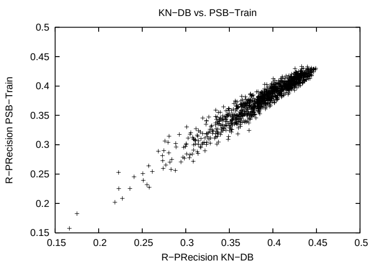
(d) Card. 4, High Precision Combinations

Figure 2.27: Participation histograms of the 10 FVs for combination cardinalities 2 (a,b) and 4 (c,d). The FVs are sorted by increasing single retrieval performance.

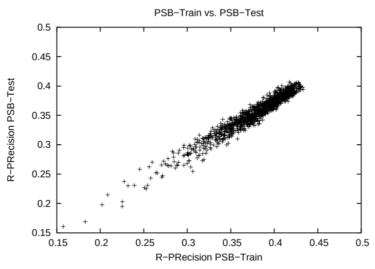
is stable with respect to different benchmarks. Similar results are also obtained for the other distance-based aggregation schemes.



(a) KN-DB/PSB-Test (94.0%)



(b) KN-DB/PSB-Train (92.9%)



(c) PSB-Train/PSB-Test (94.3%)

Figure 2.28: R-precision results of the combinations under the three considered benchmarks KN-DB, PSB-Train and PSB-Test, plotted pairwise against each other (R^2 measures are included in brackets).

2.4.5 Summary and practical recommendations

We summarize the main findings of the experiments on unweighed combinations as follows.

- Static combinations of FVs are a simple, effective approach to improve the average retrieval precision in our 3D retrieval system. Improvements of up to 40% in R-precision have been observed on a set of 10 rather differently defined single 3D FVs, and on several 3D benchmarks, as compared to the single-best individual FV.

- The improvement rates are most significant when switching from single FV usage to combinations of small cardinalities, i.e., 2 or 3. This is interesting, as the combination size also affects the efficiency of query processing, where in general, larger-sized combinations lead to higher computational costs for distance calculation or rank aggregation.

- Further improvement potential exists for larger combinations, but diminishes rather quickly. For large-sized combinations (7 FVs and above), there is danger that retrieval precision is eventually starting to decrease.

- Care has to be taken in determining the FVs to participate in a combination. Exhaustive benchmarking allows to find the optimal combination for a given cardinality.

- The results suggest there are cardinalities where even the worst performing combination outperform the best performing single FV. This is interesting as it supports the idea of using combinations by default even if it is not possible to benchmark all possible combinations by exhaustive experiments.

- Regarding the structure of the FV combinations, the participation histograms showed that FVs which provide good retrieval precision in single mode are also good candidates for inclusion in a combination.

- We found that distance-based aggregation in general gives better retrieval results than rank-based aggregation. We attribute this to the fact that distance-based aggregation employs more information about the individual rankings, as purely positional aggregation schemes do.

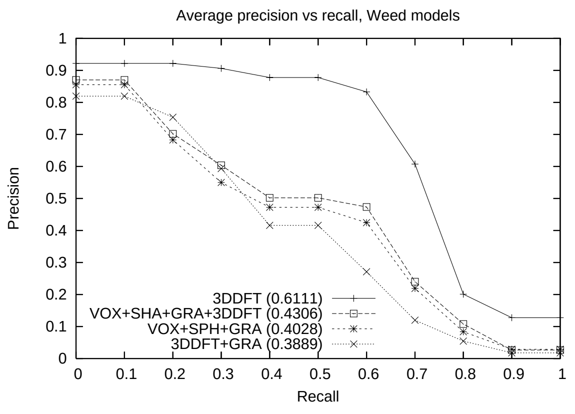
- The above discussion was done for a set of 10 different 3D FVs. We note that we have done the same set of experiments for a database of raster images (cf. Section 1.4). This database contains 6.000 classified COREL images described by 6 different image-based FVs based on features such as color and texture [75]. The corresponding results closely resemble those obtained using the 3D benchmarks. Figures A.2 and A.3, as well as Table A.2 in the Appendix present the corresponding results.

2.5 Query processing using dynamic combinations

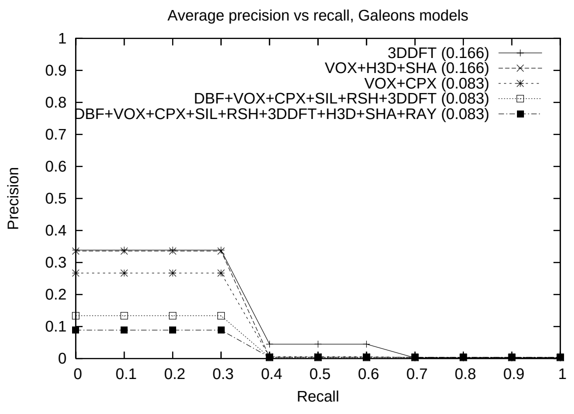
2.5.1 Query-dependent analysis of static combinations

In Section 2.3, we determined the on average best performing single FV (the Depth Buffer FV). In Section 2.4, we have seen that by building unweighed combinations, the retrieval precision could be significantly boosted, achieving about 40% improvements in average retrieval precision. Similar to the single FV case, benchmarking allowed to find the *on benchmark average* best combination of FVs. It turns out that the same phenomenon which originally motivated the usage of combinations, namely that the on average best method is not necessarily the optimal choice for processing every possible query class or object, also holds for the case of static combinations. Consider Figure 2.29, which shows the KN-DB benchmark results for the optimal single FV and combinations for two specific query classes, weed and galleons. In these classes, the single 3DDFT FV yields best retrieval results, while the next best combinations of different cardinalities yield lower precision. In these KN-DB query classes, no possible unweighed combination of the set of studied 3D FVs improves over the usage of the best single FV (3DDFT). Consequently, using a static combination of FVs to answer all queries is expected to be only suboptimal if such classes/objects occur in a retrieval system. Then, it should be possible to further increase the retrieval precision by an appropriate scheme which *dynamically* decides which FVs (including the single FV mode) to use for answering a given query.

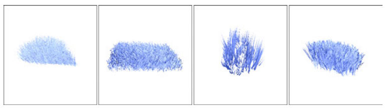
Figure 2.30 gives the precision-recall results for a hypothetical optimal query processor which for each KN-DB query object always selects the best unweighed FV combination (including usage of single FVs; a selection of 6 best FVs was the basis for these results). This intelligent query processor achieves about 60% R-precision on the KN-DB benchmark, which is another significant improvement over the retrieval quality achieved by the static (benchmark-average optimal) FV combination determined in Section 2.4.



(a) Weed retrieval results



(b) Galeons retrieval results



(c) Weed models

(d) Galeons models

Figure 2.29: Precision-recall results for the class-dependent best single FV and combinations of FVs, for two different KN-DB query classes (R-precision results are given in brackets). Any possible combination of multiple FVs results in lower retrieval precision as compared to the single FV usage. Practically, using fixed combinations of FVs for all possible queries is sub-optimal.

Building such a query processor is a difficult problem, as it is not clear how the system should decide which FVs to select to answer a given query. One approach is to use benchmarking to determine for each query class the optimal selection of FVs. Given a query object, the system could use a classifier to determine the benchmark class most similar to the query object, then lookup the optimal combination known from benchmarking (and stored in a table), and then execute the query using the indicated combination. This approach is problematic as (a) it is not clear if the dynamically submitted query objects are always representative for the benchmark used in building the combination table, and (b) classifying the query object also requires a similarity function, but finding the appropriate similarity function is the core problem, so this is a paradox.

Another approach is to use purely unsupervised information for finding a good combination to answer a given query. In [42] it was proposed to heuristically search the combination space until the histogram of resulting distances between the query and the candidate objects form a two-peaked distribution. It was assumed that if such a pattern is found, then there is a number of relevant nearest neighbors to the query (yielding the first peak of low distances), while the remaining objects are irrelevant and their corresponding (larger) distances normally distributed, yielding the second distribution peak. While this approach is an intuitively justified heuristic, we were not able to successfully implement it in our 3D search system. We presume this failure might be due to the small class sizes in our benchmarks, which do not allow the

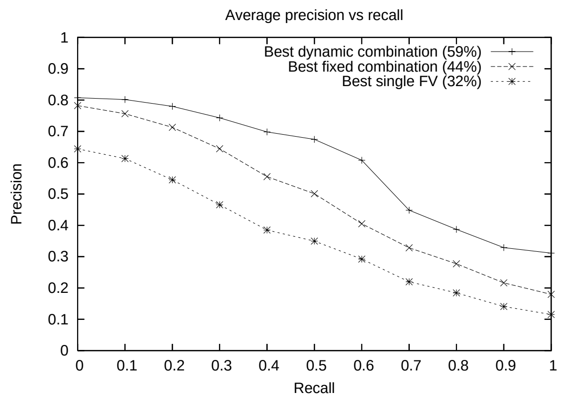


Figure 2.30: Retrieval results on the KN-DB using the single best FV (DBF), the optimal static combination, and the result using a hypothetical query processor choosing the optimal combination of FVs (including single FV usage, where optimal) for each benchmark query. Query-dependent combinations offer additional potential for boosting the retrieval precision.

respective distance histograms to yield a clear peak of low distances.

Given the problems of these two approaches, we decided to research a query processor which makes use of supervised object information without using distance histograms and without solving the classification problem. The basic idea is, given a query object, to *estimate* the retrieval precision to expect for each available FV on a so-called reference data base which consists of objects with class labels. The reference database is used to calculate a measure for the *coherence* of a prefix of the ranking resulting for each FV, and then use this measure to weight the FVs in a dynamically built combination. Weighting of FVs can be realized in a distance-based aggregation scheme (cf. Section 2.4.2) by scaling the distance contributions of each FV accordingly. Weighting as a special case also allows to form any kind of unweighted combination, including the usage of single FVs by setting the weights of participating FVs to one, and to zero for those FVs not participating.

We next present a heuristic implementing the sketched estimation of individual FV performance. We then define two methods that use this estimator to improve the effectiveness of the retrieval system by *selecting* a good FV given a query object from a pool of available FVs, and also by *combining* FVs with weights based on the estimator value.

2.5.2 An Entropy-based discrimination estimator

We use an estimator based on the *entropy impurity* [30] for determining the best FVs to use given a query object and a reference database (supervised information). The entropy impurity is a well known measure used e.g., in decision tree induction, where it measures the "impurity" of a node N of the tree w.r.t. the elements assigned to N. If these elements all have the same class label, then the impurity is 0, otherwise it is a positive value that increases up to a maximum when all classes are equally represented. Other impurity measures are the *Gini* and the *misclassification impurity* [30], but we experimentally obtained the best results using entropy impurity. We now define the concept of *k-entropy impurity*, and we show how to use it to build weighted combinations of FVs.

Method A: Query-dependent selection of a single FV

One way to implement a dynamic query processor in a multi-FV system is to try to *select the best suited FV* for a query object q. Our hypothesis for selecting good FVs is that good FVs are expected to provide a high level of *coherence* among the objects in the resulting ranking of answer objects, i.e., we expect to retrieve similar objects (objects from the same similarity class) at the first positions of the ranking. Using a reference database, we can measure the coherence of a ranking by the distribution of class labels contained in the ranking.

Let U be the universe of valid 3D objects. Let $T \subset U$ be a finite set of *training objects*, where $\omega_j \subseteq T$, $1 \leq j \leq N$, is a *class* of objects (i.e., all objects in ω_j are considered similar), and $T = \biguplus \omega_j$. Let $q \in U$ be a query object. Given a FV f, a *ranking* R_f^q is a list of objects from T sorted in ascending order by the distances between q and every object in T with respect to f. Also, $P_k(\omega_j, R_f^q)$ denotes the fraction of objects at the first k positions of R_f^q that are in class ω_j.

Definition 1. *The k-entropy impurity of a FV f with respect to q is defined as*

$$i(f,q,k) = -\sum_{j=1}^{N} \begin{cases} P_k(\omega_j, R_f^q)\log_2(P_k(\omega_j, R_f^q)) & \textit{if } P_k() > 0 \\ 0 & \textit{otherwise} \end{cases}$$

If the k objects are in the same class, the impurity is 0; otherwise it is positive, with the largest value occurring when the different classes are equally likely, and the number of classes covered by the k objects is maximal. We use the previously defined k-entropy impurity to measure the degree of coherence of each FV.

Definition 2. *Let $F = \{f_1, \ldots, f_M\}$ be a set of M FVs. The k-entropy impurity selection function is defined as*

$$EntImpSelection(F,q,k) = \arg\min_{1 \leq \ell \leq M} \{i(f_\ell, q, k)\}.$$

The FV that minimizes the k-entropy impurity for q is selected. In case of ties, the best FV according to a pre-computed ranking of FVs based on individually benchmarked performance (cf. Section 2.3) is selected.

Method B: Query-dependent weighted combination of FVs

Another way to leverage the effectiveness estimations is building a *combination of FVs* based on the estimator. The problem is to determine which FVs to combine, as inclusion of FVs unsuited for usage with q can reduce the overall effectiveness of the search system. We propose to use the k-entropy impurity to weigh each FV in the combination, giving more weight to those FVs with lower entropy impurity. Let d_ℓ be the distance function using FV ℓ, and let $dmax_\ell$ be the maximum distance between q and any object of the database using FV f_ℓ.

Definition 3. *The k-entropy impurity weighted distance between a query object q and an object $o \in U$ is defined as*

$$\delta_k(q,o) = \sum_{\ell=1}^{M} \frac{1}{1+i(f_\ell,q,k)} \frac{d_\ell(q,o)}{dmax_\ell},$$

We use $\delta_k(q,o)$ to produce the combined ranking list. Figure 2.31 gives the architecture of the proposed FV selection and combination scheme, including the set of FVs, the query object, the reference database used for weight calculation, and the actual database to execute (and evaluate) the queries on.

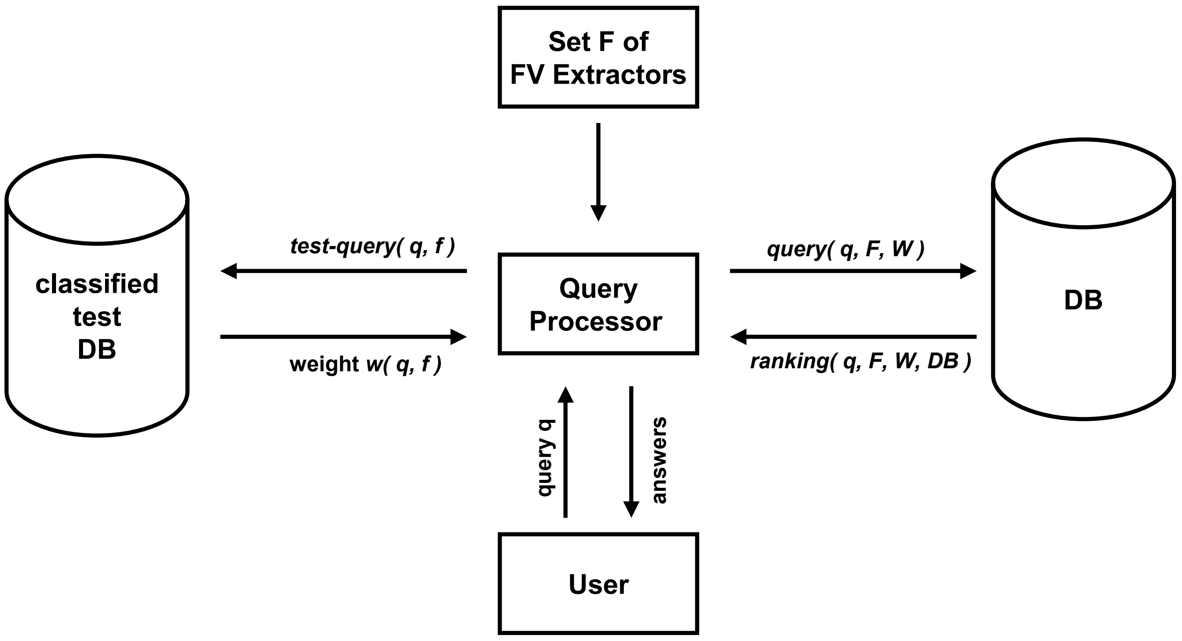


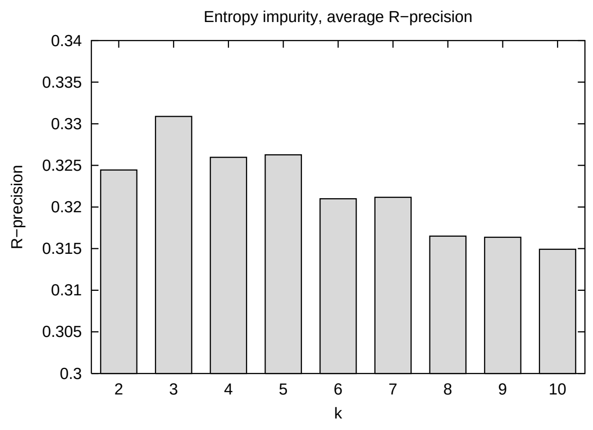
Figure 2.31: Architecture of the proposed dynamic query processor.

2.5.3 Results for dynamically weighted combinations

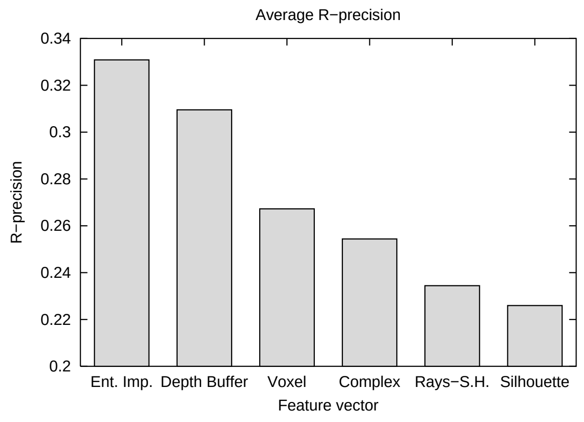
For experimentally evaluating this selection and combination heuristic we again used the KN-DB benchmark. We selected the 17 largest query classes containing each at least 9 objects for the experiments. Specifically, classes $5, 10, 14, 17, 18, 19, 20, 21, 26, 27, 32, 36, 49, 52, 53, 54, 55$ were selected (cf. Table A.1 in the Appendix). The classified objects were used as queries, and those objects which belong to the same model class as a given query q were considered the objects relevant to q. We used the L_1 norm to perform the similarity queries.

From our set of implemented 3D FVs, we selected the best five benchmarked FVs with their best dimensionality according to Section 2.3.3: Depth Buffer (366-d), Voxel (343-d), Complex (196-d), Rays-SH (105-d), and Silhouette (375-d). This ordered list of FVs also serves to resolve ties that may occur when using our selection criterion.

To evaluate the selection technique, we partitioned the set of classified objects into a training set and a test set in order to perform cross-validation [37]. We randomly partitioned the classified set of objects in two halves. One half was used as the training set T. The other half was used as the query set Q. For computing the effectiveness scores, the objects of T were not considered to be part of the database. We repeated this procedure s times, and we averaged the results to obtain final scores. We experimentally found that $s = 100$ yields stable results.



(a)



(b)

Figure 2.32: Entropy impurity, average R-precision varying parameter k (a). Average R-precision, all FVs (b).

Figure 2.32 (a) shows the average R-precision with the entropy impurity selection test (method A) while varying parameter k from 2 to 10. The best effectiveness score is achieved with $k = 3$, but the scores with $2 \leq k \leq 5$ are very similar. This result suggests that it is not necessary to search the optimum k for each similarity query, and that any k from 2 to 5 is equally robust. For $k > 7$, the effectiveness starts to decrease.

Figure 2.32 (b) compares the average R-precision of the five individual FVs and the 3-entropy impurity selection technique. The query-dependent selection technique manages to outperform the static policy of using the benchmark-average optimal FV (DBF) for every query. The improvement in effectiveness between the DBF FV and the selection technique is about 7%, which is significant in terms of retrieval effectiveness. It is comparable with the effectiveness improvement between two consecutive FVs in the list. Figure 2.33 (left) shows the precision vs. recall figures for all individual FVs and the selection technique. The average R-precision values are also indicated for each curve. The 3-entropy impurity selection has better precision for all recall levels compared with the best single FV, which means that our method is more effective than any of the studied FVs.

We also present experimental results of the proposed combination technique with k-entropy impurity (method B). Figure 2.33 (right) shows precision vs. recall curves for the best single FV and the combination method using 3-entropy impurity. One can observe a large effectiveness improvement of 29% in terms of R-precision using the combination technique, which is greater than any improvement between the single FVs used in these experiments. We obtained

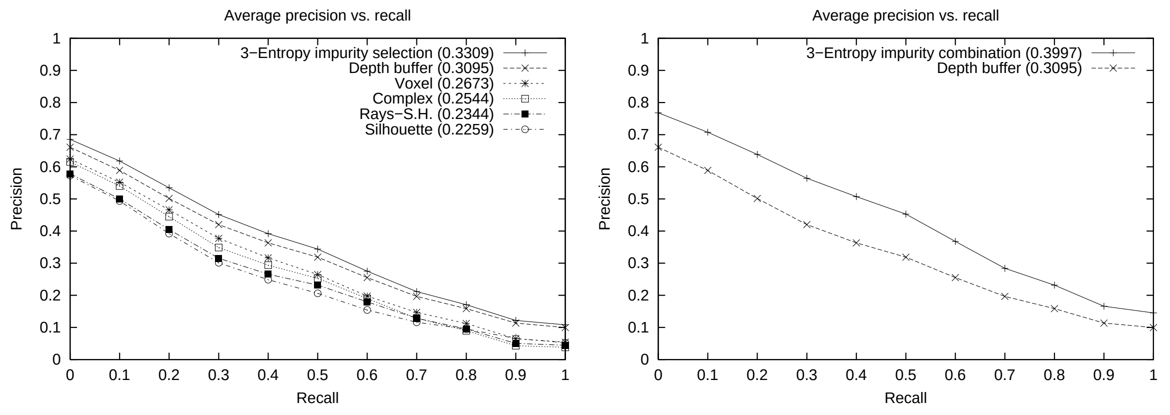


Figure 2.33: Average precision vs. recall, all FVs (left). Average precision vs. recall, combination with entropy impurity (right)

similar experimental results with $2 \leq k \leq 10$, which also suggests that it is not necessary to search for an optimum k value for each query.

Figure 2.34 presents a summary of the average R-precision values obtained with the proposed techniques, and compares them with the optimal selection score, i.e., for each query object the FV with the best performance with respect to the given query was used. It shows that the effectiveness obtained by our combination method is pretty close to the optimal single selection. We also tested the combination method using all of the 16 implemented FVs. We obtained a slightly better result (40.73% R-precision) than with the combination using 5 FVs. However, this improvement is obtained at the expense of higher CPU cost, because in that case we have to compute 16 rankings instead of just 5.

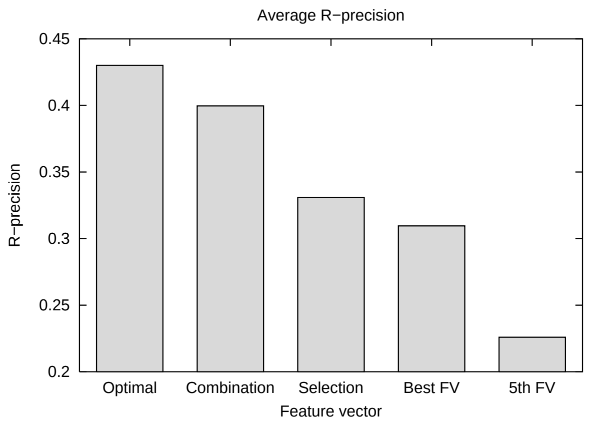


Figure 2.34: Comparison of the effectiveness obtained with the proposed methods based on entropy impurity.

2.5.4 Analysis of the results and practical recommendations

Experiments have shown that typically, it is only suboptimal from the retrieval precision viewpoint to fix a single FV or a combination of FVs to use for every possible query, but rather, appropriate selection or combination of FVs yields optimal retrieval results with respect to the available FVs. We researched two techniques based on the entropy impurity concept for query-dependent selection and weighted combination of FVs for query processing. The results show that the selection technique outperforms the DBF (single best FV) by 7 percentage points, and it comes close to the optimal single-selection strategy, which is interesting. Regarding the weighted combinations, these significantly outperform the theoretical optimal single selection. We have also compared to the dynamically weighted results with the optimal static combination determined as in Section 2.4. Then, the dynamic selection is still ahead, but only about one percentage point in R-precision. Note that the R-precision results here and in Section 2.4 are not directly comparable, as for the entropy experiments we restricted to a subset of KN-DB, as we needed larger-sized classes for cross-validation purposes. Our results show that retrieval systems actually can profit from automatic feature selection techniques. In [15], we have researched an alternative estimation function and also found it useful. The results presented in [15] have recently been confirmed using different FVs [69]. Additional new 3D FVs will probably be proposed in the future, but we do not expect that a single method will outperform all possible combinations on all possible queries. Dynamically combining FVs therefore is regarded promising.

Assessing the practicability of this architecture, we have to consider the costs and benefits of such a system in order to give practical recommendations. The main costs are *increased computational overhead* for evaluating the test queries and for dynamically forming the queries. Dynamic weighting potentially also raises the *secondary storage access costs* for retrieving the nearest neighbors from disk, as indexing multi-FV databases with variable FV weights is a challenging problem [12]. The costs also compound the *maintenance of the reference database* used for weight calculation. Conceptually, such a reference database can be build and maintained by the running system, by monitoring the users submitting queries and indicating relevant objects like done in relevance feedback. The quality of the reference database is important for the effectiveness of the discussed dynamic selection technique. Sensitivity experiments we performed indicate that the reference databases has to be similar to the evaluation database, in order to yield beneficial weight estimates. Specifically, we used two structurally similar but differen 3D databases for estimation and evaluation, namely, the PSB-Test and Train databases. Then, the retrieval results were not as good as when doing cross validation, where by definition the reference database actually is an unbiased sample of the evaluation database.

To economically judge the practicability of the system, we have to assess it's benefit. If we compare the dynamically weighted combinations' retrieval precision results with the optimal static combination results, the improvement of dynamic combination is smaller than comparing dynamic selection with static selection of single FVs. The dynamically weighted combinations' advantage over the optimal static combination amounts to about one percentage point in R-precision. Depending on the application requirements, it therefore may be sufficient to stick to a good static combination, which does not incur all of the above mentioned costs.

We here cannot come to an ultimate recommendation whether the costs of the dynamic approach will be outweighed by the benefit of a (relatively) small improvement in retrieval precision, as we would need to quantitatively assess the actual costs and benefits. Conceptually, cases can be made where even small improvements yield big returns. In certain application domains such as object recognition in industrial inspection or optical character recognition, which share similarities with content-based retrieval, even relatively small improvements in recognition rates may be highly desirable. Specifically, improvements in recognition rates on the magnitude of fractions of percentage points may be justified when compared against the costs of misclassification. As a second line of argumentation for the dynamic case, improved weight estimation schemes other than entropy impurity, or the availability of new feature vectors yielding retrieval precision highly specific to the type of query class could yield larger effectiveness gains as compared to static selection and combination schemes.

We finally point out that besides the reference-database approach, other methods for building weighted combinations are possible. One approach is to collect interactive relevance feedback information to optimize the weights to better fit a user's relevance profile, assuming that re-issuing the modified query results in more relevant answer objects. Of special interest for us are automatic techniques which do not require interaction or much supervised information. Another possibility for estimating FV effectiveness based on purely unsupervised information will be discussed in Section 3.2. Leveraging such information in a query processor for dynamic FV selection and combination is regarded as interesting future work.

3 Projection-based visual feature space analysis

Contents

In Chapter 2 we have evaluated single and combined FVs for the 3D data type domain regarding benchmarked retrieval precision. It was concluded that certain FVs and combinations thereof are well suited for capturing similarity relationships in object space. While "query-by-example" retrieval is a prominent FV-based database application, other forms of access to database content are possible. Consider, e.g., the case when a user is introduced to a previously unknown database. Clearly, she will not have query objects readily at hand, but will first need to get an overview over the database content in order to assess important database characteristics such as distribution of objects and object clusters to expect. To this end, visual representations of database content prove to be helpful. Usually, the amount of objects contained is rather large and the user cannot be expected to visually inspect each and every database object directly. Rather, clustered or compressed representations are a desirable tool.

Section 3.1 addresses visual support for database summarization, organization and retrieval using the Self-Organizing (SOM) [59] algorithm by Kohonen. The usability of the SOM algorithm for FVs of hundreds of dimensions is demonstrated on several datasets. Then, based on SOM-technology, in Section 3.2 we propose two tools for unsupervised visual FV space discrimination analysis and feature engineering. Furthermore, a form of supervised visual analysis for class discrimination analysis and comparative FV space benchmarking in projected space is then introduced in Section 3.3. There, the idea is to visually aggregate class distributions given as points in projected space by convex hulls over class distributions. This visualization metaphor is motivated, practically applied on two data sets, and statistically validated.

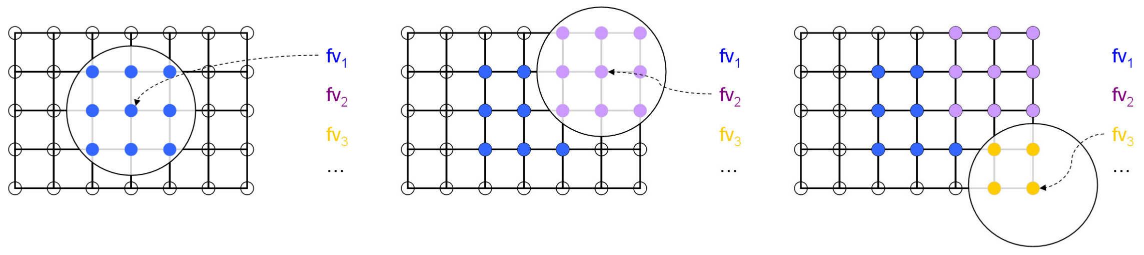
Parts of this Chapter appeared in [79, 78, 54, 53, 52, 13, 14].

3.1 Interactive organization and retrieval with Self-Organizing Maps

This Section discusses the application of the Self-Organizing Map (SOM) algorithm to several different multimedia datasets. The SOM is proposed as a practical tool for compressing large multimedia databases into a manageable set of prototype vectors which can be visualized for database presentation, and used as an interface for querying and visual analysis. Section 3.1.1 briefly introduces the SOM algorithm. Sections 3.1.2 and 3.1.3 then discuss usage of SOM-based visualizations for providing effective overviews over several multimedia datasets, as well as a concept for using them as part of a retrieval interface. We recognize that SOM is a highly popular technology that has been used for quite some time now and hundreds of papers have been published on the SOM algorithm and applications thereof (see the bibliography given in [59]). Nevertheless, to the best of our knowledge our applications of SOM to the 3D, E-Mail, and Growth-Matrix domains are novel. Also, they show the practical applicability and provision of meaningful results even on datasets of very high dimensionality (up to 500 dimensions are given in the application here). Finally, scaling the saturation attribute in SOM maps to indicate SOM occupancy density is a novel tool which we introduce in Section 3.1.2.

3.1.1 Kohonen's Self-Organizing Map (SOM) algorithm

A natural way to deal with growing amounts of data is to apply data reduction techniques. E.g., we may reduce a large data set to a small number of prototypes using clustering schemes, and then visualize the obtained cluster prototypes, possibly enriched by additional information indicating cluster characteristics. The *Self-Organizing Map* (SOM) [59] algorithm is a non-linear projection and data reduction algorithm which is popular in analysis and visualization of large high-dimensional data sets. It is a scheme for training a neural network representing a distribution of high-dimensional input data. A low-dimensional (usually, 2-dimensional) grid of reference vectors is learned from the input data by means of competitive, iterative adjustment of reference vectors to the input data space. Effectively, the SOM is a combined vector quantization and projection algorithm, as (a) many input records are represented by a fixed number of reference vectors, and (b) the reference vectors representing the input data are given a topological ordering by the SOM grid. The SOM yields (a) a clustering of the data and (b) approximately preserves the topology of the data points from the input space, and is therefore especially useful for data visualization and exploration purposes. We here do not recall the SOM algorithm details but instead refer to the standard reference for SOM technology given by its inventor, Teuvo Kohonen [59]. Figure 3.1 illustrates three adjustment steps during SOM training.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 3.1: The SOM is an iterative competitive learning algorithm. During training of the SOM, input data vectors are matched to (appropriately initialized) reference vectors located on a grid, and then the reference vectors of the match, as well as those of neighboring nodes, are adjusted to the input data.

SOM training parameters

Training of Self-Organizing maps is a semi-automatic process during which a number of parameters have to be adjusted appropriately to obtain good results. The main parameters of SOM training include the dimensionality and topology of the SOM grid; the neighborhood kernel and learning rate function; and the number of iterations for which the learning is allowed to take place. Also, for initialization of the prototype vectors, several options exist. The choice of parameters and initialization usually is determined by characteristics of the data set at hand in combination with a number of rules-of-thumb known in the literature [59, 60]. A standard approach in tuning these parameters is to generate multiple SOMs for a range of parameter and initialization settings. From these candidate SOMs, the best one according to an selection criterion is chosen. Various selection criteria are possible and depend on the data set at hand. An unsupervised quality criterion often proposed is the average quantization error of the SOM with respect to the data set. Supervised criteria include measures for classification accuracy on labeled training data. Also, manual inspection by an expert is possible.

For the SOMs to be discussed in Sections 3.1.2 and 3.1.3, we largely relied on the rules-of-thumb given in [60], in combination with manual inspection, where we judged the quality of the resulting SOMs by considering the distribution of database elements over the SOM. Usually, only a few runs were sufficient to obtain satisfactory results. Our experience after generating SOMs for many different data sets is that generally, the SOM gives quite robust results, and the specific parameter choices are not so important as long as the rules-of-thumb are roughly adhered to. Additional support for the robustness of the SOM algorithm with respect to a key parameter (grid dimensionality) will be presented in Section 3.2. There, in a sensitivity analysis main SOM-based results are shown to hold for significantly varying SOM grid dimensionality settings.

Applications areas and SOM visualization methods

SOMs have previously been successfully applied to many data analysis and visualization tasks. Examples of applications to multimedia data collections include text documents (WebSom [43]), music (Islands of Music [71]), and also images (the PicSOM retrieval system [63]). Several visualization techniques supporting different SOM-based data analysis tasks exist [95]. The so-called *Ultsch-Matrix* (U-matrix) visualizes the distances between reference vectors of pairs of adjacent nodes, and visual cluster analysis can be performed by searching for connected sets of SOM nodes which have low distance between each other, but high distances with respect to surrounding map regions. *Component planes* are useful in visualizing the distribution of individual components in the reference vectors, supporting correlation analysis. If the input data points are mapped to their best matching reference vectors, *histograms* of map population measures, e.g., the distribution of objects over the map using smoothed data histograms [72], or labels of classes, are possible.

3.1.2 Visual analysis of 3D, Email, and time series databases using the SOM

We here discuss application of Self-Organizing Maps as a means for browsing and presenting large multimedia databases. The feasibility (the meaningfulness of the results) of SOM even on datasets of very high dimensionality (up to 500 dimensions) is demonstrated. Domain-dependent interpretations of the SOMs are given, illustrating use-cases in the respective domains.

3D benchmark

The center part in Figure 3.2 displays the allocation frequency of the Konstanz 3D benchmark (KN-DB, cf. Section 2.3.2) models on a 12×9 SOM learned from the (366-dimensional) Depth Buffer descriptor (cf. Section 2.3). A smoothed density histogram [72] is generated over the node occupancy frequencies and indicated by a blue/red bipolar colormap. The surrounding images show model browser windows displaying objects located (in the nearest neighbor sense) at different map nodes. The SOM topologically organizes the objects nicely by geometric properties: Top-left are located groups of thinly-elongated objects like swords,

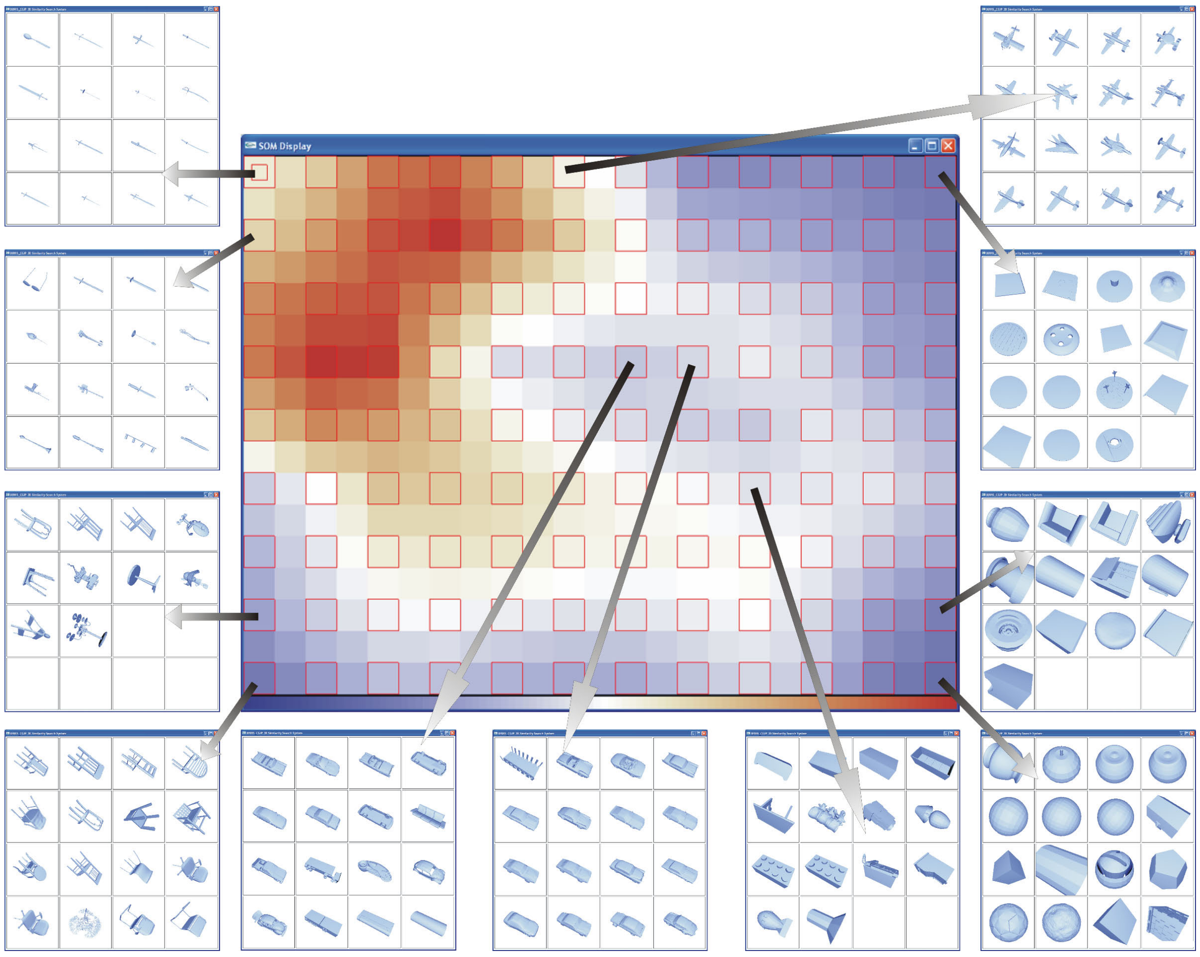


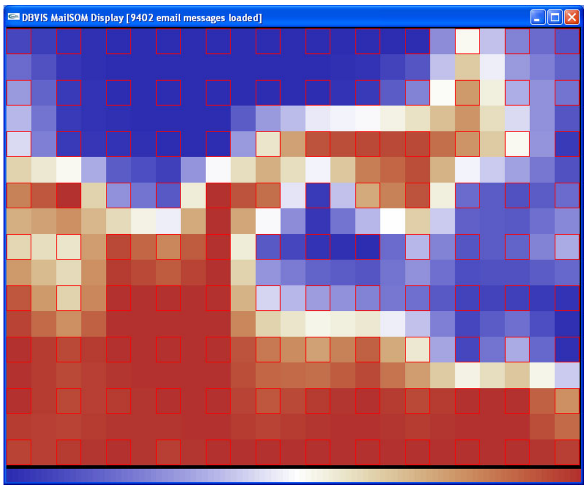
Figure 3.2: Self-organizing Map trained with the Konstanz 3D Benchmark described by the Depth Buffer FV (cf. Section 2.3). The map coloring reflects a density histogram, and the surrounding windows show database objects best-matching selected nodes on the map. Interestingly, the spatial distribution of elements over the map can be meaningfully interpreted in terms of gradual change in shape of the models.

lamps, spoons etc. On the opposite corner of the map, we have quite converse shapes - located are voluminous, block-like and spherical objects. If we move from top-left to bottom-right along the map border in counter-clockwise direction, the located objects seem to transit from the thin-elongated shapes to the voluminous shapes via more complex objects composed of combined elongated elements such as chairs. Transiting from top-left to bottom-right along the diagonal, the objects also become more voluminous, but this time via more compact shapes such as cars and planes. Combining object browsing and density histogram visualizations, a user can quickly gain an overview over large collections of models. Also, presentation of the content of a 3D database is easily possible. To support this use case, a domain expert can manually annotate the SOM, highlighting areas of interest to novice database users. Annotating SOMs in a multi-user environment could also be easily realized by a user collaboration approach.

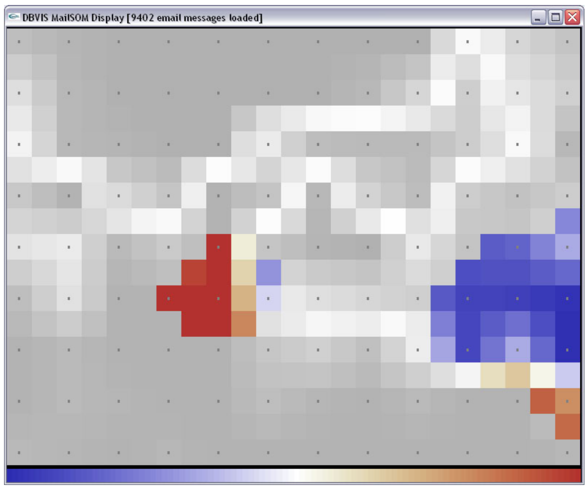
E-Mail

Email has become one of the most important means of communication. Much work has been done improving the efficiency of email management, while the effectiveness of email management from a user perspective has not received a comparable amount of research attention. SOMs seem prospective in visually supporting management of large collections of E-Mail similar to the WebSOM system [43]. To obtain FVs from E-Mail data, we employ a well-known scheme from Information Retrieval. First, we determine a number n of most frequent terms from the subject fields of all emails in the archive, after having filtered the subject fields using a list of stop-words in order to avoid the inclusion of non-discriminating terms in the description. Then, we apply the $tf \times idf$ document indexing model [6], considering each email as a document represented by its subject field. The model assigns to each document and each of the n terms a weight indicating the relevance of the given term in the given document with respect to the whole document collection. The concatenation of the term weights for a given document gives the FV for that document. The set of FVs is then used as input for the SOM generation. We note that more sophisticated email FV extractors can be thought of. Specifically, email data usually contains a wealth of meta data and additional attributes that are candidates for inclusion in the descriptor. In this experiment, we chose to start with a basic feature extractor, and leave the design of more complex E-Mail descriptors for future work.

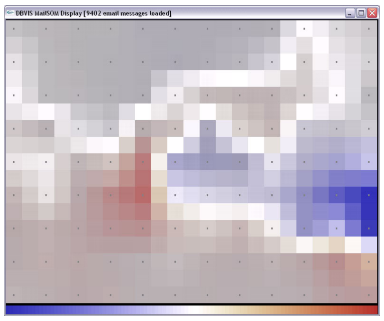
We generated a SOM from an archive of 9.400 emails collected from our working group's E-Mail server system, using the 500 most frequent subject field terms for the $tf \times idf$ descriptor. We labeled all emails as belonging to either the spam or the non-spam class, as judged by a spam filter and manual classification. Figure 3.3 (a) shows what we call the *spam-histogram* over our E-Mail database. For each map node, the colormap gives the fraction of spam emails among all emails mapped to the respective node. Shades of red indicate high degrees of spam, while shades of blue indicate low degrees of spam (these are the "good" email regions on the SOM). Clearly, the SOM learned from our basic $tf \times idf$ FVs is capable of discriminating spam from non-spam email. Note that the bipolar colormap is especially useful for visualizing binary class distributions. If the classes are well separated on the map, and as we interpolate the spam frequencies between adjacent nodes, then the regions between the class boundaries are interpolated to about $\frac{1.0+0.0}{2} = 0.5$ which corresponds to the white ("neutral") interval in the used colormap. Thereby, the separation is also visually emphasized, as there emerges a clear structure in the data.



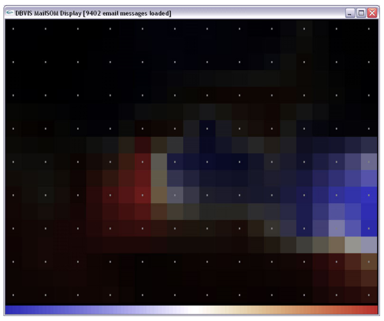
(a) Spam Histogram



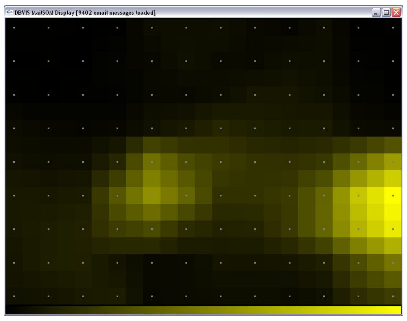
(b) Discrete brightness



(c) Continuous saturation



(d) Continuous brightness



(e) Continuous monochrome

Figure 3.3: SOM-based analysis of a spam/non-spam classified E-Mail dataset. (a) spam-histogram (red indicates spam, blue indicates non-spam). (b)-(e) represent the component plane for term #214 ("work") using several rendering options: Usage of a fixed threshold (b) and scaling the spam histogram via the saturation (c) and brightness (d) attributes. (e) gives the classical component plane visualization using a monochrome color.

Our E-Mail FV consists of weights of discriminating terms from the E-Mail. We can use the SOM to analyze the distribution of the weights over the SOM. To this end, component planes [95] have been proposed. The component plane technique visualizes the distribution of a selected dimension over the SOM reference vectors, usually by means of monochromatic intensities, or varying the sizes of symbols representing the SOM grid nodes. Figure 3.3 (e) illustrates the former approach, where we denote the weight for term #214 which is "work" by varying intensities of color yellow set in linear proportion to the normalized weight magnitudes. By comparing the weight histogram with the SOM, we learn that this term occurs in E-Mails both of type spam and non-spam. The right-hand side "work" cluster compounds university-related emails from one PhD student in our working group. While such analysis is possible, it requires the user to analyze two images in parallel (the class histogram and the component plane). Recognizing that the bipolar colormap nicely communicates the structure of the spam/non-spam distribution, we propose to combine both views by manipulating appropriate visual attributes in the spam histogram image based on the weight magnitudes. Thereby, the cognitive load of the user is reduced, as she does not have to switch forth and back between several images.

We experimented with several different options for combining both images by scaling saturation or brightness attributes in HSV color space of one image, based on the other. In Figure 3.3 (b), we employed a simple threshold scheme based on weight magnitude. For each node, if the weight is larger or equal to 0.3, we set the saturation attribute to 1.0, else we set it to 0.0. We note that the clusters are easily perceivable, and the class memberships are clearly communicated. The overall class structure on the map remains nicely perceivable even for those regions where the weights are close to zero, which is due to the nature of the bipolar colormap. On the other hand, finding appropriate thresholds requires some user intervention. Also, varying weight magnitude within the clusters is suppressed due to the quantization of the weights to zero or one.

Figures 3.3 (c) and (d) illustrate continuous combinations of the class and the weight histograms. (c) linearly scales the saturation attribute of the class histogram to the term weight, retaining the full weight density information. We see that the weight quickly diminishes when going away from the cluster centers. Again, the structure of the class histogram is visible. Finally, Figure (d) linearly scales the brightness attribute of the class histogram image based on the normalized term weight. Using brightness, the class structure is suppressed. On the other hand, the image is less complex as the structure boundaries are not included in the display. We summarize that for the case with binary distributions and an appropriate colormap, we can easily combine two complementary SOM-based data views (distribution of classes and single components in the reference vectors).

Asset price growth rates

The *Growth Matrix* [54] technique is a triangular visualization technique for showing growth rate behavior for financial asset price series, for each of the possible subintervals given in a time series of fixed length. The basic idea is to map start and end point in time to a 2D cartesian coordinate system, visualizing the magnitude of growth or loss by a color-coding scheme (cf. Figure 3.4 for some Growth Matrix examples). The growth matrix is a tool for analysis of intra period growth patterns within large time series of assets prices. It can also be used to compare growth patterns among classes of assets. For the latter task, it is necessary to find good layouts by which to visualize multiple Growth Matrices in a single display. Here, we utilize the SOM technique to cluster large sets of Growth Matrices in order to produce informative overviews. Particularly, we chose a subset of an asset database [54], consisting of 1.700 growth matrices of dimensionality 100^2 (time intervals range from $01/1997$ to $03/2005$). For training the SOM, we down-sample the original data to matrices of size 20^2, resulting in vectors of dimensionality 190 which in turn are input to the SOM algorithm. We train a grid of 12×9 reference vectors onto which the original data is mapped back. Image 3.5 (a) visualizes

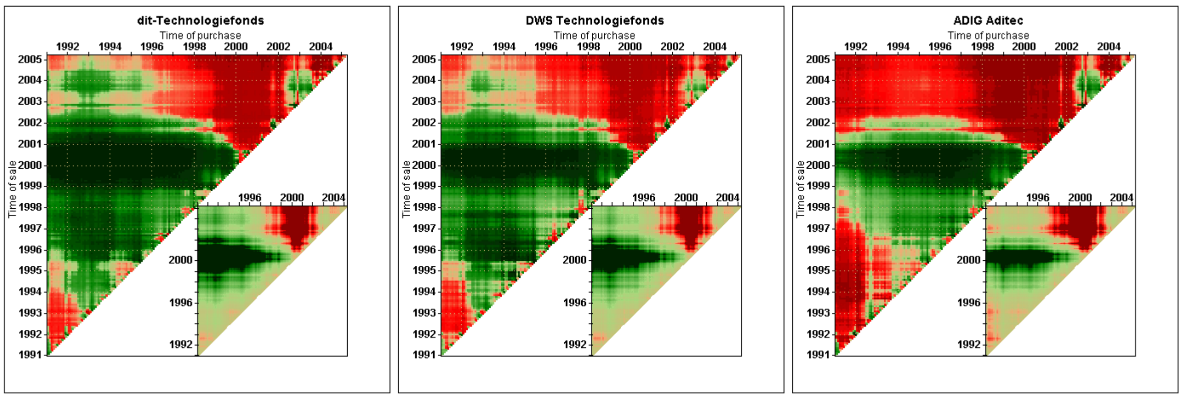


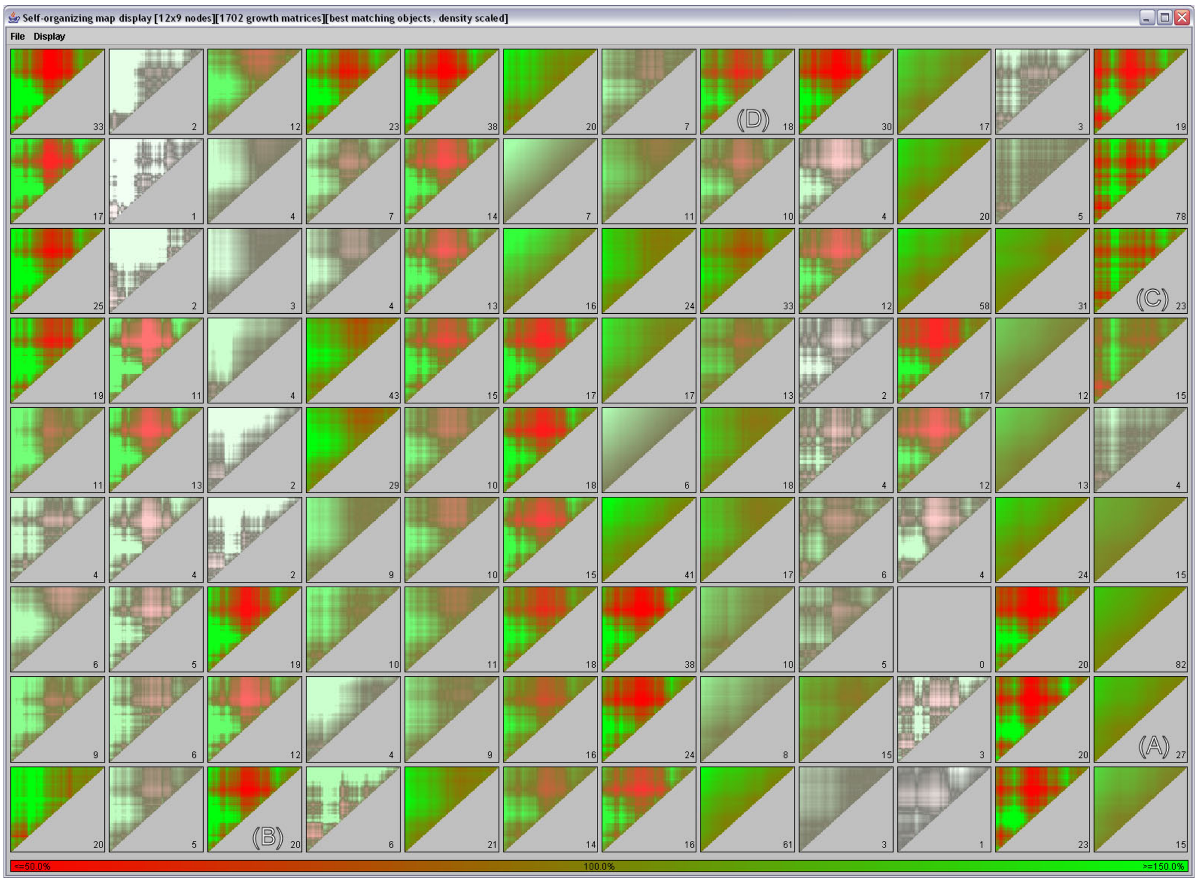
Figure 3.4: Growth matrices for funds composed of European technology stock. See [54] for a detailed discussion of this visualization technique.

the obtained SOM by drawing the best matching Growth Matrix for each SOM node, where the *saturation* attribute in HSV color space of each Growth Matrix thumbnail is scaled using 75%-quantile normalization according to the total number of objects matching the respective node. The display allows to draw conclusions regarding the characteristics of the given asset database. Particularly, we can easily identify three salient patterns occurring in the data set:

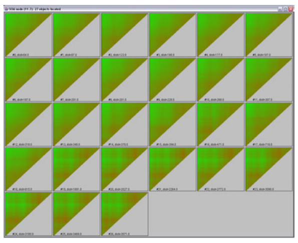
1. Steady growth throughout the considered time interval (possibly assets composed of securities; mostly green areas; cf. Node *A*);
2. Significant losses during the so-called *dot-com* crash period followed by relative recovery (mostly assets composed of stocks; mostly red growth intervals in the upper right region, and green ones in the lower left region; Nodes *D*,*B*);
3. Assets showing more mixed growth characteristics during the observation interval (heterogeneous (red/green) growth patterns; top half of the rightmost column on the map; Node *C*).

Globally, we can perceive a separation of the different patterns on the map. The steady growers separate the mixed growers and the dot-com type assets along two vertical axes in the right half of the map. On the left half, we observe a low number of degenerate Growth Matrix images with extreme and sharp color distributions, e.g., the top three nodes in the second column, due to erroneous data elements. We can also perform closer inspection of individual clusters by browsing the assets mapped to (in the nearest neighbor sense) certain nodes. Image 3.5 (b) illustrates several nodes representing the above identified patterns.

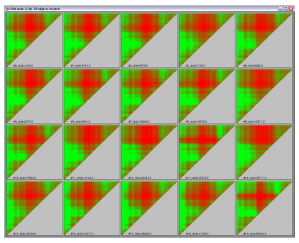
Also from application on this specific data type (matrices of growth rates) we conclude that the SOM is a well suited technique for simultaneously analyzing large multimedia databases. For training the SOM, a number of growth coefficients (190 were used here) can directly be used as input to the algorithm, producing meaningful results.


Self-organizing map display [12x9 nodes][1702 growth matrices][best matching objects, density scaled]
File Display
(D)
(C)
(A)
(B)
<=50.0%
100.0%
>=150.0%

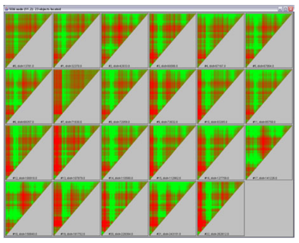
(a) Growth Matrix SOM



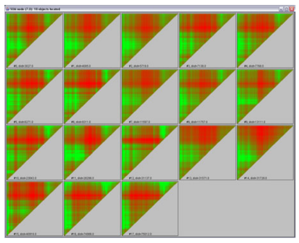
(b) SOM node A



(c) SOM node B



(d) SOM node C



(e) SOM node D

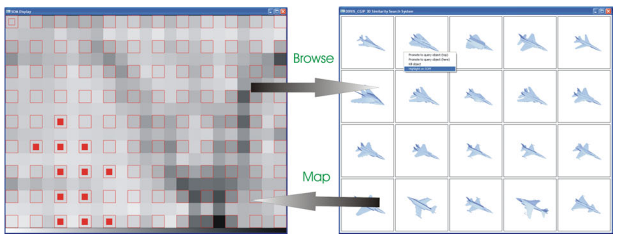
Figure 3.5: Self-organizing map of 1.700 growth matrices (a). The map was learned from the down-sampled original data. The input space is compressed in a meaningful way, allowing to identify salient global patterns in the data. Saturation scaling is performed on the thumbnail images to give additional visual hints regarding the density distribution of the patterns. Visual inspection of assets represented by different SOM nodes (b-e). (b): steady growers (SOM node A); (c,d): dot-com (SOM nodes B,D); (e): mixed patterns (SOM node C). Please refer to (a) for the location of the SOM node labels ($A-D$) on the map.

3.1.3 SOM-based support for retrieval and visual relevance feedback

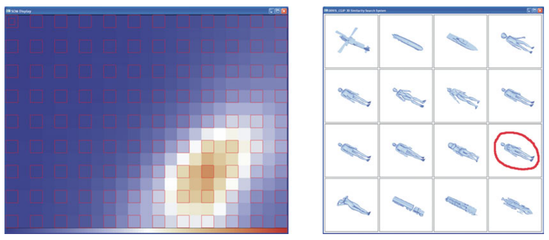
Besides data reduction and presentation purposes, SOMs are also suited as a user-friendly interface for querying for content. The PicSOM system [63, 76] demonstrated this. It is an integrated retrieval and exploration system for image databases described by multiple FVs. The core of PicSOM is a set of hierarchically structured SOMs called the TS-SOM index. From root to leaf elements along the tree, the SOM grids get larger. On a given level, each SOM grid unit references one chid SOM, organizing all the data elements which are best matching the given parent SOM node. On top of this hierarchic structure, a combined browsing and retrieval system is developed. Starting with the root of the index, the user is confronted with SOMs where a thumbnail image is drawn for the best matching image of each grid node. There are two modes of operation. First, the user can drill down the TS-SOM index top-down on a path for the best fitting image on each level, finally reaching a leaf-level SOM assumed to contain (most) relevant images according to user preferences. The other mode is based on relevance-feedback. While browsing SOM-levels, the user may mark thumbnail images she is strongly interested in. The system then identifies the SOM nodes on the leaf level where the marked objects best match, and presents other image elements matched at the respective leaf nodes by certain heuristics.

We have implemented a simpler, non-hierarchic system for SOM-based retrieval of 3D models. We use one global SOM representing the whole dataset. The SOM is an optional view which is provided in addition to the standard ranking displays. The SOM is bidirectionally linked with the retrieval interface. Such integration of query-by-example retrieval and SOM-based overview allows to explore many interesting querying paths *simultaneously*, as opposed to the linear browsing of sorted answer lists returned by standard retrieval systems. In our system, SOM nodes can be selected by the user, and the objects mapped to the selected nodes (the best-matching 3D objects) can be loaded into the 3D model browser, allowing to explore the SOM's structure. Vice versa, for each object in an answer list, its respective location on the SOM grid can be highlighted. By coupling nearest-neighbor querying and SOM visualization, the user can enhance the querying process by exploring those SOM regions where interesting answer objects are mapped to, possibly finding additional relevant objects, which in turn can be used for refining the query using relevance feedback techniques.

The SOM exploration process is supported by different SOM views which can be interactively switched. The U-Matrix allows assessment of the clustering structure of regions where relevant objects are mapped to (cf. Figure 3.6 (a) for an illustration). Furthermore, we implemented the response surface technique [95] visualizing the distances between a given 3D model's FV, and all SOM reference vectors, thereby revealing the map region(s) best representing the data vector (cf. Figure 3.6 (b) for an illustration). The SOM nodes closest to the query model are good candidates for further inspection when searching for similar objects. In this illustration, the user has selected one object she is interested in from a given list of answer objects (the human model denoted by a red circle). On the response surface regarding this object's FV, a set of node prototype vectors exhibits low distance (denoted by red shades) to the selected model, indicating a potentially interesting map region to explore.



(a) Best matching units marked on U-Matrix

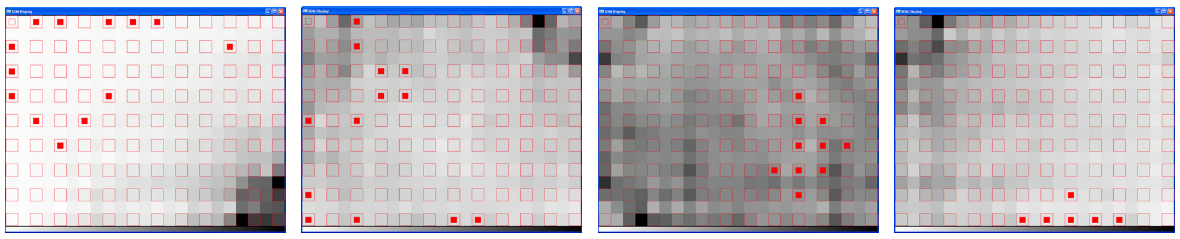


(b) Response Map to selected object

Figure 3.6: Supporting the retrieval process by marking relevant objects found in answer lists on the SOM (a). In (b), the response surface view visualizes the distances of SOM reference vectors to a selected object. The views can be used to inspect the 3D models contained in the matched nodes, possibly finding additional relevant answer objects.

Considering that not just one, but often multiple different feature extractors are available for indexing multimedia data, it is an interesting question how the SOMs generated for the same database represented in different FV spaces relate to each other. Also, we would like to have a tool supporting the user in selecting the FV extractors best suiting her needs. We therefore propose to include visualization of SOMs generated using different FV extractors for visual estimation of the appropriateness of the given FV extractors for a given query. Figure 3.7 shows SOMs for four different feature spaces generated for our 3D database. (a) and (b) show U-matrices of a moment-based and an extension-based FV, respectively. These two FVs have experimentally been found to exhibit rather low retrieval performance on database-average (cf. Section 2.3). Subimages (c) and (d) show U-matrices of two image-based FVs. These FVs offer comparably higher retrieval performance on database-average. We find these benchmarked results confirmed by inspection of the respective SOMs. For the low-performance FVs, the U-matrices show that roughly the same low distances exist between the reference vectors of many of the map nodes (darker grey shades indicate higher magnitudes of L1 node distances in vector space). On the other hand, for the high-performance case, the U-matrices show a richer and more heterogeneous pattern of inter-node distances (cf. Section 3.2 for a more thorough analysis of this phenomenon). We therefore expect these FVs to offer higher object discrimination power. In accordance with this observation, the members of a class of similar objects (i.e., the airplane models from the Konstanz 3D Benchmark, illustrated in Figure 3.8 (a)) are spread widely over the SOMs given in (a) and (b), while they cluster much better on the maps (c) and (d) (note that all map nodes best matching the airplane models are marked red in the maps).

Such comparative displays can be used interactively as follows. Assume a user has marked several objects as being relevant. SOMs for the available FVs are displayed with marks indicating the best-matching units to the relevant objects. Considering response surfaces and U-Matrices, the user can search for promising FVs and regions which to interactively explore for more objects. Ideal FVs are expected to produce SOMs where the matched nodes are compactly located, and being clearly discriminated from the surrounding map regions. The latter property can be assessed by considering the U-Matrices, and also the response surfaces for the matched objects.



(a) PMOM (16%) (b) COR (17%) (c) SIL (27%) (d) DBF (32%)

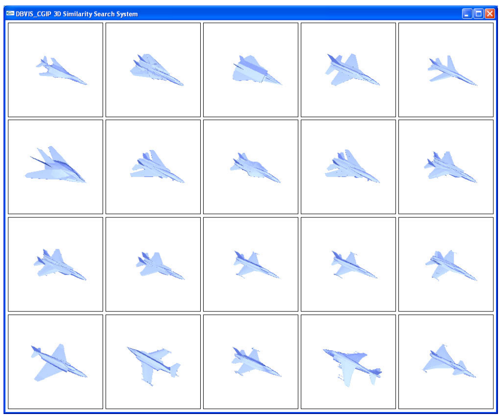
Figure 3.7: U-Matrices for four different FV extractors and the Konstanz 3D benchmark, sorted increasingly by benchmarked retrieval precision (R-precision scores are given in brackets). Marked red are best-matching units for a class of plane models, cf. Figure 3.8(a).

In addition to the above discussed distance-based characteristics, another interesting relationship between global FV retrieval performance and SOMs can be experimentally established. This is done by defining an appropriate measure for the benchmark-based purity of SOMs. We define the *map-purity* as the size-weighted average purity of all SOM nodes, where the purity of a node is equal to the fraction of objects belonging to the largest object class present at the considered node. Figure 3.8 (b) plots DB-averaged retrieval precision measures (cf. Section 2.3) of several FVs against the map-purity values of the corresponding SOMs for different SOM grid resolutions. There is a clear dependency between both metrics. It implies that better FV extractors, as measured by supervised benchmark, also better cluster similar objects on the SOM grid. We therefore can expect better benchmarked FV extractors to be better suited for the above described SOM-based exploration support for retrieval. Please note that these results were obtained by using the Princeton Shape Benchmark Train database, which consists entirely of classified objects. We conclude that by inspecting the U-matrices of different FVs, a user is able to identify those FVs that suit her needs, that is, find FVs that offer good global (or local, where required) discrimination power.

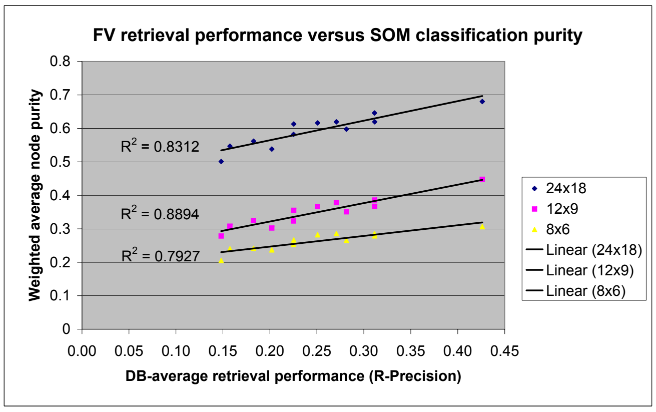
3.1.4 Summary of the results

We summarize the empirical findings from our application of SOM technology on a variety of multimedia databases as follows.

- The SOM allows compact representations of large datasets of multimedia objects described by FVs. The maps summarize many objects by a fixed amount of prototype vectors. Often, the results can be meaningfully interpreted in terms of map topology.
- The high dimensionality of some of the considered FV extractors does not seem to be a problem for the SOM algorithm. Evidence are the semantically meaningful SOM results



(a) KN-DB Class ID 20



(b) Classification purity and R-precision

Figure 3.8: (a) the airplane class visualized in Figure 3.7. (b) regression between benchmarked retrieval precision and classification purity of best-matching units on the SOMs.

obtained, e.g., for the Depth Buffer described 3D database or the Email database from Section 3.1.2, where the vectors were of dimensionality 366 and 500, respectively.

- The quality of the FV extractors used to generate the SOMs has a large impact on the quality of the resulting maps. The better the discrimination power of a FV, the better the SOM-based results with respect to topological organization of the objects regarding similarity can be expected.

- The SOM offers a wealth of options for visualization which in turn support content-based retrieval and object browsing:

  - Density histograms in combination with thumbnail maps allow summarization and presentation (abstracting) of large multimedia databases.
  - U-matrices can be used for visual cluster analysis in the datasets.
  - Marking of best-matching nodes and usage of response surfaces are suited for interactive retrieval using the SOM.
  - Comparative visual inspection of SOMs supports the interactive selection of the FV spaces best supporting the retrieval of similar objects the user is interested in.
  - Component planes allow correlation analysis, and inspection of component-based characteristics of distributions of object classes.
  - Several schemes for visual combination of class histograms with component planes using scaling in HSV color space were proposed in this Section.

3.2 Unsupervised visual feature space analysis: Tools for distance- and component-based discrimination estimation

3.2.1 Background

As has already been indicated in this thesis, in the 3D domain there exists an abundance of FV extractors to chose from. This situation is also true for other multimedia domains, such as the image domain [93]. Like done in Chapter 2, effectiveness of individual FV extractors can be benchmarked if suitable ground truth classifications (*supervised information*) are available. Also, supervised FV engineering by dimensionality reduction [104] or building appropriate combinations of FVs (c.f. Section 2.5) is then possible. Practically, due to the large number of extractors available, and the costs and even potential instability [65] associated with many benchmarks make supervised identification of the most effective extractors for a given application difficult. An alternative is to resort to *unsupervised* estimation of FV space effectiveness. To this end, a number of advanced statistical approaches have been proposed [2, 42]. These works are of rather theoretical nature and to the best of our knowledge have not been practically applied yet.

In this Section, we address the problem of *unsupervised FV space analysis* by means of characteristics obtained from compressed (clustered) FV space representations. As we are interested in visually supporting the analysis, and based on the previous work from Section 3.1, we rely on the Kohonen Map (SOM) algorithm for FV space compression. The Kohonen Map resembles a robust algorithm well suited for visualization [95]. In Section 3.1 we have explored the application of Kohonen Maps in a multimedia retrieval system. In the following, we now leverage unsupervised information extracted from Kohonen Maps for FV selection and engineering.

3.2.2 A distance-based discrimination power estimator

We propose an intuitive, simple, and practical method for unsupervised estimation of FV space discrimination power. We base our method on the following hypothesis:

Hypothesis 1. *Discrimination power provided in a given FV space can be estimated by the degree of uniformity of the distance histogram defined over inter-cluster distances in the respective FV space.*

An important assumption underlying Hypothesis 1 is that a FV space can be represented by a number of cluster prototypes as obtained by application of an automatic clustering algorithm, e.g., *k-Means* or the *Kohonen Map*. We then consider the distribution of distances between adjacent cluster prototypes. We expect the corresponding distance histograms to approximately resemble uniform distributions if the underlying FV spaces provide good discrimination power, as a-priori there is no rationality why any specific distance intervals should be preferred. While this has not necessarily to be the case for any possible combination of FV extractor and multimedia database, we expect uniform distance distributions to provide the best chances for meaningful discrimination in FV space. Conversely, we assume that for FV spaces providing only little discrimination power, cluster distances may be arbitrarily biased towards any subset of distance intervals.

3.2.3 A component-based discrimination power estimator

Any meaningful distance function $d : (\vec{p}_i, \vec{p}_j) \rightarrow \mathbb{R}_0^+$ in vector space, such as the Minkowski or Quadratic Form distance functions, has to rely on the components (dimensions) in the FV space. So it is ultimately the sum of characteristics of the individual FV components that determines the FV effectiveness. We next state a second hypothesis, and propose a tool for visualizing certain component-based FV space characteristics supporting unsupervised discrimination power estimation and feature engineering.

Hypothesis 2. *Discrimination power provided in a given FV space can be estimated by the degree of heterogeneity among the components of the cluster prototype vectors representing the FV space.*

Similar to Hypothesis 1, the intuition behind Hypothesis 2 is that FV spaces exhibiting high heterogeneity of prototype vector components can be attributed better chances to provide meaningful discrimination power. The more biased the component values are towards certain component intervals, the less chances are expected for good discrimination power.

Based on these considerations, we propose interactive FV space evaluation by visualizing the component distributions of the cluster prototypes in FV space. Again, we rely on the Kohonen Map algorithm. A Kohonen *component plane* (CP) [95] visualizes the distribution of one selected FV dimension over the respective Kohonen Map. We can visualize all component distributions in a FV space by simultaneously displaying the set of CPs in a component plane array (CPA).

3.2.4 Application

Application of the distance-based estimator

We tested Hypothesis 1 (the distance-based estimator) on a database of 3D models - the *Princeton Shape Benchmark* (PSB) Train partition [81] - described by a set of ten competing FV extractors. The FVs are the PMOM, SD2, H3D, RIN, 3DDFT, CPX, SIL, VOX, and DBF methods (cf. Section 2.3.1). Also included is the DSR FV, which is a statically combined (concatenated) FV compounding four different FVs [97]. We generated Kohonen Maps of dimensionality 12×9 for the database and each of the FV extractors. Figure 3.9 visualizes the distribution of L_1 distances between neighboring SOM prototype vectors using diamond-like diagrams (cf. Figure A.6 in the appendix), for four different FV spaces. We note that we use L_1 as there are results that L_1 may be the most robust of the Minkowski distances for high-dimensional data [3]. In the respective images, brighter (darker) shades correspond to lower (higher) L_1 distances. From left to right, the degree of uniformity of the respective maps' distance distributions increases. While image (a) is dominated by low distances, image (d) consists of a rich mix of different distances. In terms of distance histograms, image (a) is skewed towards low distances, while image (d) approximately resembles a uniform inter-cluster distance distribution. Based on Hypothesis 1, we therefore expect the FV extractor underlying (d) to have best chances to provide good discrimination power, while we expect the converse for the FV extractor underlying (a). The two FV extractors of (b) and (c) should provide medium discrimination power as they show neither uniform nor extremely skewed distance distributions. Note that these assessments are based on unsupervised information automatically extracted from the respective FV spaces.

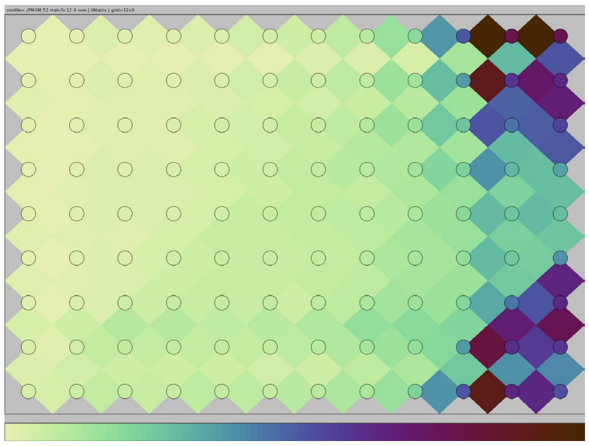
We verified these visually obtained effectiveness estimations by comparing them with benchmarked effectiveness scores obtained using the classification information accompanying the PSB database [81]. Specifically, we considered averaged *R-precision* scores [6] over the PSB in the different FV spaces (cf. also Section 2.3 in this thesis). The R-precision scores for each of the four FV extractors are included in Figure 3.9 and correlate positively with the degree of uniformity of the distance distributions. Please refer to Figure A.7 in the Appendix for U-Matrices for all ten FV spaces.

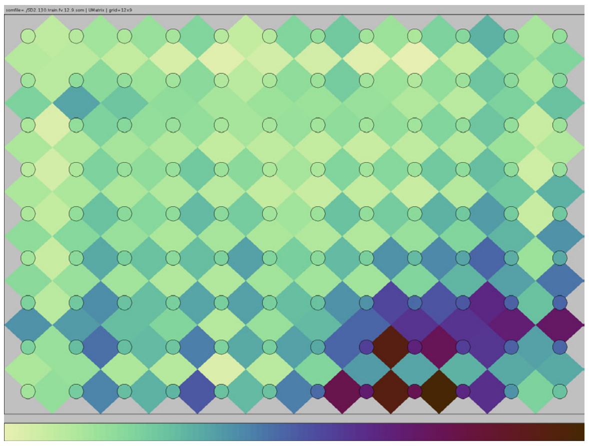
Application of the component-based estimator

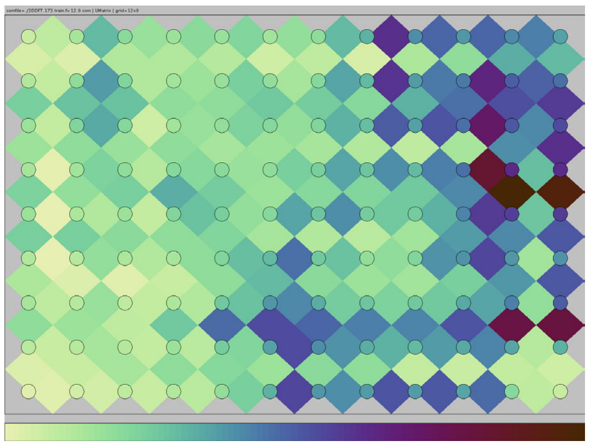
We tested Hypothesis 2 by applying the CPA technique also on the PSB-T data set. Figure 3.10 shows CPAs of four different FV spaces, ordered by increasing R-precision scores (please refer to Figure A.8 in the Appendix for CPAs for all ten FV spaces). Figure (a) contains the worst benchmarked FV extractor from our setting. Its CPA indicates that most components of the prototype vectors are biased towards certain value intervals, with substantial variance in component values only towards the bottom-right area of the CPs. We do not expect such characteristics to provide good chances for meaningful object discrimination. Conversely, image (d) corresponds to the most discriminative FV extractor according to the PSB benchmark. The respective CPA exhibits heterogeneous patterns for almost all components. We therefore are lead to expect good discrimination power.

Images (b) and (c) represent middle-ground situations regarding component heterogeneity. The extractor underlying image (b) exhibits significant variance among roughly the upper half of FV components. The lower half of components seem to be significantly correlated, as the respective CPs show similar patterns. Taking together these facts, we expect moderate discrimination power. A similar situation is present in image (c). About half of the components show significant variance, while the other half of the components represent roughly constant values which cannot meaningfully contribute to object discrimination. In this case, we note that the respective FV extractor was wrongly configured which lead to the observed outcome. Again, taking together both observations leads us to expect moderate discrimination power.

Besides discrimination power estimation, the CPA technique is also helpful in interactive FV engineering. The respective CPAs suggest that the highly correlated or approximately constant components can be aggregated or removed in the FVs underlying CPAs (b) and (c) in Figure 3.10. Doing so should lead to more compact FVs expected to retain the discrimination power provided by the original FVs.


(a) PMOM (15%)


(b) SD2 (18%)


(c) 3DDFT (25%)

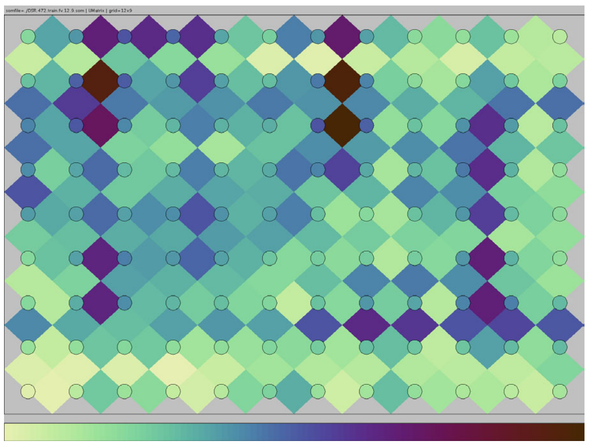
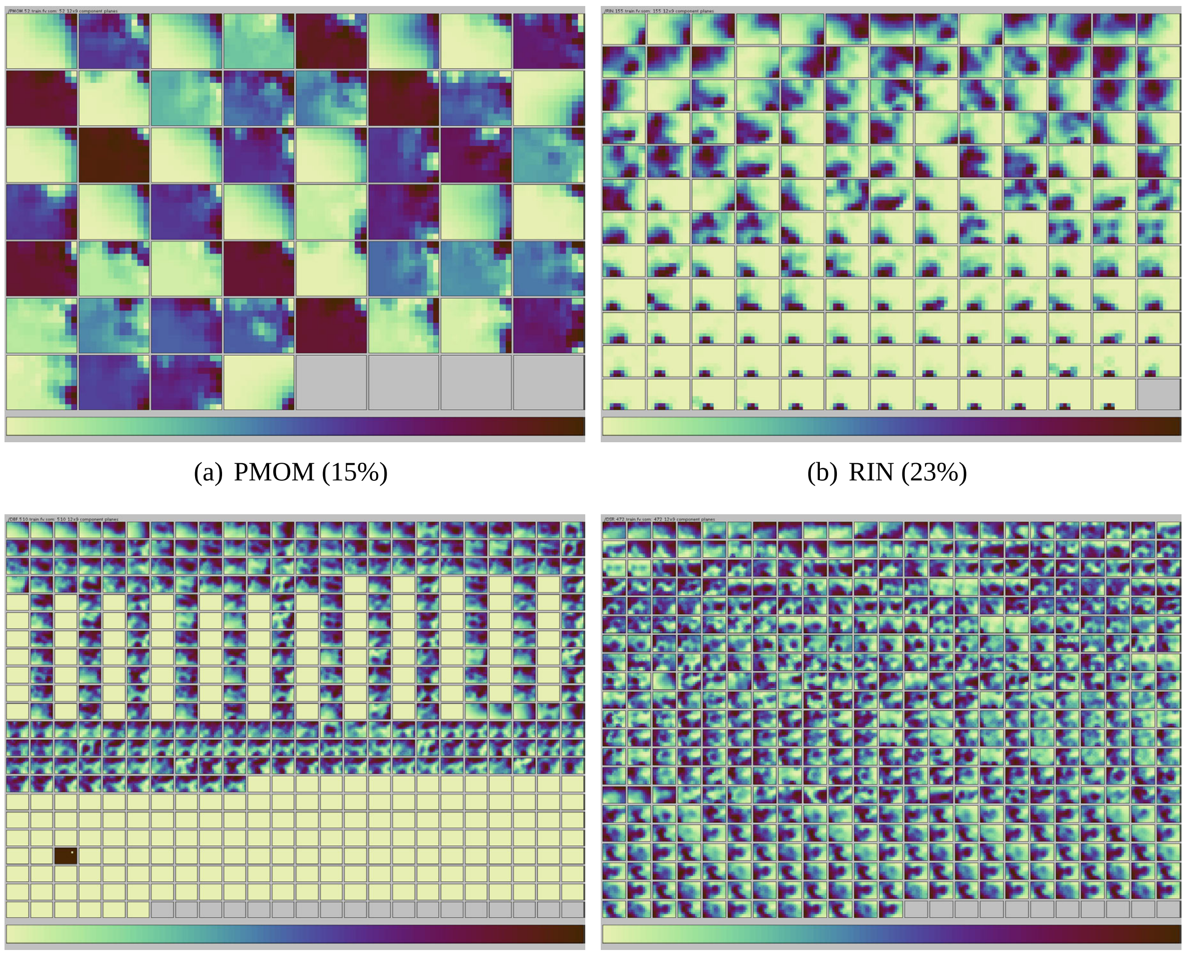

(d) DSR (43%)

Figure 3.9: Visualization of the L_1 distances between adjacent cluster prototypes of Kohonen Maps generated for the PSB-Train database represented in four different feature spaces. Bright (dark) shades correspond to low (high) distances. The degree of uniformity of the respective distance distributions increases from left to right. This is in accordance with the increase of a supervised discrimination precision benchmark score (R-precision, given in brackets).



(a) PMOM (15%)

(b) RIN (23%)

(c) DBF (31%)

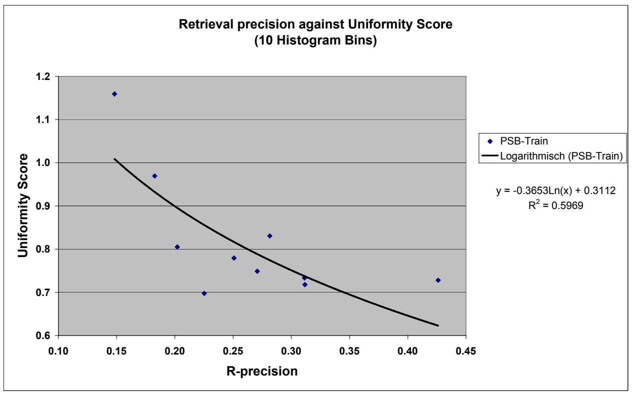
(d) DSR (43%)

Figure 3.10: Component plane arrays for the PSB-Train database represented in four different feature spaces, sorted by benchmarked precision scores. The visualization allows unsupervised selection of prospective FV extractors, and can be used to identify highly correlated or indiscriminating components for removal from the FV. Note that the number of component planes differs among the arrays, as each FV extractor was equipped with a specific, method-dependent dimensionality setting.

3.2.5 Evaluation

Evaluation of the distance-based estimator on real FV data - inter FV space evaluation

We substantiate the practicability of the distance-based estimator by a regression analysis between R-precision scores and degree of uniformity of the Kohonen Map distance distributions given in the ten FV spaces. For each FV space f, we calculate the *uniformity score* $us(h^f) = \sum_{i=1}^{b} |h_i^f - \frac{1}{b}|$ as the L_1-distance between its distance histogram h^f defined over b bins, and the uniform histogram of length b. The lower this score, the more uniform the resulting distance histogram is. Figure 3.11 gives the results of the logarithm model regression analysis for the ten FV extractors using $b = 10$ bin distance histograms. We verify the correlation between the supervised and the unsupervised FV space metric at squared correlation coefficient $R^2 = 0.60$. While this is not a perfect functional dependency, both metrics clearly correlate in the expected sense. We obtained similar results for different bin and Kohonen Map dimensionality settings. We conclude that the proposed analysis is a valid and practical option for addressing the unsupervised FV extractor selection problem. Note that the presented results are an *inter-FV space* analysis in that we contrast different FV spaces with each other in the regression, fixing the dimensionality of each FV space to the optimal dimensionality as determined by benchmarking in Section 2.3.4.



(a) Regression

Feature Vector Extractor	R-precison Score	Uniformity Score
PMOM	14.82%	1.1590
SD2	18.26%	0.9692
H3D	20.20%	0.8051
RIN	22.52%	0.6974
3DDFT	25.08%	0.7795
CPX	27.08%	0.7487
SIL	28.15%	0.8308
VOX	31.13%	0.7333
DBF	31.16%	0.7179
DSR	42.61%	0.7282

(b) Data

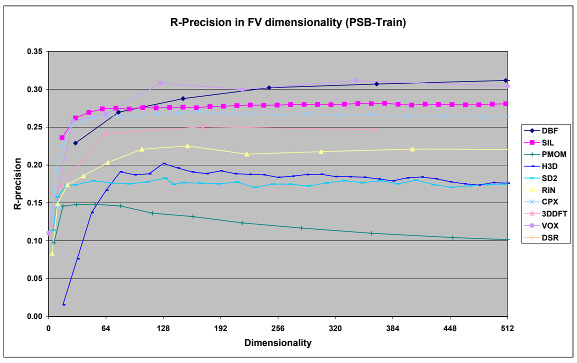
Figure 3.11: Regression analysis between uniformity score of Kohonen Map distance histograms (unsupervised information) and a supervised discrimination precision metric for ten FV extractors. The expected correlation is verified, indicating viability of the analysis for automatic discrimination power estimation.

We give technical details regarding the above experiment. The SOMs were generated using the SOMPAK implementation by the Helsinki University of Technology [60]. The parameters were set as follows: 12×9 rectangular grid; randomly initialized reference vectors; a bubble kernel was used. We performed a two-staged training process. First, 25 DB iterations using radius $r_1 = 15$ and learning rate $\alpha_1 = 0.05$ were performed. Then, 50 DB iterations using radius $r_2 = 5$ and learning rate $\alpha_2 = 0.02$ were performed. For each SOM, the distance histogram was calculated with $b = 10$ bins over the L_1 distances between the reference vectors of all pairs of adjacent SOM nodes. The histogram was equi-width and constructed over the $[d_{min}, d_{max}]$ interval. Many more settings were tried regarding SOM grid dimensionality, histogram size (number of bins), and Minkowski distances L_1, L_2, and L_3, on the PSB-Train dataset. The setting given above is the one reported here and represents the best correlation results. Correlation quality as measured by R^2 on the exponential model was sensitive to the distance (only L_1 gave good correlation). Also, larger and smaller SOM grids gave somewhat deteriorating correlations. The bin setting regarding the setting with a 12×9 SOM grid and using the L_1 norm was rather robust.

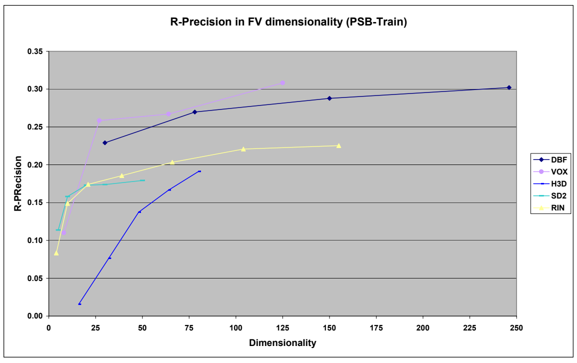
We suppose that the moderately-sized SOM grid and histograms are beneficial in that they provide a smoothing effect suppressing outliers and noise, thereby stabilizing the result. We note that regarding SOM grid size, several rules of thumb are proposed in the literature, e.g., recommending setting the number of nodes to the square root of the database size [60]. We used the 12×9 settings for many different databases of several thousands of objects each, and mostly obtained good clustering results.

Evaluation of the distance-based estimator on real FV data - intra FV space evaluation

It is interesting to ask whether the observed dependencies also hold for an *intra-FV space* analysis. We recall from the experiments in Section 2.3.4 that the FV extractors usually can be configured to different resolution levels, e.g., by setting the granularity of sampling features from the objects. Determining the optimal FV dimensionality is in itself a feature selection problem which is traditionally solved by the benchmarking approach. If the dependency between distance distribution heterogeneity and benchmarked retrieval precisions also holds for the different dimensionalities possible within the different FV spaces, then the estimator should also be useful for the dimensionality selection problem.



(a) All FV spaces



(b) Selected FVs and dimensionalities

Figure 3.12: R-precision as a function of dimensionality for all 10 FV spaces considered in Figure 3.11 (left) and the 5 selected (right) FV spaces, measured on the PSB-Train benchmark.

We therefore consider an intra-FV space regression experiment by plotting R-precision val-

Table 3.1: Dimensionality and R-precision spans for the selected PSB-Train FV spaces. The selection is based on a sufficiently large span in both variables.

FV space	min dim	max dim	min R-prec.	max R-prec.	Num. of dim settings
SIL	15	60	23%	27%	4
RIN	4	155	8%	22%	7
SD2	5	50	11%	17%	5
DBF	30	246	22%	30%	4
H3D	16	80	2%	19%	5

ues obtained for different dimensionality settings in a given FV space against the uniformity scores of the respective SOM spaces. We selected a set of five FV spaces for which we tested the dependency. Specifically, we selected the SIL, RIN, SD2, DBF, and H3D FV extractors, as these (a) allow for a sufficiently fine granularity of available dimensionality settings, and (b) the respective dimensionality settings yield a sufficient span of resulting R-Precision scores. In other words, we required the FV spaces to provide a significant spread in both variables. Figure 3.12 reports the benchmarked R-precision scores for the FV spaces and dimensionalities, and Table 3.1 summarizes the span of dimensionality and R-precision variables of the selected FV spaces.

Figures 3.13 and 3.14 report the results of the uniformity score regression experiments for each of the selected FV spaces. The left columns contain plots of respective R-precision values against uniformity scores. The uniformity scores were calculated using histogram width such that the log regression dependency was maximized w.r.t. the R^2 statistic. The dependency strengths range between 93% for the SIL FV space, and 45% for the H3D FV space. The scatter plots indicate that a correlation exists between the supervised and the unsupervised FV metrics within each FV space and for varying dimensionality. Looking at the scatter plots, we see that selecting the FV dimensionality which minimizes the uniformity score not always manages to choose the optimal FV dimensionality, but gives good selection decisions significantly outperforming the random choice. If we chose the FV dimensionality setting yielding the *lowest* or *second lowest* uniformity scores, we select FV spaces which perform significantly better than the random choice of any FV dimensionality.

Table 3.2 gives the selection results. In both unsupervised FV selection polices, in 4 out of 5 cases, the choice is better than random. The second lowest uniformity score selection rule manages to select the optimal dimensionality setting in 3 out of 5 cases (for the SIL, RIN, and SD2 FV spaces), manages to chose the second best dimensionality in one case (H3D FV space), and performs worse than random only for one FV space (DBF, where it selects only the 3rd performing out of 4 FV dimensionality settings). The policy of choosing the dimensionality setting minimizing the uniformity score performs slightly worse in terms of selection ranks, as compared to the second-lowest selection policy. It manages to detect the best setting once (H3D), and performs better than random for SIL (2nd of 4), RIN (3rd of 7), and DBF (2nd of 4). It under performs the random selection only once in case of SD2, where

Table 3.2: Performance of dimensionality selection policies based on selecting the FV space yielding the lowest or second lowest uniformity score. Both policies perform better than random for 4 out of the 5 FV spaces. The 2nd lowest selection policy hits the optimal dimensionality in 3 out of 5 cases.

FV space	lowest choice	2nd lowest choice
SIL	2/4	1/4
RIN	3/7	1/7
SD2	4/5	1/5
DBF	2/4	3/4
H3D	1/5	2/5

it chooses the 4th best performing out of 5 available dimensionality settings.

We note that this evaluation was based purely on the ranks of selected dimensionality settings. In terms of resulting R-precision scores, the selection policies perform even better, if one compares against the worst case, namely, ending up with the R-precision scores resulting from selecting the worst performing dimensionality. From the scatter plots we see that for the low uniformity score data regions, the associated R-precision scores are rather similar, making the lost precision not so severe if not the optimal dimensionality setting is selected. E.g., the lowest uniformity score selection policy hits only the 4th best out of 5 dimensionality settings in the SD2 space. Still this is a good choice, considering that the 4th best setting yields 15.8% R-precision, while the best choice would yield 17.9%, which implies that not too much R-precision is lost by this choice. What is even more important, is that the worst choice is avoided, which would yield only 11.3%. Generally, the avoidance of the worst case seems to be robust, as the uniformity scores for the lowest realized R-precision observations are significantly higher than those for the better realizations, for all of the FV spaces.

Regarding the robustness of the intra-FV evaluation, we also examined the correlation dependencies while varying the bin size settings for calculating the uniformity scores. The right columns of charts in Figures 3.13 and 3.14 report the R^2 values obtained by varying the bin widths between $b = 2$ and $b = 20$ in the correlation experiments. As converse to the results to be presented for the synthetic data set in the next section (indicating analysis robustness w.r.t. histogram width), the correlation strength seems to depend on the histogram width settings. Basically, we obtain the tightest dependencies for small histogram widths between 3 and 7 bins. Larger histogram widths lead to diminishing dependencies. Also, there occur oscillations in the dependencies, alternating between significant R^2 measures, and insignificant results below 10%. This is an interesting observation which at current is advocated to the simple equi-width binning approach chosen in the definition of the uniformity score. It is presumed that histogram discretizing artifacts account for the fluctuations observed regarding dependency strength. It is also presumed that data-adaptive binning strategies should lead to improved robustness in the histogram-based dimensionality selection heuristic, an idea which is left for future work.

Evaluation of the distance-based estimator on synthetic data

The above findings are interesting as they link unsupervised information extracted from several given 3D FV spaces with respective effectiveness benchmarks. As benchmarks are supervised in nature and expensive to build, unsupervised effectiveness estimation is desirable. We are therefore interested whether these empirical results generalize, and we would like to assess the robustness of these findings. As the number of available high-quality research benchmarks and multimedia FV extractors is limited, we consider additional synthetic data sets. We generated several datasets simulating FV spaces of varying discrimination between the object classes. The specifications of the FV generation are as follows:

- The *dimensionality (dim)* was set to 128, which represents a middle ground resolution regarding many real FV extractors.
- The *database size* was set to contain between 50 and 200 classes (*n_classes*), where each class consists of 50 elements (*c_size*). These sizes seem typical for many multimedia benchmarks.
- The *FVs* were modeled as being normally distributed with standard variance of 1.0 around a given, class-specific center point for all dimensions in FV space.
- The class-specific *class centroids* (center points) were modeled as being uniformly distributed along all dimensions in FV space.
- The degree of *inter-class discrimination* was modeled by varying the interval from which the class centers were drawn (*d_span*) between 1.0 and 5.0.

These settings were chosen as a model of typical multimedia benchmarks. While modeling inter-class discrimination is probably the most difficult part, we believe that using classes with uniformly distributed centroid and normally distributed component values is a reasonable choice. In combination with the dimensionality span chosen from which to draw the class centroids simulates class discrimination which is comparable to real FV benchmarks. We have measured R-precision scores for the synthetic data range, and observed they ranged between 2% and 100% (cf. Figure A.9 in the Appendix).

Regarding the SOM parameter settings, we consider three SOM size settings. In the *equal-sampling* scenario, the number of SOM nodes is roughly equal to the number of classes, so that we expect the respective SOMs to represent each synthetic class by one SOM node. We also considered *over-sampling* and *under-sampling* scenarios, where the number of SOM nodes exceeds or falls short of the number of classes. Considering differently sized SOMs is important for assessing the robustness of the analysis. This is because in automatic FV evaluation we cannot be sure about the number of classes present in a given database (this supervised information is beyond the scope of the analysis). Consequentially, we cannot adjust to SOM grid size to optimize the analysis in case there was some data-dependent best SOM grid size depending on the FV space characteristics. Intuitively, we would expect the number of SOM nodes having to match or exceed the number of classes, as only then the SOM algorithm has a chance to map the classes to the SOM grid in a discriminating way. Table 3.3 summarizes the three experiment settings we devised based on this reasoning.

Table 3.3: Three experimental settings. The data distribution chosen tries to capture important characteristics of real multimedia FV data. The database and SOM sizes model cases where the number of SOM nodes exceeds (matches, falls short of) the number of simulated classes.

Scenario	dim	n_classes	c_size	d_span	grid	nodes per class
over-sampling	128	50	50	$1.0-5.0$	32×24	15.36
equal-sampling	128	100	50	$1.0-5.0$	12×9	1.08
under-sampling	128	200	50	$1.0-5.0$	12×9	0.54

Figures 3.15 (a), (c), and (e) plot the uniformity scores (using $b = 10$ histogram bins) obtained when increasing *d_span* from 1.0 up to 5.0 in steps of 0.2, thereby gradually increasing inter-class discrimination. (c) gives the result for the equal-sampling scenario, where the relation between SOM nodes and object classes is roughly 1 : 1. There is a clear dependency between the two metrics: As discrimination is improved, the uniformity score decreases, indicating the SOM-based distance histograms move towards a uniform distribution. The squared correlation coefficient R^2 amounts to about 85%, indicating a significant correlation between the two metrics. Figures (a) and (e) show the over-sampling and the under-sampling case, where the number of SOM nodes exceeds or falls short of the number of classes in the synthetic data. Interestingly, there is also a significant relation between class discrimination and uniformity score in both scenarios, even if the number of SOM nodes is not sufficient to represent each class with a SOM node (reference vector) of its own as in the under-sampling scenario. The respective R^2 values amount to about 70% for under-sampling (0.5 nodes per class) and 50% for over-sampling (15 nodes per class), respectively. Again, histograms of length 10 were used to calculated the uniformity score for these scenarios.

Besides SOM grid size, another parameter needs to be set for the analysis: The histogram length b used for calculating the uniformity score from the SOM node distances. As we are interested in assessing the influence that b has on the analysis, we performed all regressions while varying b between 2 and 20. Figures 3.15 (b), (d), and (f) report the observed R^2 values for the three scenarios and for varying histogram lengths. For small histogram sizes up to about 5 bins, R^2 is rather low (but increasing). We attribute this to the fact that the small histogram sizes perform too much aggregation, eliminating needed information regarding the distance distributions. For histogram lengths of about 7 to 8 bins and above, we observe stable dependencies between 0.50 and 0.85 R^2 levels for the different scenarios. This is another interesting finding, as it indicates that we do not have to worry too much about finding appropriate histogram length settings, as there is a stable interval of useful settings.

We conclude from the experiments on synthetic data that Hypothesis 1 is quite robust with respect to the SOM grid size, which supports the usability of the discrimination power estimation technique. Also, there is no need to also estimate the number of classes, but we expect reasonable SOM sizes, such as indicated by the well-known rules-of-thumb for SOM generation to deliver good results. We note that we have performed additional experiments on data sets with different dimensionality and d_span settings, obtaining comparable results.

Evaluation of the component-based estimator

Regarding evaluation of the CPA technique, we summarize that the CPA technique allows visual assessment of *variance*, *component-correlation*, and *noise / error* characteristics present among FV space components. In our experiments, we were able to verify these unsupervised, visually obtained assessments using supervised benchmarking results, indicating the usefulness of the CPA tool for FV selection and engineering.

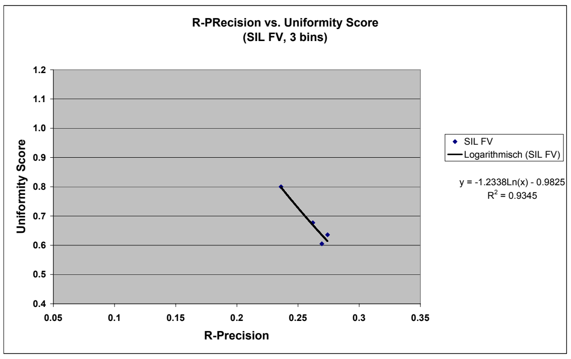
We note that numerically capturing such discussed CPA characteristics is difficult. Recently, there have been efforts to automatically diagnose the effectiveness of visualization results by means of pixel-oriented analysis. So-called *Pixnostics* [45] explores interestingness measures defined on images by appropriately engineering image-analysis functions considering e.g., the degree of variance, structural changes, or block-wise correlations within an image. The goal is to have a scalar interestingness function for solving the parameter choosing problem in generating effective visualizations. It are such interestingness functions that we need to research to automatically leverage the component-based visual FV space analysis proposed in Section 3.2.3. We leave the design of appropriate Pixnostics functions and regression experiments similar to the ones given for the distance-based estimator for future work.

We conclude that the CPA technique complements the distance-based technique, and that both tools offer visual and analytical access to a wealth of useful FV space information.

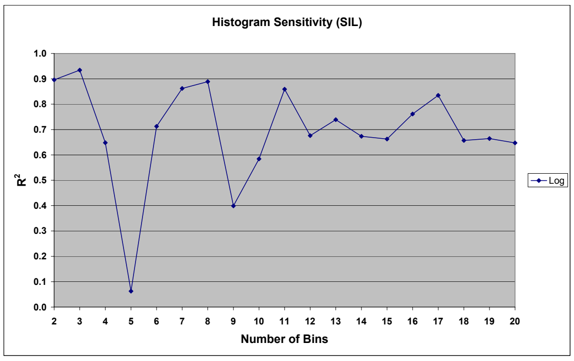
3.2.6 Conclusions

We gave two intuitive hypotheses linking FV space characteristics obtained by unsupervised means with the discrimination power (effectiveness) to expect in the respective FV space. We gave experimental evidence based on real and artificial FV data supporting the hypotheses, and we demonstrated the applicability of two corresponding tools for visual FV space analysis. The tools are proposed to complement the (expensive) supervised benchmarking approach to FV space evaluation, and they are advocated for interactive FV selection and engineering tasks. Specifically, the distance-based discrimination power estimator is suited for automatic, unsupervised FV space selection, where the task is to find the best discriminating FV space in a set of different FV types, or dimensionality settings thereof. With some minor limitations, the experiments showed the automatic unsupervised FV selection heuristic to be robust regarding parameterization. The tools are expected to be helpful in designing improved similarity-based multimedia applications. The tools are specifically useful in cases where no appropriate benchmark is available.

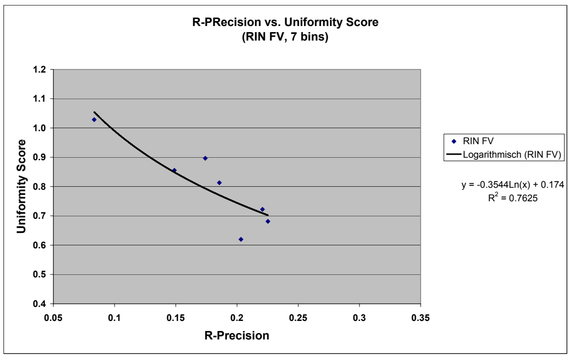
Future work involves exploring and comparing additional unsupervised metrics for FV space discrimination power estimation. Besides the 3D FV domain considered in this Section, we plan to apply the techniques in additional multimedia data domains.



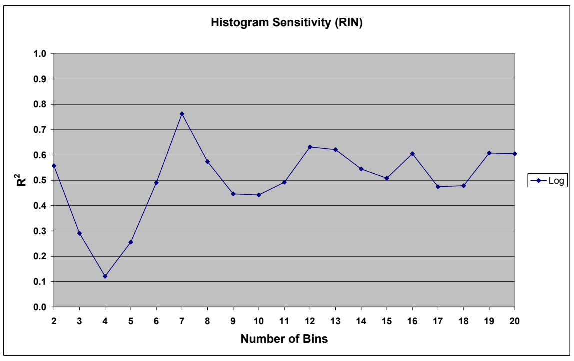
(a) SIL ($R^2 = 93\%, b = 3$)



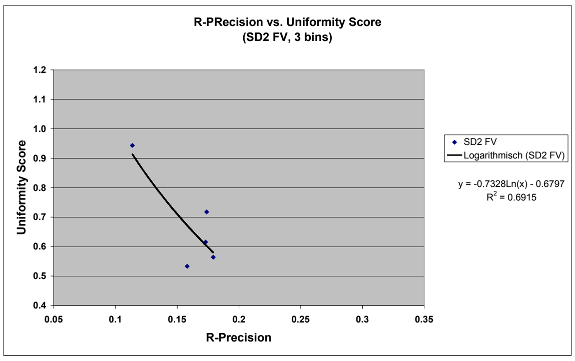
(b) SIL sensitivity



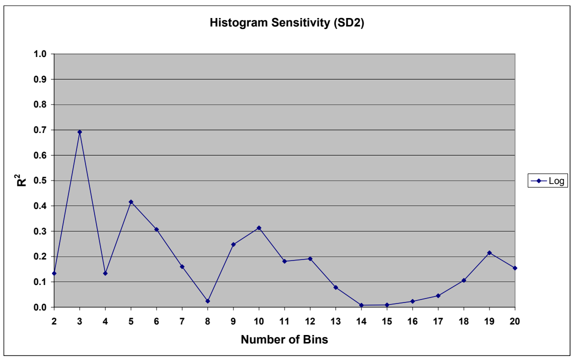
(c) RIN ($R^2 = 76\%, b = 7$)



(d) RIN sensitivity

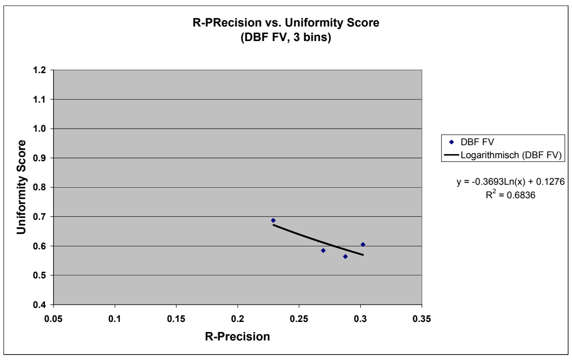


(e) SD2 ($R^2 = 69\%, b = 3$)

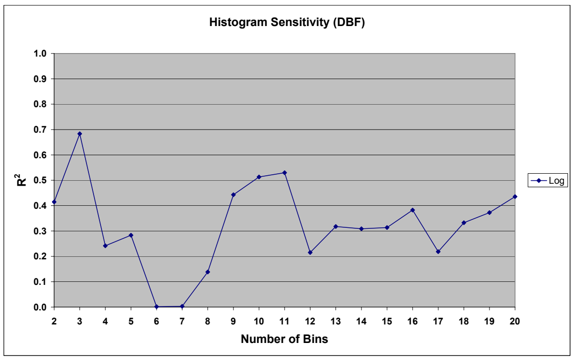


(f) SD2 sensitivity

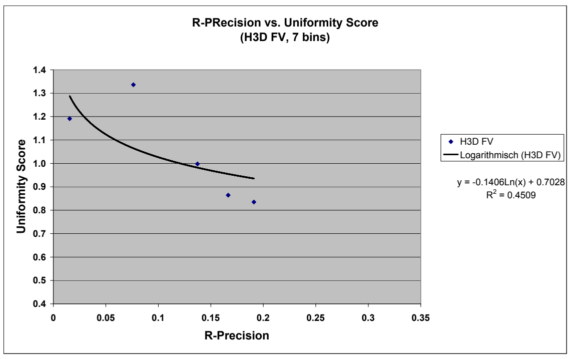
Figure 3.13: Intra-FV space experiments for the SIL, RIN, and SD2 FV spaces in variable dimensionality. The left column of charts gives the strongest dependencies for selected histogram bin widths. The right column gives the correlation sensitivities for histogram widths between 2 and 20 bins.



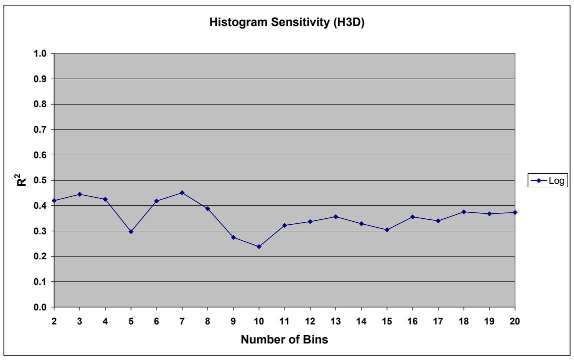
(a) DBF ($R^2 = 68\%, b = 3$)



(b) DBF sensitivity

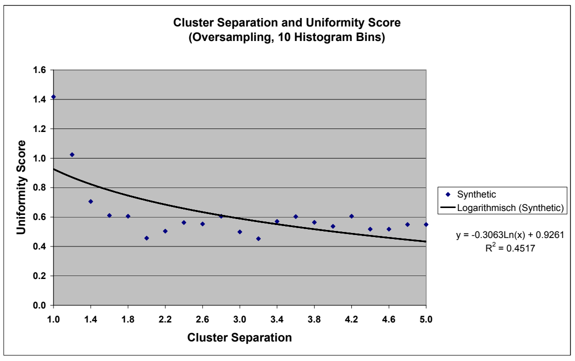


(c) H3D ($R^2 = 45\%, b = 7$)

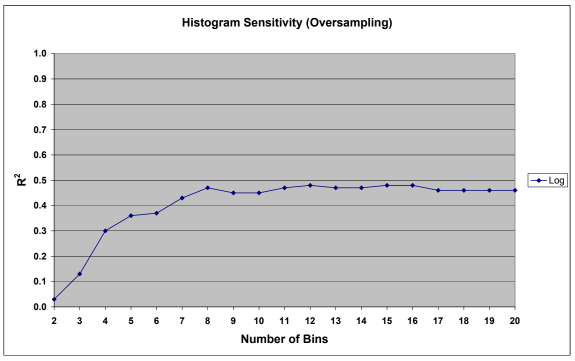


(d) H3D sensitivity

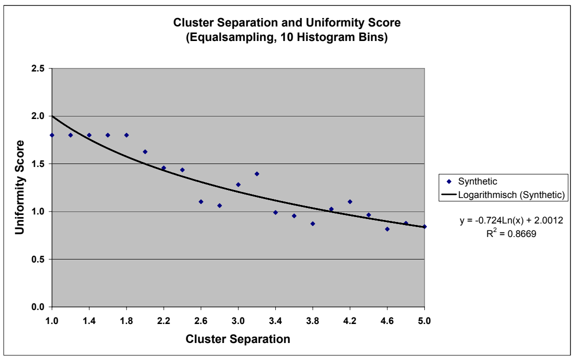
Figure 3.14: Intra-FV space experiments for the DBF and H3D FV spaces in variable dimensionality. The left column of charts gives the strongest dependencies for selected histogram bin widths. The right column gives the correlation sensitivities for histogram widths between 2 and 20 bins.



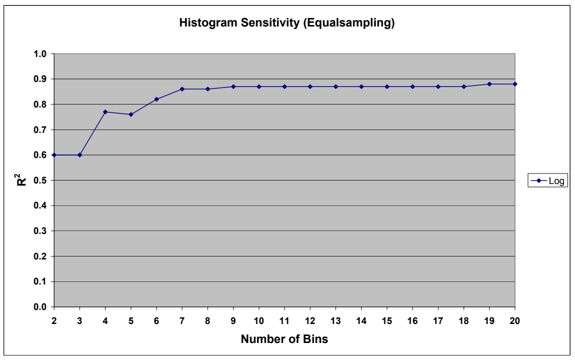
(a) Over-sampling, 20 bins



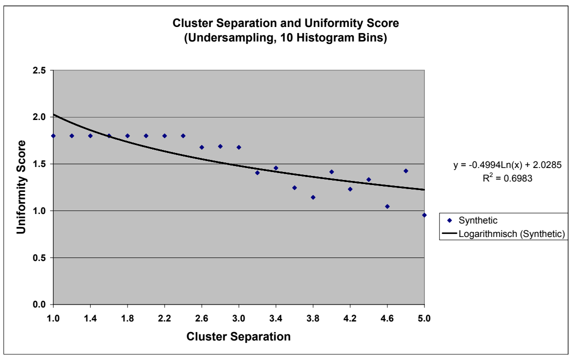
(b) Over-sampling sensitivity



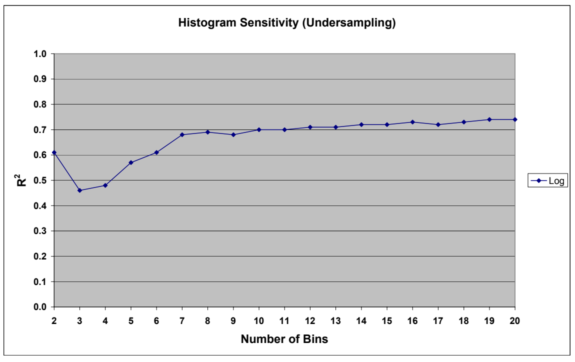
(c) Equal-sampling, 20 bins



(d) Equal-sampling sensitivity



(e) Under-sampling, 20 bins



(f) Under-sampling sensitivity

Figure 3.15: Dependency between uniformity score and discrimination in FV space on synthetic data. (a,c,e): Log regression between rate of discrimination and uniformity score for distance histograms of length 10. (b,d,f): Sensitivity of the dependency for varying histogram lengths in terms of R^2.

3.3 Supervised visual feature space analysis: Convex hulls over class distributions in 2D projected space

3.3.1 Background

Huge and quickly increasing volumes of complex data are collected and archived in many important application domains. To make sense of archives of complex data, the *similarity concept* is of fundamental importance as it allows the application of mining algorithms such as clustering, classification, association, and filtering. Similarity concepts can be implemented by mapping the elements from (possibly complex) object space O to an appropriate metric space $\mathbb{X}$ in which a distance function $d(x,y) \in \mathbb{R}_0^+$, $x,y \in O$ is defined for any pair of objects (x,y). d is interpreted as a scale for the (dis)similarity between objects. Approaches for establishing a metric space $\mathbb{X}$ for an object space O are to implement the distance function d to operate either (a) directly on pairs of objects, or (b) on points in vector space representing the objects. Approach (a) can be implemented by associating $d(x,y)$ with the costs of efficiently transforming object x into object y using a set of predefined edit operations. (b) corresponds to the Feature Vector (FV) approach which extracts characteristic numeric features from the objects forming vectors of real-valued properties, that is, points in FV space (cf. Section 2.1.1). We understand the *effectiveness* of a metric space as the degree of how accurately distances in metric space $\mathbb{X}$ resemble similarity relationships in object space O. Designing effective metric spaces for complex object spaces is difficult, as often different transformation or feature extraction algorithms are possible, and it is a-priori not clear what the best choices are.

Projection-based visualization of metric spaces is a power tool for analyzing key distribution and similarity characteristics among the elements in complex, possibly large data sets. Many different projection methods such as Multidimensional Scaling, Principal Component Analysis, and the Self-Organizing Map algorithm exist for projecting data from metric space $\mathbb{X}$ to display space $\mathbb{R}^k$, $k = 2,3$ (cf. Section 3.3.2). While each projection has specific advantages (and disadvantages) in preserving distances and topology, all of them require effective visualization for the data in projected space. Most visual analysis tasks in projected space include in one way or the other the estimation of shape, size, distribution and overlap of groups of objects. Standard visualization approaches using clouds of symbols do not scale well with growing data set sizes.

In this Section, we present an intuitive and effective novel projection-based visualization approach for discrimination analysis and evaluation. The approach is based on the hull metaphor for visually aggregating sets of points in projected space, and can be used with a variety of different projection techniques.

We next briefly recall several important data analysis applications operating on metric spaces, and discuss basic options for evaluating metric space effectiveness. We also describe two data sets we will use in the remainder of this Section.

Applications in metric space

Many data-driven applications rely on a representation of the input data in an effective metric space to produce meaningful results. In *Content-based Database Retrieval* (cf. Section 2.1), distances between a query object and candidate elements are used to produce answer lists sorted by increasing distance to the query object [33]. Distances in metric space are also required for *Classification* and *Clustering* [37]. In Classification, unknown data instances are assigned the class label of the most similar class, according to a classifier learned from supervised training data. In Clustering, distances between data instances are used to automatically find clusters of similar elements. Also, in *Information Visualization* [22] often similarity relationships among the data objects are exploited for effective image generation.

Due to the complexity associated with many data types, usually it is not clear what the most relevant metric space to use is a-priori. E.g., in the multimedia retrieval domain an abundance of structurally different, complementary FV extractors to chose from is evident: In the image [93] and in the 3D model retrieval [18] domain, each several dozens of competing schemes for mapping objects to metric space have been proposed to date.

Numeric metric space evaluation

The effectiveness of a metric space can be benchmarked if a suitable ground truth classification (*supervised information*) is available. In many domains, reference benchmarks have been designed containing data and supervised classification information. Example benchmarks are the TREC document collection [68] for text retrieval, and the COREL images in image retrieval [65]. Such retrieval-oriented benchmarks can be evaluated with metrics like Precision and Recall [6]. For benchmarking the effectiveness of classification and clustering algorithms, e.g., the UCI Machine Learning Archive [25] provides data sets for machine learning problems from a wealth of application domains.

Supervised benchmarking can be problematic due to the costs associated with building and evaluating benchmarks, and also potential instability and ambiguities [65]. Therefore, *unsupervised* FV space benchmarking is desirable, but still a largely unsolved problem. Certain theoretical approaches exploiting statistical information calculated in FV space exist [2, 42]. These works are of rather theoretical nature and to the best of our knowledge have not been practically leveraged yet.

Visual metric space evaluation

Visual benchmarking is a highly interesting option complementing the numeric benchmarking approach. It aims to support the evaluator in understanding and explaining numeric benchmarking results. To this end, projections to display space are popular. The benchmark objects are mapped to and visualized in 2D (sometimes 3D) using a suitable projection technique, and then class distribution characteristics can be analyzed. Important projection-based analysis tasks include:

- Discovery of interesting inter-class relationships: Which classes are similar, which ones are dissimilar?

- How compact and discriminated are the classes?
- Are there problematic classes which do not separate well, possibly perturbing the discrimination of other classes?
- What is the overall discrimination quality in the given metric space? That is, how good is separation w.r.t. all classes simultaneously?

The visual analysis of such questions can help in *selecting and engineering* of metric spaces to better fit a given application. E.g., in a classification scenario, badly discriminated classes can be identified, and the metric space can be fine-tuned in a subsequential step to improve the identified problems.

Data sets considered

Two data sets will be used in the remainder of this Section. The first is the *ISOLET-5* data set [25] consisting of 1559 samples of the letters 'A' to 'Z' spoken out by different persons, represented in 616-dimensional FV space. The used features consist of a combination of different aural properties extracted from the samples. This FV space provides considerable discrimination power, as precision results of up to 95% have been reported for appropriately trained classifiers [25]. The other data set is the *Princeton Shape Benchmark Train Partition* (PSB-T) [81]. It comes from the 3D model retrieval domain and consists of 907 polyhedral meshes representing real-world objects like animals, humans, vehicles, and so on (cf. Section 2.3.2). We use a subset (in total, 12) of the 3D FV extractors recently proposed [18] for mapping the database into different FV spaces. The individual FV spaces represent geometric properties such as curvature, volumetric- and image-based features of the models and vary in dimensionality (tens to hundreds of dimensions) as well as in average retrieval precision [19].

The ISOLET-5 data set is already represented in a fixed metric space providing good discrimination power. We use it to discuss projection-based visualization techniques and use cases in Sections 3.3.2 and 3.3.4. The PSB-T data set is represented in many different metric spaces. We will use this data set to apply our projection visualization for *comparative visual FV space analysis* in Section 3.3.4, and also for statistical experiments in Section 3.3.6.

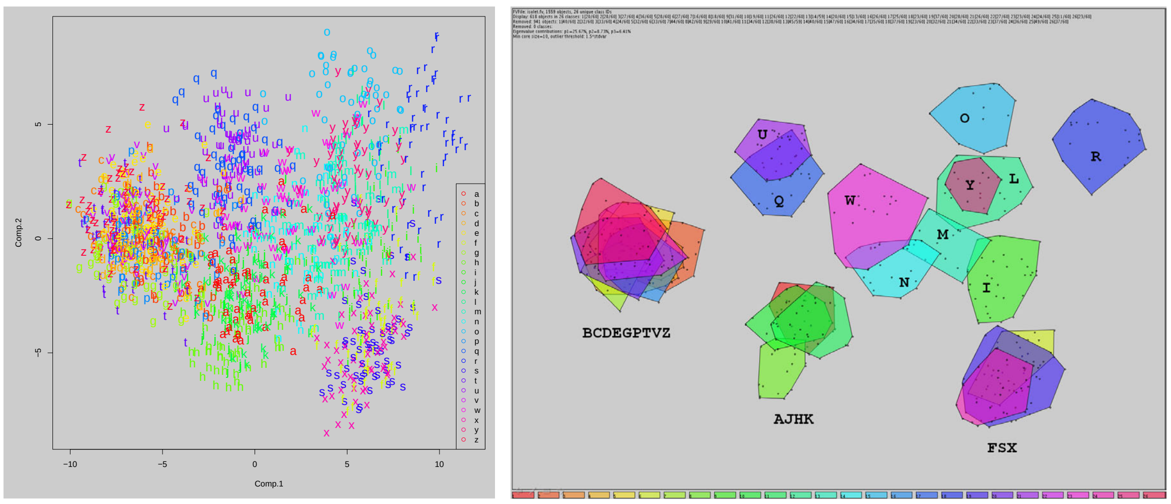
The remainder of this Section is structured as follows. Section 3.3.2 recalls several popular projection techniques, and discusses the symbol clouds projection drawing approach. Based on this discussion, in Section 3.3.3 we motivate and define our convex hull-based visualization technique. In Section 3.3.4, we apply the technique on two data sets, illustrating its effectiveness in a number of important projection-based use cases, including a novel use case: Comparative visual metric space benchmarking. In Section 3.3.5, we present experimental evidence statistically supporting our hull-based metaphor. Finally, Section 3.3.6 summarizes and outlines future work in the area.

3.3.2 Projection and visualization methods

In this Section, we briefly illustrate three well-known projection techniques based on Principal Components Analysis, Multidimensional Scaling, and Self-Organizing Maps. We then point out certain shortcomings of the symbol-based drawing approach usually adopted in projection-based visualization [28].

Principal-Component-based projections

Principal Component Analysis (PCA) [46] is a popular statistical method for summarizing multivariate data by capturing a maximum of data variance in a small number of derived dimensions. *Principal Components* (Principal Axes) are orthonormal directional vectors given by linear combinations of the original dimensions. They effectively form a new base system for the data, based on a rotation of the original base system. The Principal Axes of a d-dimensional data set are found by Eigenvector analysis of the respective $d \times d$ covariance matrix. Sorted by respective Eigenvalue magnitudes, the Principal Axes $p_1, \dots, p_d$ subsequently capture a maximum of remaining variance in the data. By projecting a multivariate data set into the plane formed by the two Eigenvectors with largest Eigenvalues, $p_1 \times p_2$, 2D projections expected to be useful for visual data analysis are generated. Refined PCA-based projection algorithms have been addressed by several authors including optimization for interactive projections [28] and for providing better robustness and class separation properties [62]. Figure 3.16 (a) exemplarily shows the projection of the 616-dimensional ISOLET-5 data set into $p_1 \times p_2$ space using the standard PCA. The class labels of the projected data elements are indicated by colored letters, so called symbol-clouds.



(a) Symbol plot (b) Convex hull plot

Figure 3.16: Projection of the ISOLET-5 data set to the plane spanned by the first two Principal Components of the data set. The classes are visualized using symbol clouds (a). Image (b) gives the convex hull-based visualization of the data at outlier removal level $\varepsilon = 1.5$ with class label annotations (cf. Section 3.3.3).

Multidimensional Scaling

The family of Multidimensional Scaling (MDS) [24] algorithms is another popular projection technique. The basic idea is to find an arrangement of the data elements in a low-dimensional Euclidean output space such that the pairwise distances in the output space resemble the pairwise distances in the given input space as closely as possible. Such an arrangement can be found by diverse iterative algorithms adjusting the element positions to minimize an appropriate global *stress* scale measuring the disagreement of pairwise distances in input and in output space. The MDS projection is applicable on data described in any metric space, not just in vector space as is the case for PCA. As the positioning heuristics run the danger of finding local optima, several MDS projections can be generated and the one with the best stress result be chosen. MDS projections can be visualized analogous to PCA-based projections.

Self-Organizing Maps

The Self-Organizing Map (SOM) algorithm [59] (cf. also Section 3.1.1) is a combined vector quantization and projection algorithm. It represents an input data set described in an input vector space by a set of reference vectors each uniquely located at one node on a low-dimensional regular grid. The reference vectors are fitted to the input data by an iterative learning process, where the nodes compete for data elements. The SOM gives a compressed representation of the input data space, partially capturing topological properties of the input data space on the grid [59]. 2D projections are possible by mapping the input data elements to their best matching SOM reference vector each. Figure 3.17 shows a 12×9 SOM learned from the ISOLET-5 data set. We have visualized the distribution of object classes on the SOM by marking all nodes matched by at least one data element of a given class. The left and right images show the best matching reference vectors for the classes 13 (letter 'N') and 14 (letter 'M'), respectively. Marking SOM grid nodes can give an idea of the distribution of object classes along the grid. Usually, the SOM is configured with much less nodes than input objects. This in turn means that most SOM nodes will represent many objects, often also belonging to different classes. As it is not clear how to define good marking schemes indicating mixes of many classes on low-resolution grids, we feel that grid-marking visualization is rather problematic. An alternative would be to resort to the symbol cloud approach in combination with appropriate rearrangement of coinciding elements for all grid nodes.

Discussion of symbol-based projection visualization

In the standard approach, projected elements are visualized by plotting symbols indicating their class membership, leading to what we like to call *symbol clouds*. Throughout a given display, the symbol cloud should support the *fast, effective, and parallel perception of class membership* by the user to facilitate analysis of class distribution characteristics. The primary visual attributes of a symbol for indicating class membership are color and shape (label). But both attributes are not expected to scale for large numbers of classes and elements. It is assumed that color and shape are limited in effectively discriminating more than a certain number of nominal values due to the human perceptual system [99]. From experiments we

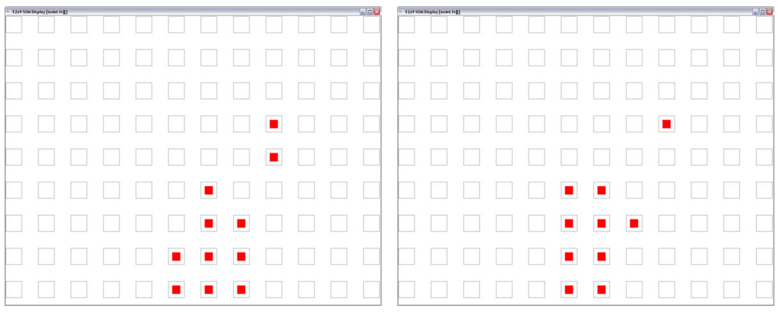

(a) Letter N (b) Letter M

Figure 3.17: Self-organizing maps for the ISOLET-5 data set. The reference vector nodes marked in red are the best matching units to the objects from classes ‘N’ and ‘M’, respectively. Both classes are rather close to each other in FV space, and so are their projections onto the SOM grid.

conclude that depending on distribution characteristics, symbol clouds are not optimal for visually analyzing more than about ten classes. Of course, if all classes are separated perfectly then the problem is not as severe. But as (a) the number of classes increases, (b) the number of elements per class decreases, and (c) the inter-class overlap increases, analyzing class distribution characteristics clearly gets more difficult.

Projection-based class distributions are analyzed primarily along the following characteristics (cf. Section 3.3.1):

- *compactness* of a given class;
- *overlap* of a given class with other classes;
- *separation* between different classes; and
- *shape* of a given class distribution.

The correct visual estimation of these features in a given symbol cloud is difficult, and gets worse in the data set size. Therefore, we next motivate a new approach for visualizing class distributions with better scalability and support for visual estimation of the above mentioned features.

3.3.3 The convex hull metaphor for projection-based visual analysis

Considering the difficulties with symbol clouds, we propose a more abstract, yet useful and empirically supported visualization for analyzing projected class distributions: The *convex hull* shape metaphor. We motivate this metaphor from a discrimination analysis point of view, and show how to easily produce effective visualizations with it. The discussion is done for $\mathbb{R}^2$, but extension to $\mathbb{R}^3$ is straightforward.

Discrimination analysis with shapes

With symbol clouds, the analyst has to first form a mental model describing the shape of each class distribution, and then estimate compactness and overlap metrics based on these shape models. This is not an easy, but rather demanding and ambiguous task. We therefore propose to integrate the shape modeling step into the visualization. To do so, we need to find shapes in projected space appropriately representing the given class distributions. Including shapes in the projection, it should be much easier to visually discriminate many different classes simply by tracking corresponding shape boundaries. Also, quantitative estimation of compactness (shape area), overlap (degree of inter-shape intersection), and separation (distance to other shapes) should become more intuitive.

Many different shapes are possible describing a set of projected points. E.g., we can simply model rectangles or circles either minimally spanning all projected elements (cf. Section 3.3.4), or centered and scaled to reflect mean and deviation statistics in the fashion of a 2D box plot. More sophisticated, we can define models for fitting free form shapes to the element clouds representing certain density characteristics. Having in mind that (a) projections to low-dimensional space usually incur an information loss, and (b) representing point distributions by shapes is an abstraction anyway, we here propose a simple and intuitive shape: The convex hull. It is the smallest convex polygon containing all points in a finite point set. Its perimeter is minimal for all possible enclosing polygons, and it can be computed in $O(n \log n)$, where n is the number of points. By experimenting with different enclosing shapes we found the convex hull to be very effective in visualizing class distribution and overlapping relationships among many classes simultaneously. Per se, the convex hull does not reflect local density properties, and is sensitive to outlier elements. In our visualization, we address this by giving visual clues on the distribution of elements by including element marks inside the hulls, and by applying moderate outlier removal prior to rendering of the hulls. We support overlap perception by rendering the hulls using transparency like in [58], and applying a suitable colormap for distinguishing different classes. We will see in Sections 3.3.4 and 3.3.5 that the convex hull metaphor is an effective, useful visualization as justified by application and statistical results.

We note that also in [46] convex hulls were used ad-hoc for indicating class membership of points in projected space. We state that our work contributes beyond simple hull-based diagram drawing as we support large data set sizes via outlier removal preprocessing and transparent layering. We also support visualization of an additional attribute in the display, and we present statistical justification for the convex hull metaphor. All of this has not been done previously to the best of our knowledge.

Convex hull-based class visualization

We here describe the ingredients of the convex hull-based visualization approach. Let

$$D = \bigcup_{c=1}^{|C|} D_c \ \subset \mathbb{X}$$

denote a set of data elements in space $\mathbb{X}$, where $\mathbb{X}$ can be any metric space, or more specifically, a high dimensional vector space $\mathbb{R}^d,\ d > 3$. Let D be partitioned into a set C of object classes, where each class $c \in C$ contains a number $|c|$ of distinct data elements. Furthermore, let

$$P : \mathbb{X} \to \mathbb{R}^k,\ k = 2,3$$

be a suitable projection function such as PCA or MDS mapping the elements from $\mathbb{X}$ to $\mathbb{R}^2$ or $\mathbb{R}^3$. Let

$$T : S \to S',\ S' \subseteq S,\ S \in \mathbb{R}^k,\ k = 2,3$$

be an appropriate thinning function for removing outliers from sets S of (projected) data elements. Let $H(S)$ be a convex hull generator such as *Graham Scan* [36] operating on sets S of points in projected space. Finally, let CM be an array of at least $|C|$ distinct, appropriate colors (colormap). An implementation for drawing convex hull-based projections is then given in Algorithm 1.

We note that in general, the class partitioning scheme C must not necessarily be obtained using supervised information. Often, such a partitioning can be effectively obtained by an appropriate clustering preprocessing step.

Input: Data set $D \in \mathbb{X}$, Projection algorithm P, convex hull generator H, thinning function T, color map CM.

```
 1: perform the projection: $D^p \leftarrow P(D)$
 2: clear the display
 3: /* loop all classes */
 4: for $c \in C$ do
 5:     perform outlier removal: $T_c^p \leftarrow T(D_c^p)$
 6:     /* loop remaining class members */
 7:     for $o \in T_c^p$ do
 8:         mark $o$ in the display
 9:     end for
10:     find convex hull: $H_c^p \leftarrow H(T_c^p)$
11:     fill $H_c^p$ using color $CM[c]$ with alpha blending
12: end for
```

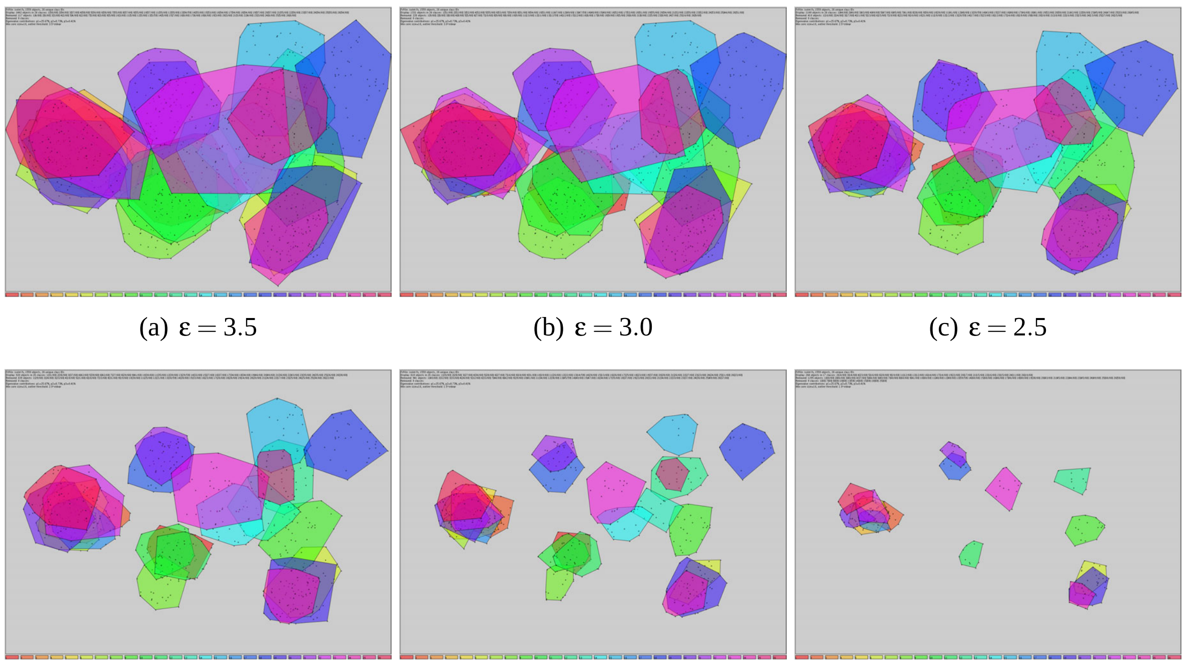
Output: Display of convex hulls over projected and thinned class distributions.

Algorithm 1: Hull-based class visualization

3.3.4 Application

We now apply our convex-hull based projection visualization approach on two data sets, demonstrating the usefulness of the technique for visual analysis in a number of use cases.

Global visual discrimination analysis



(a) ε = 3.5 (b) ε = 3.0 (c) ε = 2.5

(d) ε = 2.0 (e) ε = 1.5 (f) ε = 1.0

Figure 3.18: Convex hulls over PCA-based 2D projections of the ISOLET-5 data set. From top-left to bottom-right, outlier removal is done more aggressive. The convex hull visualization metaphor is an intuitive and effective tool supporting a number of important projection-based analysis tasks. Supported by application examples and statistical results, we advocate it over the symbol-based projection approach.

Figure 3.18 shows the application of the convex hull visualization on the ISOLET-5 data set. We projected the 1559 616-dimensional data elements onto the $p_1 \times p_2$ plane obtained by PCA analysis of the data set (the first three principal axes explain 26%, 9%, and 6% of overall data variance, respectively). We then thinned each class cloud by removing all elements more distant to their respective class centroid than a multiple ε of class-specific standard deviation, as measured in projected space. Specifically, we applied thinning thresholds $\varepsilon = \{3.5,\ 3.0,\ 2.5,\ 2.0,\ 1.5\ 1.0\}$ retaining 92%, 85% , 74%, 59%, 40% and 17% of the original data elements, respectively. We finally rendered the convex hulls using selected colors with a 50% transparency setting. The polygon colors were selected by equal-spaced sampling of the *rainbow* colormap in order to obtain discriminating colors for each polygon.

From the display, typical discrimination characteristics in the ISOLET-5 data set can be easily read (cf. Figure 3.16 (b) for a larger and annotated Figure). The convex hulls allow quick perception of a number of overlapping hull clusters representing e.g., letter groups {BCDEGPTVZ}, {AJHK}, and {FSX}. Distinguishing of classes even with multiple overlaps is nicely supported by the transparently filled convex hulls. Note that the same data is also visualized in Figure 3.16 (a) using the symbol clouds approach. Clearly, the convex hull visualization is more effective in indicating compactness, separation, and overlapping relationships present among the projected data. In our implementation, the class labels can either be read from the colormap legend included in the visualization, or via mouse-over functionality in the visualization. In addition to analyzing static projections, the user can also animate the display by cycling though intervals of outlier removal thresholds to better understand the compactness characteristics of the class distributions. E.g., by going from moderate to more aggressive thinning, the hulls are shrunk in size. By comparing the different 'shrink patterns', the user can easily get an assessment of the compactness and outlier characteristics of the projected classes.

Class contrast plots

The convex hull visualization is useful for contrasting individual classes which might be specially important in an application. Figure 3.19 shows the convex hull-based projections of the PSB-T benchmark classes in two different FV spaces. We have outlined non-filled hulls of classes No. 23 and 48. The classes' discrimination benchmark scores in the two FV spaces are roughly converse to each other (cf. Table 3.4): While the "3DDFT" FV space provides a good discrimination score of 60% for class 48 (3D models of shelves objects), it produces a low score for class 23 (3D models of swords objects). The converse situation holds for the "DBF" FV space. These benchmark scores can be visually confirmed by the convex hull diagrams. In the projected "3DDFT" FV space (Figure 3.19 (a)) class 48 is quite compact and shows little overlap with neighboring classes. On the other hand, class 23 is much less compact with significant overlap with neighboring classes. Roughly the opposite situation is given for the two classes in the "DBF" FV space (cf. Figure 3.19 (b)).

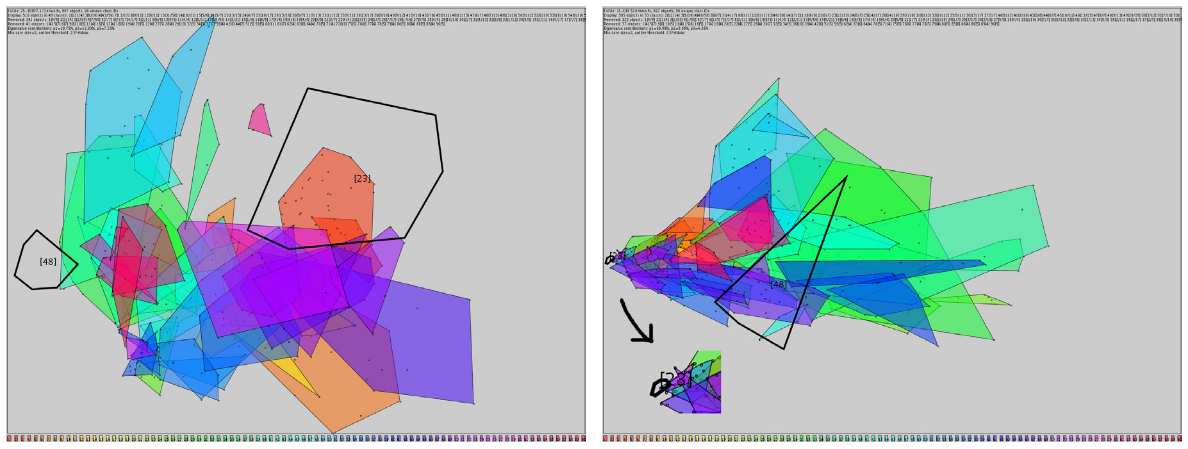
Besides visually explaining class-specific benchmark scores, contrast plots are helpful in analyzing the root causes of class-based discrimination quality: Which other classes share overlap with a given class? How many different classes interfere? Results of such analysis can be fed back into the metric space design process for better supporting problematic classes.

Table 3.4: Discrimination precision benchmark scores for PSB-T classes 23 and 48 under the "DBF" and "3DDFT" FV extractors, respectively. Clearly, higher benchmark scores correlate to more compact and less overlapped class hulls in Figure 3.19.

Class-ID	Name	Size	3DDFT-Score	DBF-Score
23	Swords	15	18%	64%
48	Shelves	13	60%	30%

Integrating additional information

Up to special cases, the projections $P : \mathbb{X} \rightarrow \mathbb{R}^k,\ k = 2,3$ incur a loss of information regarding preservation of topology and distances. While this is an inherent problem of projecting to lower-dimensional spaces, we can enrich our visualization by additional clues on lost or suppressed information, making the projected display more complete regarding the original data

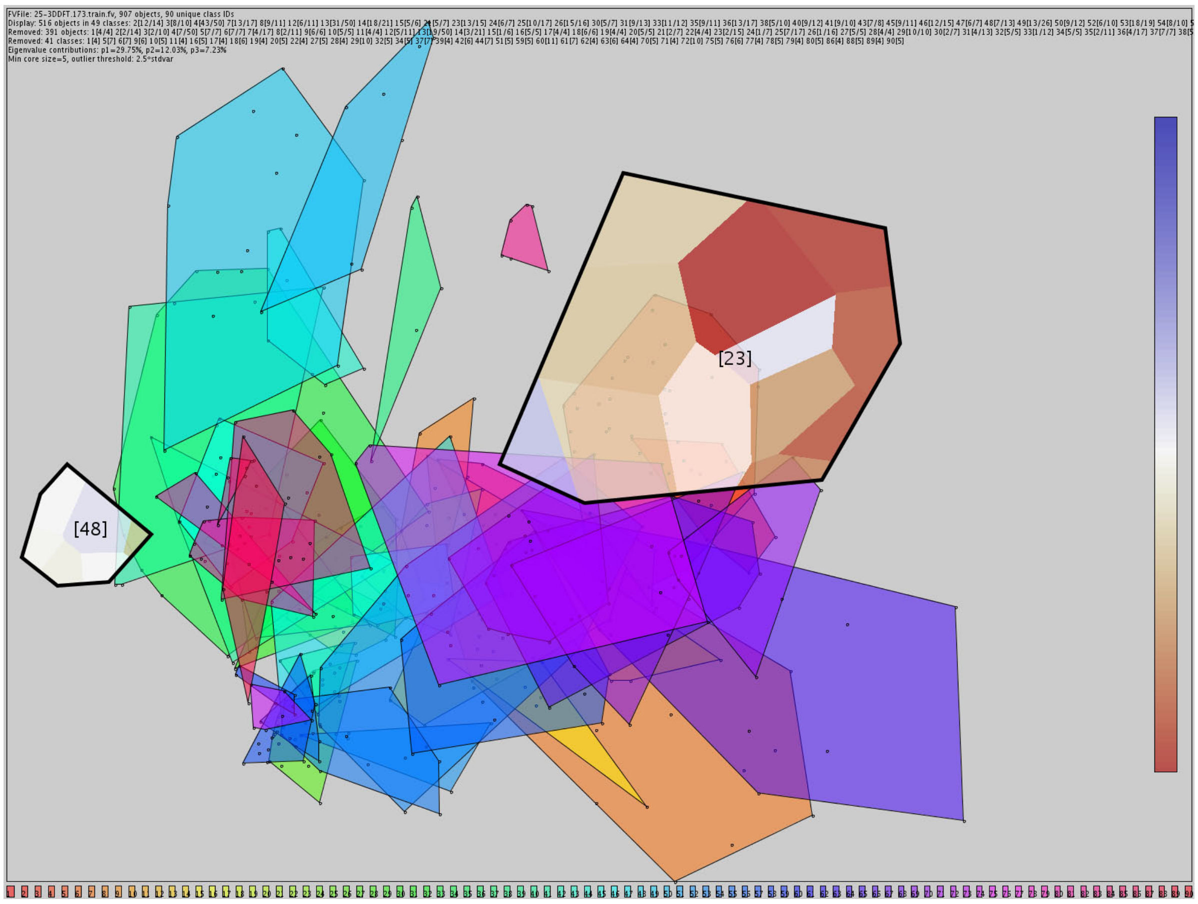


(a) 3DDFT (b) DBF

Figure 3.19: Contrasting the discrimination of two PSB-T object classes in the “3DDFT” (a) and “DBF” (b) FV spaces. The plots allow visual verification of numeric benchmark scores and identification of potentially problematic classes for fine-tuning the respective metric spaces.

space. With PCA-based projections, a natural extension of the (2D) display is to include the third principal axis p_3. It explains the maximal portion of remaining data variance among any of the remaining principal axes. Specifically, we propose integrating the p_3 information by first performing the *Voronoi* decomposition of a given convex hull based on its class elements. Then, each Voronoi cell indicates the respective elements’s p_3 value by transparently filling the cell using a (secondary) colormap defined linearly along the p_3 axis spread $[p_3^{min}, p_3^{max}]$. Figure 3.20 shows the projected FV space from Figure 3.19 (a). The convex hulls of classes 23 and 48 are Voronoi tessellated and the p_3 distribution is visualized, allowing further visual exploration of class-specific discrimination differences. For the already compact convex hull of class 48, we see that the elements are also compactly located along the p_3 axis. Specifically, we perceive that the p_3 components of the elements (a) are similar to each other (the cells are colored-coded homogeneously) and (b) positioned around a middle-ground p_3 value (the cells are color-coded in white to pale shades; cf. the bipolar color scale included in Figure 3.20). Conversely, class 23 not only has a less compact convex hull with significant overlap in $p_1 \times p_2$ space, but the element’s p_3 components show significant variance. This is clear as the Voronoi cells are color-coded rather heterogeneously, indicating significant spread along p_3 towards the lower $[p_3^{min}, p_3^{max}]$ interval.

Based on the concrete projection P employed, other additional metrics are candidates for inclusion in the display. Consider the MDS projection. There is no p_3 axis but instead we could visualize an appropriate local measure of projection quality, e.g., local MDS stress values. We note that Voronoi-based color-coding potentially reduces the scalability of the display w.r.t. the number of hulls a user can effectively distinguish visually, as the additional cell coloring interferes with the transparency-based indication of overlapping hull regions. We therefore advocate the selective application of cell-based visual hints for a limited number of classes



```
FVFile: 25-3DDFT.173.train.fv, 907 objects, 90 unique class IDs
Display: 516 objects in 49 classes: 2[12/14] 3[8/10] 4[43/50] 7[13/17] 8[9/11] 12[6/11] 13[31/50] 14[18/21] 15[5/6] 21[5/7] 23[13/15] 24[6/7] 25[10/17] 26[15/16] 30[5/7] 31[9/13] 33[11/12] 35[9/11] 36[13/17] 38[5/10] 40[9/12] 41[9/10] 43[7/8] 45[9/11] 46[12/15] 47[6/7] 48[7/13] 49[13/26] 50[9/12] 52[6/10] 53[18/19] 54[8/10] 5
Removed: 391 objects: 1[4/4] 2[2/14] 3[2/10] 4[7/50] 5[7/7] 6[7/7] 7[4/17] 8[2/11] 9[6/6] 10[5/5] 11[4/4] 12[5/11] 13[19/50] 14[3/21] 15[1/6] 16[5/5] 17[4/4] 18[6/6] 19[4/4] 20[5/5] 21[2/7] 22[4/4] 23[2/15] 24[1/7] 25[7/17] 26[1/16] 27[5/5] 28[4/4] 29[10/10] 30[2/7] 31[4/13] 32[5/5] 33[1/12] 34[5/5] 35[2/11] 36[4/17] 37[7/7] 38[5
Removed: 41 classes: 1[4] 5[7] 6[7] 9[6] 10[5] 11[4] 16[5] 17[4] 18[6] 19[4] 20[5] 22[4] 27[5] 28[4] 29[10] 32[5] 34[5] 37[7] 39[4] 42[6] 44[7] 51[5] 59[5] 60[11] 61[7] 62[4] 63[6] 64[4] 70[5] 71[4] 72[10] 75[5] 76[6] 77[4] 78[5] 79[4] 80[5] 86[4] 88[5] 89[4] 90[5]
Eigenvalue contributions: p1=29.75%, p2=12.03%, p3=7.23%
Min core size=5, outlier threshold: 2.5*stdvar
```

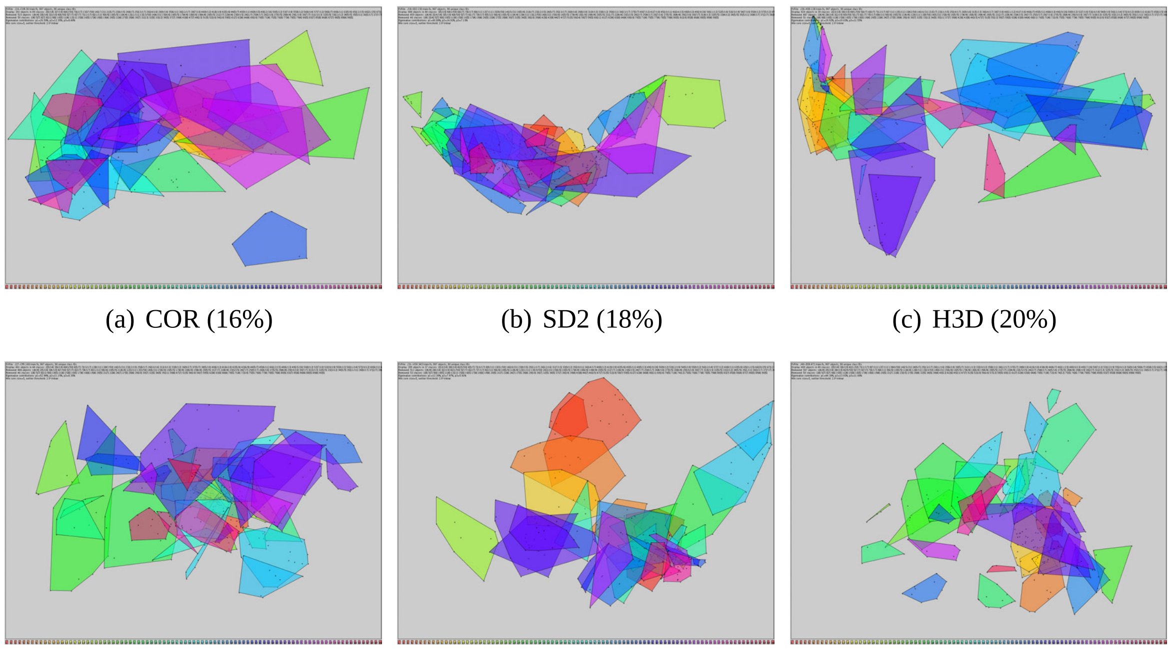
Figure 3.20: Visualizing the element distribution along the third principal axis p_3 for selected classes by a color-coded Voronoi partition of respective convex hulls (cf. Figure 3.19 (a)). In the data set, the Principal Components p_1, p_2 and p_3 together explain $29\% + 12\% + 7\% = 48\%$ of total variance.

simultaneously. This can be controlled by mouse-over functionality, or by automatic determination of a subset of classes showing the most interesting patterns w.r.t. the selected metric.

Comparative FV space analysis

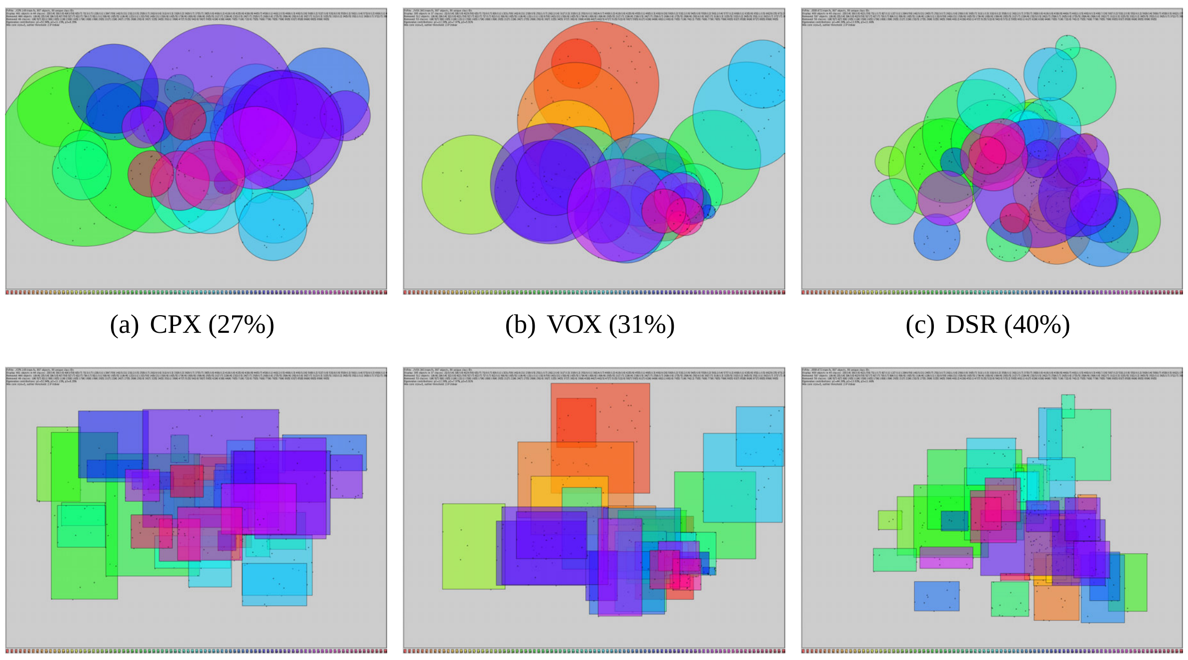
In many domains it is not clear how to best design appropriate metric spaces, but often, different choices are possible for a given data type and application. Then, benchmarking assists in identification of the best choices. Convex hull-based projected space visualization is well suited for comparative visual benchmarking: It provides compact assessments of the discrimination power we can expect for a benchmark in competing metric spaces. Figure 3.21 shows the convex hulls over all PCA-projected PSB-T benchmark classes consisting of at least 5 elements after outlier removal at $\varepsilon = 2.0$, for a number of different metric spaces. The images are sorted by increasingly better benchmark scores. It is interesting to correlate visual features of the convex hull displays with respective benchmark scores: The more compact, less overlapping, and better separated the convex hulls are, the higher the respective numeric benchmark results get. Besides purely interactive FV space evaluation, we can also numerically capture such hull distribution features for automatic evaluation of visually motivated discrimination benchmarks (cf. Section 3.3.5).

For comparison, we also visualized the PSB-T metric spaces using minimum bounding discs and rectangles to visually aggregate the class distributions. Figure 3.22 shows the last three projections from Figure 3.21 using those metaphors (cf. Figure A.11 in the Appendix for all six projected spaces simultaneously.). While the three given metric spaces provide the most compact and well discriminated projections in the example, the displays are still highly cluttered, making it more difficult to visually distinguish individual class distributions, compared to the convex hull approach. This is due to the fact that minimal discs and rectangles tend to *overemphasize* the size of the distributions, and also, the respective shape boundaries by definition are more homogeneous, therefore less visually discriminating to the user.


(a) COR (16%) (b) SD2 (18%) (c) H3D (20%)

(d) CPX (27%) (e) VOX (31%) (f) DSR (40%)

Figure 3.21: Convex hulls over the PCA-projection of the PSB-T benchmark classes after outlier removal, obtained from 6 different FV spaces. The images are sorted by increasing average benchmark scores (indicated in brackets), indicating better discrimination in original FV space. Visual attributes of the convex hulls in projected space such as *compactness*, *spread* and *overlap* clearly correlate with the observed benchmark scores.



(a) CPX (27%) (b) VOX (31%) (c) DSR (40%)

(d) CPX (27%) (e) VOX (31%) (f) DSR (40%)

Figure 3.22: The last three projections from in Figure 3.21, using minimum bounding discs (top row) and rectangles (bottom row). The displays are much more cluttered, making it more difficult to visually distinguish individual class distributions.

3.3.5 Evaluation

In Sections 3.3.3 and 3.3.4, we have motivated and applied the convex hull metaphor for visual discrimination analysis. The underlying assumption was that compactness, overlap, and cluster separation metrics visually obtained from convex hulls formed over the class elements in projected space $\mathbb{R}^k$, $k = 2,3$ may be used to assess the class discrimination quality in metric space $\mathbb{X}$. The justification of the convex hull approach depends on the correctness of this assumption, which in turn depends on:

(a) The fidelity by which the projection P preserves distance and topological relationships among the data elements in metric space $\mathbb{X}$; and

(b) The effectiveness of the convex hull shape in aggregating class distribution characteristics in projected space.

We here cannot discuss factor (a) in light of the wealth of projection algorithms available, but stick to the PCA-based projection, noting that it is a linear projection preserving a maximum of variance information in projected space. Concerning factor (b), we regard the application discussion in Section 3.3.4 as empirical evidence supporting the usefulness of the convex hull approach. We also gathered statistical evidence supporting the convex hull metaphor by correlating benchmark-based discrimination metrics obtained in original space with certain discrimination metrics defined over convex hulls in projected space. Specifically, we defined:

- *Relative area metric*: The area (size) of a convex convex hull as the fraction of total projection space covered by the hull. We evaluate this metric by the number of pixels a given hull covers in the display, among all display pixels.
- *Overlap metric*: The degree of overlap of a given convex hull with other hulls in the display. We evaluate this metric by averaging the number of hulls intersected by the given hull, for every pixel of the given hull in the display.
- *Silhouette metric*: This metric corresponds to the silhouette coefficient as defined in [48] of a given hull. Briefly, this is a distance-based measure rating the average separation of the hull members from their nearest neighbor hull.

We scaled these metrics such that larger values indicate the hulls to be larger, more overlapping, and less separating - that is, less discriminating according to common visual interpretation. We then performed regression experiments using these hull metrics as regressors to explain benchmark-based discrimination scores obtained under the PSB-T benchmark data set represented in 12 different FV spaces. We set up two different regression experiments:

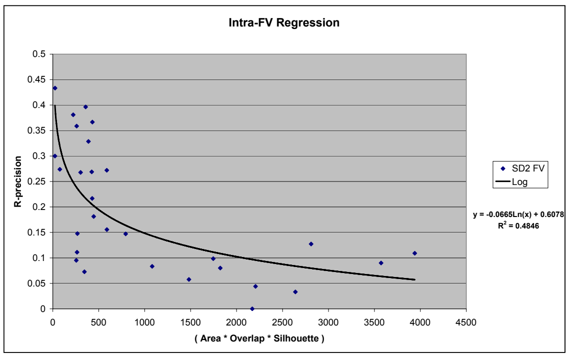
- *Intra-FV experiments*. We fix the FV space and explain the class-specific benchmark scores by hull-based compactness metrics along all benchmark classes in the given FV space.
- *Inter-FV experiments*. We fix the benchmark class and explain the class specific benchmark scores by hull-based compactness metrics along all FV spaces for the given benchmark class.

The compactness metrics were calculated from the convex hulls of all benchmark classes which after medium outlier removal contained at least 4 elements. The projected space consisted of a 800×800 pixel display. We first give exemplary regression results, before we discuss sensitivity aspects. The results indicate a clear dependency between convex hull-based compactness metrics in projected space on one hand, and discrimination benchmark scores on the other hand.

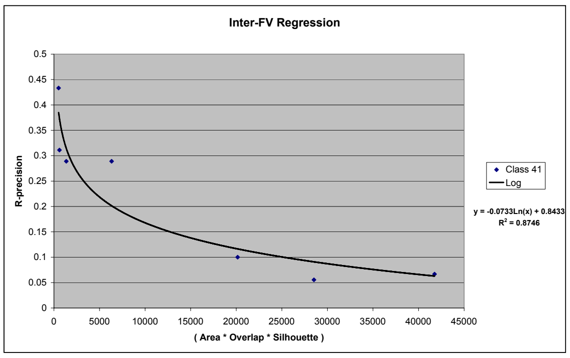
Exemplary regression results

Figure 3.23 shows exemplary regression results obtained when setting outlier removal level $\varepsilon = 1.5$ (see below for a sensitivity analysis). Chart (a) gives an *intra-FV* analysis. It plots the compactness scores of the convex hulls of the PSB-T classes in the projected SD2 FV space against the R-precision benchmark score for the respective classes in original FV space. The compactness scores were calculated by evaluating and multiplying the area, overlap, and silhouette metrics as defined above, for each of the convex hulls in the projected SD2 FV space. We observe a logarithm dependency of both metrics at $R^2 = 48\%$. While this is not a perfect dependency, the metrics clearly correlate: The lower the hull compactness scores (indicating higher hull compactness/separation), the higher the benchmarked discrimination scores are.

Figure 3.23 (b) shows an exemplary *inter-FV* regression. We have plotted the combined regressor against respective R-precision scores for the benchmark class 41, along all FV spaces for which there remained at least 4 projected class members after outlier removal. We verify the dependency at $R^2 = 87\%$. This represents a significant dependency between the visual and the numeric discrimination power metrics for this class and parameter setting, which again is evidence for the practicability of the convex hull-based visual discrimination analysis.



(a) Intra-FV analysis



(b) Inter-FV analysis

Figure 3.23: Regression between compactness of convex hulls in projected space, and respective benchmark discrimination scores calculated in original space. Both metrics significantly correlate in the expected sense, substantiating the usability of the convex hull approach.

Intra-FV regression sensitivities

We performed a series of regression experiments to understand the sensitivity of the relationship between benchmark scores in FV space and compactness of convex hulls in projected space. First, we considered the intra-FV analysis setup. For each of the 12 FV-spaces, we report the R^2 results of the logarithm regression between the hull compactness metrics and R-precision scores, for all benchmark classes which after outlier removal consisted of at least 4 elements. We varied the outlier removal threshold ε between 1.5 and 4.0. Charts (a) to (c) in Figure 3.24 report the R^2 results for the overlap, the area, and the silhouette regressors, respectively. We observe that the overlap regressor (chart a) yields the lowest dependency strength around $R^2 = 10\%$ for most of the FV spaces. An exception are the 3DDFT and VOX FV spaces, where R^2 is around 15%. The area (chart b) regressor delivers better dependencies roughly between 20% and 40% R^2 for the different FV spaces. Similar dependency strength is found for the silhouette regressor (chart c). For all regressors, the dependency strength seems to be rather robust in terms of the ε setting.

We also formed hybrid regressors by combining the individual compactness metrics. To this end, we normalized and multiplied individual compactness metrics. The combined regressors combine information about hull compactness (area metric) and separation (overlap and silhouette metrics), and are expected to add stability to the regression. Chart (d) in in Figure

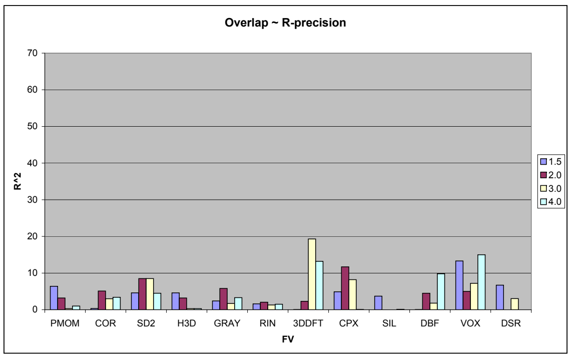
3.24 reports the results when the area and silhouette regressors are combined. The respective R^2 results are significantly improved over each of the single regressors, yielding around 50% R^2 for a majority of the FV spaces. An even better improvement is observed when all three regressors are combined (chart e). This is interesting, as here also the overlap regressor is included, which on its own has only low explanation power (chart a). We also observe that again, the dependencies are rather robust w.r.t. the different ε settings. We finally observe that the dependency strengths are rather similar for the individual FV spaces, with the exceptions of the PMOM and COR FV spaces. In these FV spaces, the dependencies are the lowest among all FV spaces and experiment settings. Also, the PMOM and COR FV spaces yield the lowest benchmarked discrimination power in original space.

Inter-FV regression sensitivities

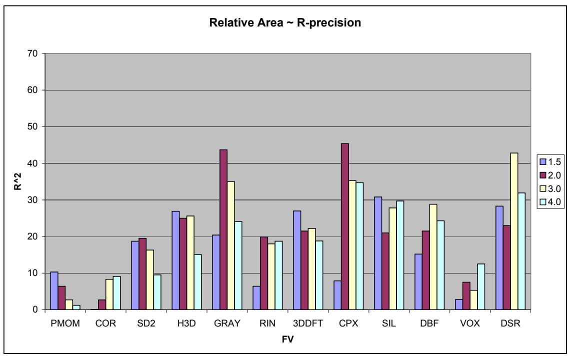
We also looked at the correlation strengths for each of the individual benchmark classes of the PSB-T database (inter-FV correlation experiments). Having in mind the results from Figure 3.24, we fixed $\varepsilon = 1.5$ for the experiment series, and we removed the PMOM FV space from the analysis, as this space showed the lowest overall dependency in the regression. As relevant classes we chose those for which at the given outlier removal threshold there were at least 4 FV spaces for which in the respective class projections there remained at least 4 convex hull members. We requested at least 4 hull members per class to stabilize the hull formation. We note that specifically, for hulls formed over fewer than 3 points, the shapes would degenerate to lines or points, rendering the area-based metrics meaningless. We also requested 4 FV spaces to have at least those many regression points, noting that for fewer points the regression easily degenerates. Consider specifically, that for just two regression points we can always perfectly fit a linear or logarithm function. Applying these restrictions, of the 90 classes defined in the PSB-T benchmark, we retained 44 benchmark classes.

Figure 3.25 reports the R^2 results of the log regression analysis for each of the qualified benchmark classes along the remaining 11 FV spaces. Charts (a), (b), and (c) report the regression results for the overlap, the area, and the silhouette hull metrics along the different classes, respectively. The results show a rather mixed pattern. For some classes and regressors, there is very tight correlation of up to 80% of R^2 or more, while for other classes and regressors, there is little or no predictive power of the regressors. There is no regressor which delivers best predictive power for all of the classes. On average over all benchmark classes, the regressors perform quite similar: the overlap, area, and silhouette metrics deliver 24%, 26%, and 25% of R^2, respectively.

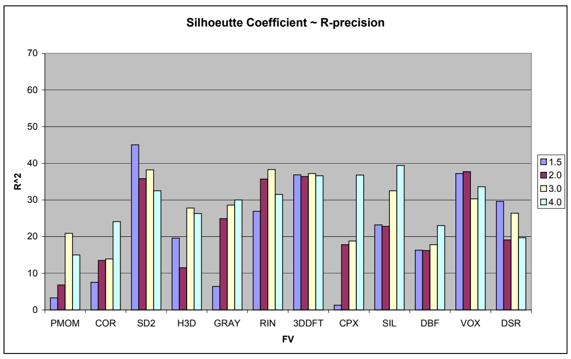
We also considered combined regressors. Charts (d), (e), and (f) report the R^2 results obtained when forming two combinations of the individual metrics using multiplication, like done above for the intra-FV analysis. We firstly observe that the regression results stabilize along the benchmark classes, and that both combinations yield comparable regression power given the benchmark class. Secondly, the combined regressors possess more predictive power than each of the three metrics on their own. On average over all classes, combining the area and silhouette (area, silhouette, and overlap) metrics yields 29%(33%) of R^2.



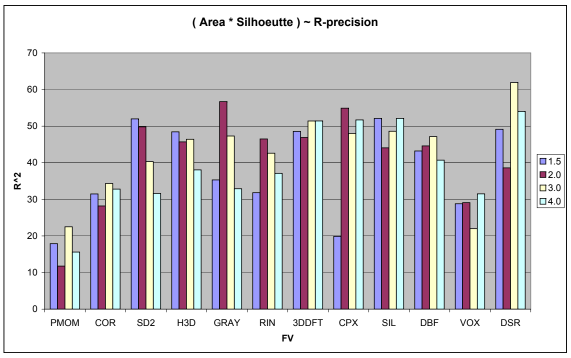
(a) overlap regressor



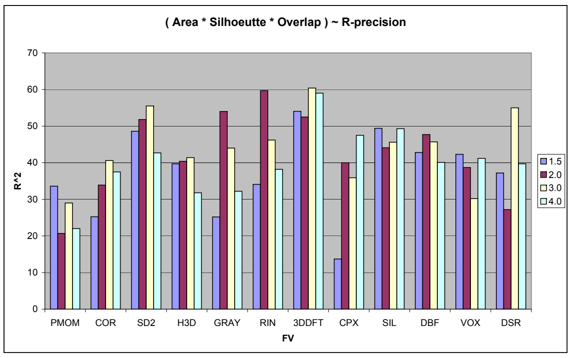
(b) relative area regressor



(c) silhouette coefficient regressor

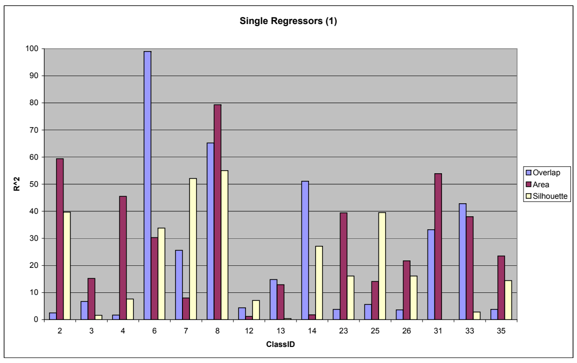


(d) combined regressor (area*sil)

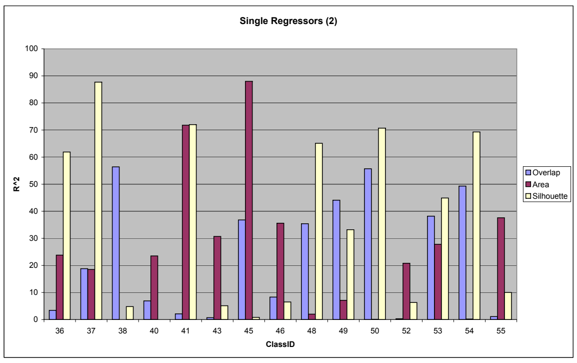


(e) combined regressor (area*sil*Overlap)

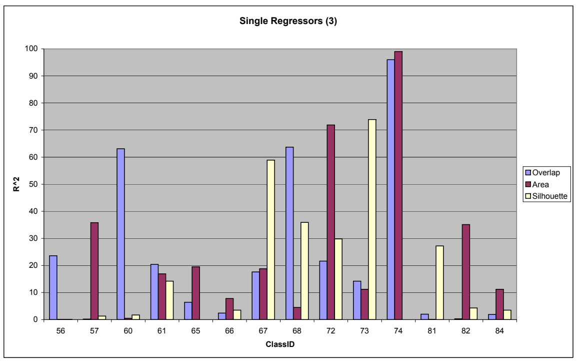
Figure 3.24: R^2 results of the log regression for the different FV spaces (intra-FV analysis), for single (a,b,c) and combined (d,e) regressors. Note that the FV spaces are sorted by increasing discrimination benchmark scores.



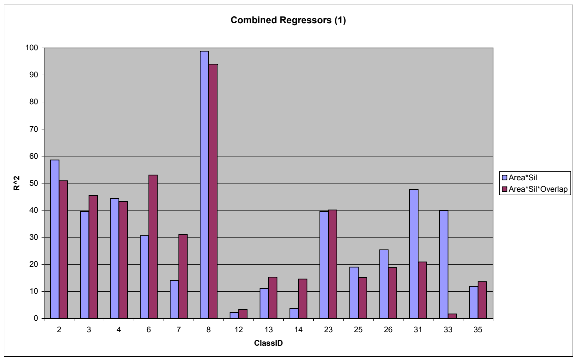
(a) Single regressors (1)



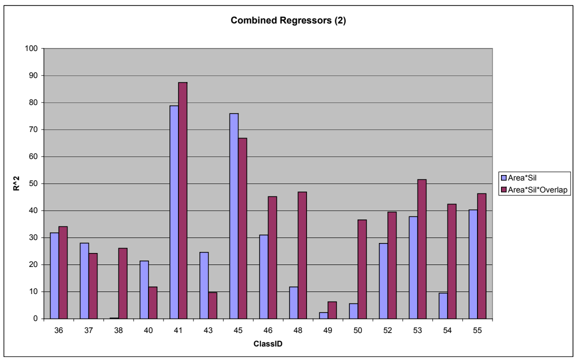
(b) Single regressors (2)



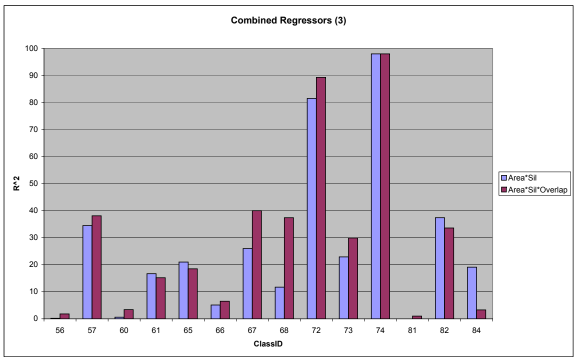
(c) Single regressors (3)



(d) Combined regressors (1)



(e) Combined regressors (2)



(f) Combined regressors (3)

Figure 3.25: R^2 results of the log regression for the individual benchmark classes (inter-FV analysis), for single (a,b,c) and combined (d,e,f) regressors.

Summary

To summarize this evaluation, we obtained good to quite strong correlation between the defined compactness and separation metrics of convex hulls in projected space, and discrimination benchmark scores in original space. For appropriate parameter settings, R^2 results of 50% and above are observed. Of course there is some sensitivity involved in such experiments. Varying regressor model, benchmark classes, and outlier removal levels, the predictive power varied among the classes and FV spaces. The strongest dependencies resulted when employing a combined regressor model with medium outlier removal setting. We conclude that there exists a substantial correlation between visual compactness and numeric benchmark scores, at least for the considered PSB-T data and the PCA projection as used in this evaluation. Considering that projection of high-dimensional FV space to 2D suppresses information, and that the convex hull metaphor was motivated visually rather than theoretically, this in an interesting result. It underscores the practicability of convex hull based projections for visual discrimination analysis. We finally note that by designing convex hull discrimination metrics, we are effectively developing visually motivated, at the same time numerically computable benchmarks. We believe such “benchmarking design” is an additional interesting use case of the convex hull visualization.

3.3.6 Conclusions

In this Section, we have motivated, practically applied, and statistically supported an intuitive, simple, yet effective approach for supporting important projection-based visual analysis tasks. The convex hull metaphor visually aggregates element classes in projected space, allowing to assess distribution metrics such as compactness, shape, separation and overlap among data sets consisting of classified objects. The metaphor can be combined with any 2D projection technique and supports visual analysis tasks such as discrimination analysis, visual benchmarking, metric space engineering, and database exploration. We believe projection-based visual analysis is a power tool for handling increasing volumes of complex data becoming available, which often are represented in multiple metric spaces.

Future work involves exploring further projection-based data analysis use cases, and experimenting with additional projection algorithms. We plan to design more complex metaphors considering free-form shapes and/or the density of points in projected space. A starting point for the latter idea could be to adapt the approach presented in [91]. Comparing such advanced metaphors to the plain convex hull should be interesting.

4 Space-filling visual object space analysis

Contents

In Chapter 3 of this thesis, we have discussed several approaches for visual analysis of *feature space* or *metric space*-based properties in multimedia databases aiming at summarizing database content (cf. Section 3.1) as well as estimating discrimination power in feature or metric space (cf. Sections 3.2 and 3.3). While the presented techniques are helpful for analyzing feature and metric-space properties, they rely on a high degree of abstraction of the database content: They provide aggregated or projected representations of the data and rely on FV or metric space representations of the multimedia content. Often, visualization solutions less abstract are required, mostly of course when the content has to be studied at the *object*-level which is the case e.g., for database browsing, for inspection of retrieval or clustering results, and for presentation of selected database content.

A requirement for object level visualization often encountered is a high degree of regularity when visualizing collections of multimedia objects. Consider e.g., the visualization of collections of 3D models (displayed by thumbnail images) or time series (displayed by bar- or line charts). Clearly, to support visual comparability between the objects, it is desirable to provide each object with the same amount and shape of display space when laying out the elements. In this Chapter, motivated by the popular TreeMap [82] family of layout algorithms, we consider layout algorithms providing regularity properties not present in the original TreeMap algorithm. Our algorithms try to be space-efficient, reflect hierarchic and ordering relationships present among the data elements, and target good regularity with respect to size, aspect ratio, and position of the resulting display partitions. The methods are discussed in the context of comparative *time series* visualization, but are applicable on a variety of multimedia data types such as images, and 3D models, among others. In Section 4.1, we survey the main TreeMap algorithms and variants. In Section 4.2, we introduce the *ID-Map* layout algorithm based on the idea of not doing continuous splitting of display space, but rather mapping data subsets into predefined partitions of display space using so-called splitting masks. Then, in Section 4.3, we revisit the *Quantum* TreeMap by Shneiderman et al, and compare it against variants of the *Grid* TreeMap algorithm we introduce in that Section. The algorithms are discussed in terms of application examples on real data, as well as evaluation of diverse error metrics in batch experiments.

Parts of this Chapter appeared in [26, 77].

4.1 A survey of TreeMap-based layout algorithms

4.1.1 Visualization of hierarchies with TreeMaps

The TreeMap family of layout algorithms is an Information Visualization success story. Proposed originally by Shneiderman in the early 90s [82], its suitability for the visualization of large, hierarchically structured data sets has been quickly recognized. The technique has been successfully applied in many different problem domains such as file system management, stock market monitoring, newsgroups monitoring, electorial results presentation, and biological data, among others [5]. Also, several shortcomings have been identified in the original algorithm, leading to a number of improved algorithms addressing certain layout goals. TreeMaps continue to be a highly interesting topic in the InfoVis community: At the 2005 IEEE Symposium on Information Visualization, five accepted papers focussed on TreeMaps, and experts in the field believe that the potential for improvement is not yet exhausted [90].

TreeMaps are about visualization of large, *hierarchically* structured data sets, i.e., data where all the data elements can be regarded as leaves in a tree data structure, with the tree encoding hierarchical relationships. There exist well-known techniques to visualize hierarchic data such as indented node lists, node-link diagrams, or so-called Venn diagrams (cf. Figure 4.1 (a) to (c)). While such techniques are rather familiar and allow the user to form a mental model of the hierarchic relationships present in the data, these techniques do not scale well with data set size, as a large fraction of the display space consists of background [82]. Desirable are techniques with higher space efficiency, allowing for better utilization of limited display space, at the same time supporting important analysis tasks such as reading of hierarchic relationships and comparing of data elements. The original TreeMap algorithm corresponds to a rectangular Venn diagram (cf. Figure 4.1 (d)), where all data subsets on a given hierarchical level are allocated rectangular display space in sequence to each other. By alternating the orientation of the data subset sequence layouts from one hierarchical level to the other between horizontal and vertical layout, and also minimizing the space allocated to the boundaries drawn around the data partitions, the TreeMap layout is obtained (cf. Figure 4.1 (e)).

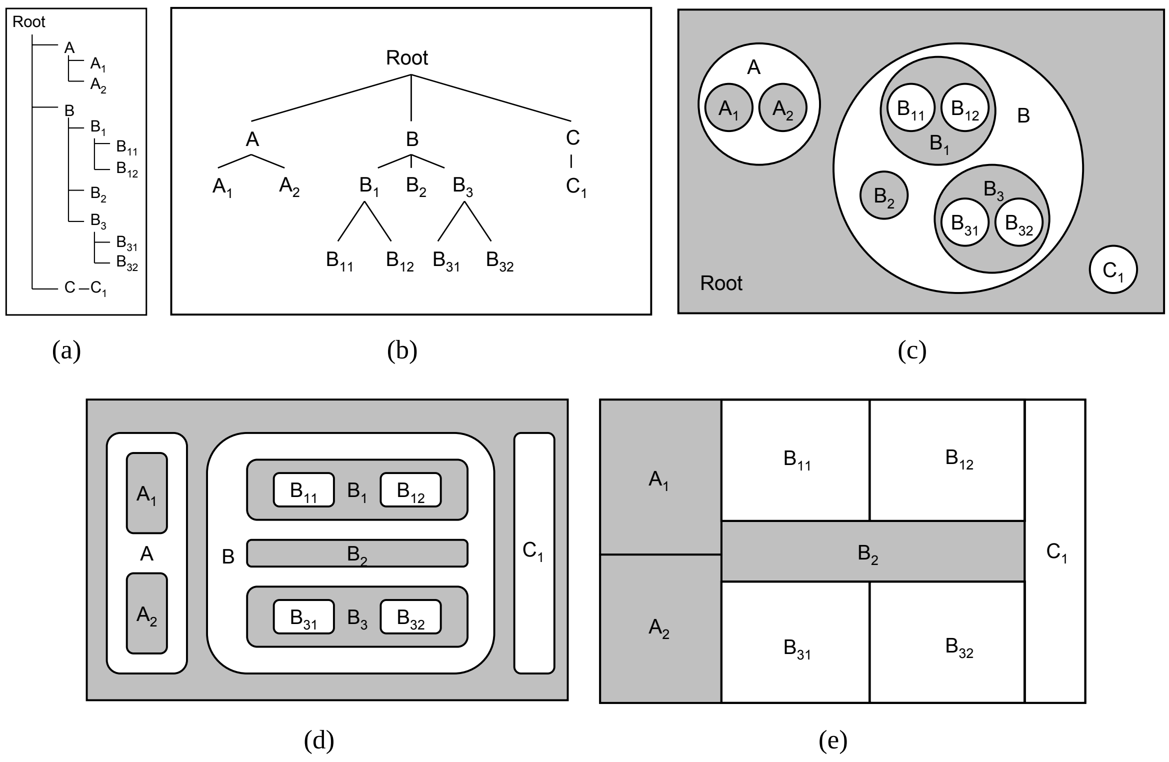


Figure 4.1: Visualization of hierarchies using familiar indented lists (a), node-link diagrams (b), and Venn diagrams (c). The classical TreeMap displays (e) can be regarded as a rectangular Venn diagram (d) with minimal border indention. Illustrations adapted from [83].

In the standard implementation [82], the algorithm recursively descends a given data hierarchy in top-down direction, allocating data subsets to display partitions by placing horizontal or vertical split lines in proportion to weights obtained by aggregating numeric properties underlying the data partitions. Such weights may include the number of elements, the sum of some quantity given by the data elements, or other explicitly or implicitly given numeric features of the data. A construction example is given in Section 4.3.1. The resulting display is space-filling and visualizes the data elements as well as the hierarchical relationships via rectangular containment.

The standard TreeMap algorithm has been recognized for its effectiveness in communicating element distributions in large data sets. Van Wijk et al stated in [11] that "*TreeMaps are efficient and compact displays, which are particularly effective to show the size of the final elements in the structure.*" On the other hand, the effectiveness (speed and accuracy) of perception of hierarchic relationships might not be as good as in traditional approaches due to the fact that hierarchy is visually supported with just a minimal amount of display resources, e.g., split lines of just a few pixels width. The standard TreeMap algorithm provides the straight split line property (cf. also Section 4.3.2) which in theory allows the user to determine the hierarchic structure of the data set by visually parsing the display. But parsing the display by tracking the split line structure top-down requires some effort, at least for complex hierarchies.

Several other drawbacks of the TreeMap approach have been identified. The original algorithm lays out the data in the order as it is encoded in the tree structure, and places split lines in strict proportion to sums of underlying weights. Thereby, the shape (aspect ratio) of the resulting display partitions is not considered, which in practice results in the occurrence of arbitrary aspect ratios. Specifically, long and thin (skinny) rectangles make it difficult to find, to compare, or to select respective data rectangles, and also, such rectangles are difficult to label appropriately [9, 94]. Also, by visualizing the data tree without reorganization, the resulting displays show discontinuous behavior if the underlying data weights change. This means that data elements will change their position within the display based on updates to the weights, which make the standard TreeMap inappropriate for visualization of dynamically updated content [9]. Several improvements have been made to the original TreeMap algorithm addressing such drawbacks, which will be discussed in the next Sections.

4.1.2 Desirable properties of TreeMap algorithms

When comparing different TreeMap algorithm variants, we can see that they try to satisfy certain (usually, contradictory) layout objectives which have been identified as desirable. By design of the respective TreeMap variants, the objectives are either guaranteed, optimized, or ignored. We next sketch the main layout objectives traditionally considered by TreeMap algorithms.

Space-Efficiency. Space efficiency (the utilization of display space with data elements) is the main objective in TreeMap layouts. Most TreeMap algorithms optimize this criteria by minimizing the amount of display space allocated to visualization of hierarchic relationships (e.g., rectangle indention and width of split lines).

Overlap. Besides space efficiency, it is commonly accepted in TreeMap algorithms to produce displays where display partitions do not overlap each other.

Hierarchy. The display should encode the hierarchic structure of the data set which is visualized. It should be possible for the user to easily identify the hierarchic relationships present among groups of elements throughout the data set. The hierarchy should be effectively (quickly and correctly) perceivable by the user. In TreeMaps, hierarchy is encoded by nesting data partitions inside non-overlapping, rectangular (or differently shaped) display partitions.

Proportionality of display partitions. The linear proportionality between display partition area and a selected quantitative property of the data elements contained therein, such as volume, amount, sum or the like, is considered important by many TreeMap designers. Proportionality provides that individual data elements as well as groups of elements (given the non-overlap property) can be quantitatively compared with each other by considering the area of the respective display partitions.

Ordering of elements. If the data set also contains an ordering among the data (leaf level or the inner levels), then this ordering should also be reflected by the position of the display partition in the map. The standard TreeMap preserves a given order, as it lays out the elements on a given hierarchy level sequentially in the given orientation, that is, either in horizontal or vertical stripes of display space. Other algorithms which perform resorting of elements to optimize the display typically violate the ordering goal.

Aspect ratios. Several algorithms consider the aspect ratios of the resulting rectangular display partitions. The reason is to avoid or minimize the occurrence of arbitrary rectangle shapes, e.g., long and skinny rectangles, which would be difficult to select, compare or label, or otherwise inappropriate to accommodate specialized content. Aspect ratios can be considered for rectangles just on the leaf level, or both on leaf and inner tree levels.

Computational complexity. A property typically desired in TreeMap layout generation is the efficiency of computation. TreeMap algorithms are usually designed for interactive applications, and consequentially implement computationally light optimization (if any). Typically, heuristics and greedy optimizers are used. To the best of our knowledge, there are only few exceptions of TreeMap techniques which employ computationally heavy optimization. Brute-force exhaustive searching in the layout space was used in [100]. The specific scenario there considered the layout of stock market data, which due to the limited size of the data set made it possible to do exhaustive search. Another exception is one technique proposed in [40] which employs genetic algorithm-based optimization for layout of geo-related data. This algorithm was designed for offline layout generation.

4.1.3 Existing TreeMap variants

Two main lines of improvement to the original TreeMap algorithm have been pursued in the literature. One line of work considers improving the shapes of the display partitions in order to better support comparability and interaction with leaf elements. The other line considers supporting the easy perception of hierarchy structure by improved rendering methods, e.g., using shading [92], or exploring non-rectangular display tessellation schemes such as circles

[74] or Voronoi cells [7]. We here cover algorithms of the first line of research and operating on rectangles, as our work also focuses on optimizing rectangle shapes for supporting the display of regularity-requiring content inside the rectangles (cf. Sections 4.2 and 4.3).

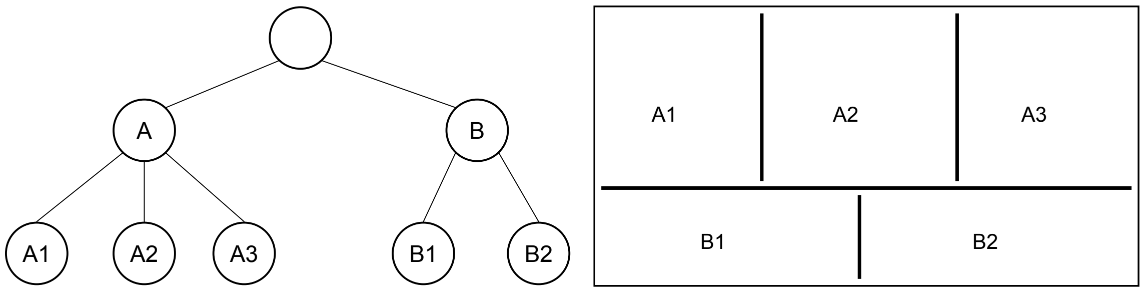
We classify the main body of TreeMap algorithms based on how the splitting decisions are found. We consider two dimensions to this end.

- The first dimension refers to how the split axis (either horizontal or vertical) is found for carrying out the split action. To this end, the splitting algorithm can either simply alternate between the split axes, which is called *slicing and dicing*. The standard slice and dice algorithm does not consider the shape of the resulting rectangles at all. *Optimizing* algorithms on the other hand base their selection of the split axis on characteristics of the data to lay out, aiming at optimizing the resulting aspect ratios towards meeting certain predefined target ratios. They employ certain layout heuristics expected to optimize the (aggregated) deviation of display rectangle aspect ratios from some preset target ratios. Optimizing algorithms are usually space-filling but cannot give guarantees for the aspect ratios of the resulting displays.

- The second dimension for our classification is whether on a given split axis, the split point is found in strict linear proportion to the weight of the underlying data set to be laid out, or whether it is constrained to assume only a limited number of different positions on the split axes. The first policy we like to call *continuous* splitting, as any of the infinitesimal many possible split points on a given split axis may be chosen. The other option we like to call *discretized* splitting. Discretization is usually done with regular grids (or use of splitting masks, cf. Section 4.2). The goal is to regularize the TreeMap display. Discretizing algorithms by design do better at achieving aspect ratio goals than do continuous algorithms, at the expense of the space-filling or proportionality properties (cf. Sections 4.2 and 4.3).

These two dimensions result in four possible classification possibilities, of which the three most important combinations are used for our classification. We next briefly recall specific algorithms from each of these three classes.

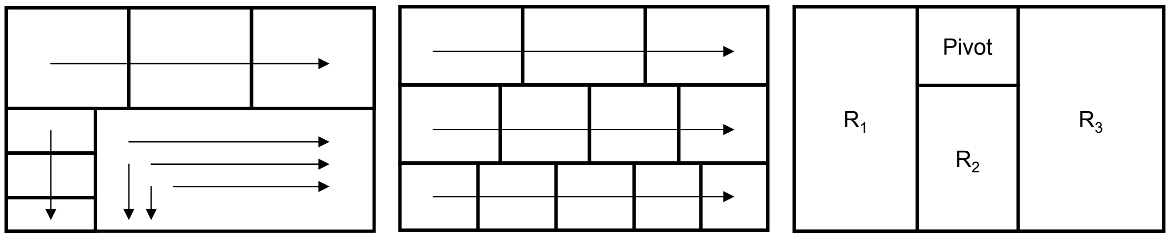
Continuous slice-and dice algorithms. The slice-and-dice approach continuously and recursively allocates a given (sub)tree of data elements to a given display space by either horizontally or vertically slicing the display. The number of slices corresponds to the number of elements to be laid out, where the elements are either leaf-level atomic data elements, or disjoint sets of elements (data subtrees). The split points are determined in linear proportion to the underlying weights associated with the data elements (or sums or other aggregates of weights in case a set of subtrees). Based on the data distribution to be laid out, arbitrary rectangle shapes may emerge. The original TreeMap algorithm [82] is a prototypical slice-and-dice algorithm. Figure 4.2 illustrates the mechanism of the original continuous slice-and-dice TreeMap layout on a simple hierarchical input data structure. A comprehensive list of TreeMap applications can be found in [5].



(a) Hierarchically structured data (b) TreeMap layout

Figure 4.2: The standard continuous TreeMap algorithm alternatingly splits display rectangles along horizontal and vertical lines while recursively traversing a hierarchically structured data set in top–down direction. In this illustration, a data set of 5 elements organized in 2 groups (a) is laid out by using the number of leaf elements as the weights for split point determination (b).

Continuous optimization algorithms. These algorithms perform rectangular tessellations where the resulting rectangle areas are in linear proportion to sums of underlying weights of the data elements. Opposed to the slice-and-dice algorithms, the data partitions on a given level are not necessarily assigned to a consecutive linear sequence of adjacent slices of display space, but rather, a number of different layout alternatives are evaluated, and the best one according to some quality criterion is selected for layout. Several techniques fall into this category.



(a) Squarified layout (b) Stripe layout (c) Ordered layout

Figure 4.3: Several layout principles proposed in the literature for squarifying the TreeMap display.

The *Squarified TreeMap* by van Wijk et al [11] targets square-shaped rectangles. When allocating a number of elements in a display partition, the algorithm lays out a number of elements, not necessarily all of the elements in the given set, inside one strip of display space such that a cumulated aspect ratio error measure is optimized. Then, the algorithm continues by laying out another partition of elements inside another strip of display space which may be oriented either parallel or perpendicular to the previous strip, and so on until all of the elements are allocated (cf. Figure 4.3 (a)). The algorithm rearranges the order of the elements such that they are laid out by decreasing element weight. The algorithm achieves good results in practice, but the (size-based) ordering of elements may be discontinuous if the algorithm breaks up a given sequence of elements into many small element subsequences allocated in differently oriented strips each.

Shneiderman et al in [9] gave two TreeMap variants also aiming to produce squarified layouts. Their *Ordered* layout algorithm partitions a list of elements to be laid out into three groups of elements and one special single element, the pivot element. The pivot is assigned to a rectangle along the top (or right, depending on the display rectangle's overall aspect ratio) edge of the display rectangle (cf. Figure 4.3 (c)). The first group of elements contains all elements with index smaller than the pivot element and is assigned to rectangle R_1. The remaining elements are partitioned into groups 2 and 3 (assigned to rectangles R_2 and R_3), such that the aspect ratio of the pivot rectangle approaches 1.0 as closely as possible. Then, the algorithm recursively continues to lay out groups 1 to 3 inside their assigned rectangles. Predefined layouts exist for cases where the number of elements is smaller than 4. The display is completely determined by this procedure up to the selection of the pivot element. To solve this, the authors propose several decision rules such as choosing the pivot as the central element or the largest element according to weight. The second layout variant given in [9] is similar to van Wijk's Squarified TreeMap, but fixes the layout stripe orientation to be horizontal for each stripe (cf. Figure 4.3 (b)): Each horizontal stripe is filled until the average aspect ratios of the elements contained is as close to 1.0 as possible. Both layouts are shown to produce good squarified layouts in practice. The authors point out the high degree of order their layouts provide. Opposed to the Squarified TreeMap, which performs layout by descending weigh, their algorithms perform the layout in the order given by the data set. The ordered layout places partitions of consecutive elements inside the same display rectangles, and the stripe layout lays out consecutive elements inside horizontal stripes. Both layouts provide that elements which are close to each other in the given element ordering are also located close in the display.

The algorithms recalled up to here heuristically optimize aspect ratios to be square like, other approaches are possible. E.g., in [94] the authors propose a scheme (called Modifiable TreeMap) which first partitions a set of elements to be laid out into two subsets. Then, each subset is laid out by dicing the two slices of the initial display rectangle which holds each one of the two partitions. The algorithm evaluates many options for partitioning the initial set of elements into two groups until a satisfactory result with respect to an aggregated error function is found.

Discretizing algorithms. The previous algorithms optimize rectangle aspect ratios to approach some desired aspect ratio. But there are also situations where a fixed aspect ratio must be guaranteed for each element in the display. Then, quantization of split points is an option. The *Quantum TreeMap* [9] is doing so. The basic idea is to discretize the intermediate output of a continuous TreeMap such that the resulting display partition's dimensions are matching integer multiples of a predefined base size for width and height. Thereby, it is possible to lay out elements inside cells of the predefined dimension (width and height) which are also positioned on a global regular grid. Figure 4.4 (a) illustrates the Quantum TreeMap. In Section 4.3, the Quantum TreeMap will be subject to an analysis regarding its space efficiency properties, and an improved quantized layout method will be proposed based on that analysis.

In addition to quantizing continuous layouts it is also possible to first tessellate the given display space into some regular scheme and then map the data elements to the obtained base units. *Bubble Maps* [8] follow this idea by allocating the cells on a regular grid with data elements in a rectangular or bubble-like scheme, effectively scanning a grid like done in bucket filling algorithms from Computer Graphics (cf. Figure 4.4 (b)). Furthermore, in [101] *Jigsaw Maps* were introduced which map data elements to grid cells by using a space-filling curve, e.g., the Hilbert-Curve. These quantized algorithms guarantee the same aspect ratio for each data element to be laid out. On the other hand, they cannot guarantee space-filling results, as it is not always possible to find a grid of dimensionality such that the exact number of elements in a given data set is matched, and the target aspect ratio is satisfied. Also, for the Bubble and Jigsaw Maps, perception of hierarchy can be expected to be more difficult than in the Quantum and Continuous algorithms, as the data (sub) partitions do not form rectangles any more.

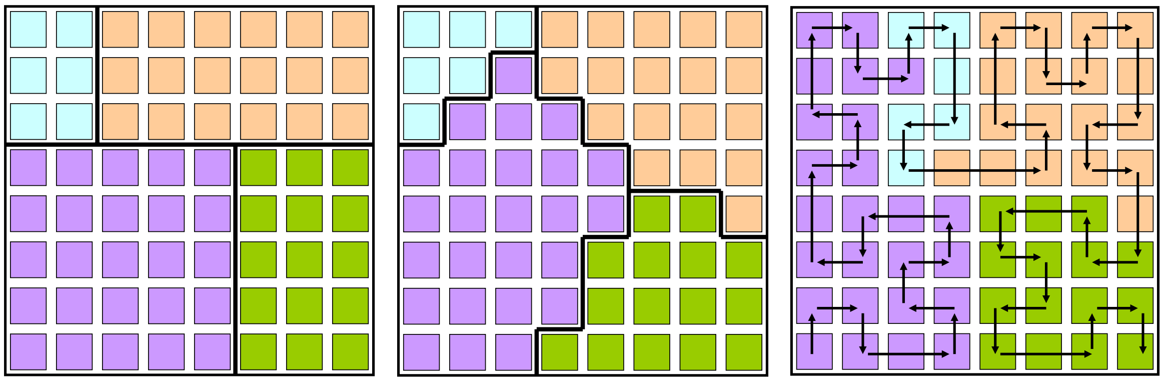

(a) Quantum TreeMap (b) Bubble Map (c) Jigsaw Map

Figure 4.4: Illustrations of several quantized TreeMap layout algorithms. Illustrations adapted from [8] (a,b) and [101] (c).

4.1.4 Analysis and proposed TreeMap algorithms

It is possible to compare the above discussed TreeMap variants along the display properties identified in Section 4.1.2. We first note that by design, all algorithms are *overlap-free*. The continuous algorithms are *space-filling* up to the space dedicated for split line drawing or indention of partitions which we consider an implementation issue beyond the main layout algorithm. The discretized algorithms are space-filling up to quantization, that is, the difference between the size of the base grid used, and the number of elements to be laid out in the given data set. We note that for the Quantum TreeMap relying on the continuous slice-and-dice algorithm, more subtle space-inefficiencies exist (cf. the discussion in Section 4.3.2). Regarding *hierarchy encoding*, the algorithms reflect hierarchy by nesting of data subsets within enclosing rectangular display space. An exception are the Bubble Maps, where the enclosing display space is not necessarily rectangular but can assume free-forms depending on the filling algorithm used. Also, the Jigsaw Map is an exception to this, as hierarchical relationships are given along the space-filling curves.

Proportionality between data element weights and respective area of the display partition is provided by design in the algorithms which perform continuous, proportional partitioning of display space. The quantized algorithms introduced so far map data elements to cells of constant size and aspect ratio. They thereby guarantee some atomic constant area for each element, but are unable to reflect varying individual weights, as all element rectangles have the same size. Therefore, differing individual weights for the data elements cannot be reflected. On the other hand, this predefined atomic dimensionality (*aspect ratio* property) for each data rectangle is guaranteed in the quantizing algorithms, a feature which is either ignored in the original slice-and-dice algorithm, or optimized in the optimizing line of TreeMap algorithms.

The most complex layout property distinction between the algorithms is probably the *ordering* feature. Ordering refers to how the display encodes an ordering between data elements as given by the data structure. The probably best reflection of ordering is achieved by the slice-and-dice algorithm which places all elements to be laid out linearly inside a given slice of display space. The Stripe TreeMap algorithm achieves slice-based linear ordering of elements only within each stripe, but exhibits discontinuity in element order at the stripe breaks. The Squarified TreeMap algorithm lays out the elements by decreasing weights, thereby destroying any distinct order which might be defined on the data set elements. Even if the original element order is in accordance with the sorted weights, the algorithm may provide only weak ordering, as in the worst case the stripe layout orientation switches on each subsequent data partition during the layout process (cf. Figure 4.3 (a)). The Ordered TreeMap achieves some kind of ordering by clustering together sequences of elements in enclosing elements (rectangles R_1 to R_3 in Figure 4.3 (c)). Other than that, it cannot be guaranteed that a linear ordering in the sense of placing the elements continuously inside display stripes is achieved. The quality of ordering then has to be experimentally evaluated using certain metrics of proximity of elements in the resulting displays as done in [9].

Considering the quantized algorithms, the Quantum TreeMap's ordering capability is defined by the underlying layout algorithm which is used to obtain the intermediate display which is then discretized. For the Bubble Maps, the ordering of elements depends on the concrete layout technique used for assigning the display cells, but can be expected to be moderate for the rectangular filling algorithm, and low for the bubble-like schemes. The Jigsaw Map in theory provides perfect (recursive) ordering as the space-filling curve used is a consistent linear scheme along which the elements are organized in the display. On the other hand, from the usability perspective it seems questionable whether space-filling curve schemes are intuitive to use for average users.

Table 4.1 summarizes the relationship between the TreeMap algorithms recalled in Section 4.1.3 and the main TreeMap visualization objectives identified in Section 4.1.2. Not included in that table are the less discriminating criteria *overlap freeness*, at it is given by design in these algorithms, and *computational complexity*, as all algorithms discussed work in linear time or employ light (greedy) optimization strategies. Also not included is the *space-efficiency* criterion, which raises some subtle problems for the discretizing algorithms only (to be discussed in depth in Section 4.3.

Table 4.1: Comparison of TreeMap variants recalled in Section 4.1.3 along important display properties identified in Section 4.1.2.

Algorithm class	**Algorithm name**	**Hierarchy**	**Proportionality**	**Ordering**	**Aspect Ratios**
Slice-and-dice	TreeMap [82]	rectangular nesting	linear	linear	ignored
Optimizing	Squarified [11]	rectangular nesting	linear	linear, discontinuous	optimized
	Stripe [9]	rectangular nesting	linear	linear, discontinuous	optimized
	Ordered [9]	rectangular nesting	linear	clustered	optimized
	Modifyable [94]	rectangular nesting	linear	linear, discontinuous	optimized
Discretized	Quantum [9]	rectangular nesting	atomic	as base algorithm	fixed
	Bubble [8]	freeform nesting	atomic	linear, discontinuous	fixed
	Jigsaw [101]	nesting along curve	atomic	recursive	fixed

In the following Sections we will introduce two TreeMap variants which each aim to improve the display to provide tessellations suited for display of data elements which are not only represented by the area of a rectangle, but which render specialized content inside them such as time series data. The ID-Map algorithm given in Section 4.2 introduces the notion of *splitting masks* into which partitions of data are allocated, and which provide certain regularity and ordering properties in the tessellation. These splitting masks can be considered as a quantization of the number of display partitions produced for a given layout level, and can be configured to provide different degrees of proportionality regarding the resulting display partition sizes. The *Grid TreeMap* introduced in Section 4.3 is a quantized algorithm which tries to avoid certain drawbacks given in the Quantum TreeMap algorithm by performing the TreeMap allocation directly on a grid of homogenously shaped display cells. Table 4.2 concludes this Section by summarizing properties of the proposed algorithms in context of the preceding discussion.

Table 4.2: Systematization of the ID-Map and Grid TreeMap algorithms to be introduced in Sections 4.2 and 4.3.

		Hierarchy	**Proportionality**	**Ordering**	**Aspect Ratios**
Optimizing	ID-Map [26]	rectangular nesting	approximate	recursive	optimized
Discretized	Grid TreeMap [77]	freeform nesting	atomic	linear, discontinuous	fixed

4.2 Importance-driven space-filling layouts for time series data

Time series are a multimedia data type of utmost importance in many application domains. Information Visualization to date has contributed with a variety of helpful techniques to understand and analyze time series data, where the focus has been mainly to support a limited number of time series, or to consider aggregated views of large collections of time series. For example, the *Polaris system* [38] allows the analyst to easily pivot and refine visual specifications of table-based graphical displays. Schumann employs a time wheel [1], where the basic idea is to present the time axis in the center of the display, and circularly arrange the variables around the time axis. Van Wijk [103] introduced a clustering-based visualization to condense multiple time series data into a calendar-based view. Shneidermans interactive pattern search [4] provides fast information retrieval on over-laid time series data. Sets of time series consisting of hundreds of thousands of observations may be visualized by resorting to pixel-based rendering paradigms [55].

In this Section, we address the problem of generating appropriate visualization layouts for simultaneously analyzing large sets of time series using the familiar bar or line charts drawing methods. Our goal is to allow an analyst to quickly perceive relative importance and hierarchy relations within sets of time series, while at the same time supporting good comparability of the data by highly regular layouts. We next introduce the notion of *importance-driven layout generation* for sets of time series (Section 4.2.1), and we formalize a set of constraints that we feel an effective layout for comparative analysis tasks on large time series data should provide (Section 4.2.2). We then develop a simple but effective heuristic algorithm that generates rectangular layouts based on the identified layout goals (Sections 4.2.4 to 4.2.3). We continue by applying the algorithm on several real-world data sets, demonstrating the practicability of the approach in Section 4.2.5. Section 4.2.6 presents an experimental analysis and a comparison against a popular space-filling and aspect-ratio aware TreeMap algorithm. Finally, in Section 4.2.7 we draw conclusions and outline possible future research directions.

4.2.1 Importance relationships on time series and layout requirements

When considering comparative analysis tasks on collections of time series, often there can be perceived a partial or total intrinsic *importance* (or *interestingness*) relation among the different time series. Such importance relationships should be reflected in the layout. For example, in a sales analysis application, the primary importance measure might be the total sum of sales numbers in each time series. In a network monitoring application, importance relationships may be derived from certain performance metrics taken from hosts on a network. Or, in a stock trading application, importance relationships may be derived from the variance in the stock price time series (a risk measure). An effective layout should support the perception of importance relations by using the two in our opinion most important display properties: *position* and *size*.

Regarding position, usually, the objects at the top of a display are perceived to be more important than those at the bottom, and objects on the left hand side are considered to be more important than those on the right hand side in a given display row (as subject to convention). Regarding size, larger objects are perceived to be more important than smaller objects. These natural ways to reflect importance relationships enable an analyst to quickly locate the most important objects as the data set grows large. In Sections 4.2.4 and 4.2.3, certain scalar importance measures derived from time series data serve as input for our layout algorithm, which in turn allocates size and position of display partitions into which to place the time series.

We note that our approach is inspired in part by the *degree of interestingness* (DOI) [35, 23] concept. The DOI concept models the interestingness of each data element in a data set as a function of its a-priori interestingness, and its distance to one or more current focus centers. The DOI concept can be used to generate interactive focus-and-context displays using distortion techniques. In terms of the DOI concept, here we only consider the a-priori interestingness component in the data set.

Time series are usually displayed using *bar* or *line* charts. Traditionally, multiple time series are accommodated by overlaying them in one common chart, or by using tabular, equal-sized layouts. Both approaches are problematic due to occlusion (overlaying) and an emerging need for scrolling interaction (tabulating) as the data set grows large. Also, the possibilities for encoding importance and hierarchical relationships are limited in these approaches. We therefore propose an overlap-free space-filling approach to time series layout to address both importance-coding and scalability. Overlapping layouts are also possible to address scalability, but we here do not investigate this line of design.

When laying out sets of bar or line charts in a space-filling display, it is not sufficient to allocate rendering space by assigning position and size according to importance, but also, regularity is a vital criterion for comparing time series. Regularity consists of the aspect ratio, which should be favorable for rendering a given number of time steps within each time series display partition. The aspect ratios of multiple time series should be homogeneous. Also, the alignment of the partitions should be as good as possible, and the number of unique horizontal scales should be low. Experiments we performed suggest that a low number of horizontal scales might be more important than a low number of unique vertical scales. We can support this observation by the fact that in bar and line charts, horizontal scale influences the perception of value sequence and duration of time intervals. Vertical scale influences perception of value magnitudes. While value perception can be easily supported using color maps, supporting perception of time sequence on many different horizontal scales is nontrivial, especially in space-filling layouts.

An additional requirement for importance-driven time series layout arises if there are also hierarchical relationships present among the set of time series. In a sales scenario, for example, the world might be divided into regions, and these regions themselves might be further subdivided into sub regions. For each sub region there may exist a time series for a given product by observing respective sales figures for consecutive points in time. Note that the embedding of hierarchical layout constraints may conflict with the importance-driven layout generation.

4.2.2 Formal problem definition

We here formalize certain requirements that an effective importance-driven time series layout should provide. Let $TS = \{TS_1, \ldots, TS_n\}$ denote a set of n time series objects, where a time series TS_i is a set of $|TS_i|$ pairs of real-valued observations with corresponding time stamps. $I_i = I_i(TS_i)$ is a real-valued function defined on time series implementing the application-specific, normalized importance measure, where $0 \leq I_i \leq 1 \;\; \forall i \in \{1, \ldots, n\} \wedge \sum_{i=1}^{n} I_i = 1$. The task of the layout algorithm is to partition an initial (root) rectangular display area R of width $R.w$ and height $R.h$ into a partition $P = P(R, TS)$ consisting of one sub-rectangle R_i for each time series TS_I. Let $|R_i| = R_i.w * R_i.h$ denote the area of R_i, and units are normalized such that $|R| = \sum_{i=1}^{n} R_i = 1$. Let $R_i.cx$ and $R_i.cy$ denote the x and y coordinates of the center of mass of R_i, with the display origin located in the south-west corner. Next we give constraints for display partitions over unstructured as well as hierarchically organized sets of time series.

Constraints for unstructured time series sets

$$\sum_{i=1}^{n} ||R_i| - I_i| \rightarrow min! \tag{4.1}$$

$$\uplus_{i=1}^{n} R_i = R \wedge \forall i, j \in \{1, \ldots, n\}, i \neq j : R_i \cap R_j = \oslash \tag{4.2}$$

$$\sum_{i=1}^{n} I_i * |\frac{R_i.w}{R.i.h} - c * |TS_i|| \rightarrow min! \tag{4.3}$$

$$\forall i, j \in [1, \ldots, n], i \neq j : \begin{cases} I_i > I_j \Rightarrow (R_i.cy > R_j.cy) \vee (R_i.cx < R_j.cx \wedge |R_i.cy - R_j.cy| < \varepsilon) \\ I_i < I_j \Rightarrow (R_i.cy < R_j.cy) \vee (R_i.cx > R_j.cx \wedge |R_i.cy - R_j.cy| < \varepsilon) \end{cases} \tag{4.4}$$

$$\sum_{i=0}^{n} \sum_{j=i+1}^{n} diff\_dim(i, j) \rightarrow min! \tag{4.5}$$

$$diff\_dim(i, j) = \begin{cases} 1 & R_i.w \neq R_j.w \vee R_i.h \neq R_j.h \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The *proportionality* constraint (Equation 4.1) defines incentive that the area of each time series rectangle should be proportional to the importance of the respective time series, $R_i \propto I_i$. The *space-filling and non-overlap* constraint (Equation 4.2) demands that the individual partition rectangles R_i never overlap, and that their union completely covers the available display space R. The *weighted aspect ratio error* constraint (Equation 4.3) demands minimization of the sum of importance-weighted aspect ratio errors, where the aspect ratio error is a function of the deviation of the actual aspect ratio of a rectangle R_i from a targeted aspect ratio. The targeted aspect ratio is modeled as a linear function of the respective time series length $|TS_i|$, and parameter c models the relation between time series length and targeted aspect ratio. The *ordering* constraint (Equation 4.4) demands that whenever there is an importance ordering possible between two time series, then this importance relationship is reflected in the display by placing the more important one above the less important one, or to the left of it, give they are placed on the same height in the display. The latter is tested for using a threshold parameter ε. Note that it is straightforward to adjust the ordering constraints such that, according to user convention, the rectangles would be positioned conversely, e.g., from right-to-left instead left-to-right. Finally, the *regularity* constraint (Equation 4.5) asks that the number of unique rectangle dimensions $(R_i.w, R_j.h)$ be kept minimal in the display. This can easily be measured by summing a function $diff\_dim$ (Equation 4.6) giving whether two rectangles have the same dimensions or not, over all pairs of rectangles. This is the core requirement we have in mind when designing the ID-Map algorithm.

Constraints for hierarchically structured time series sets

Let the hierarchical structure of TS be encoded by a tree TST, where each leaf distinctively contains one time series object $TS_i \in TS$, and inner tree nodes encode the hierarchical ordering. Let N_{TST} denote the set of all inner tree nodes of TST, and let $u \in N_{TST}$ denote an inner tree node. Let the set $TS(u)$ denote all time series rooted at inner tree node u, and let $MBR(TS(u))$ denote the minimal bounding box covering the rectangles of the time series from $TS(u)$. Finally, let $Agg(TS(u))$ bet a function aggregating the individual importance measures $I_i, i \in TS$ over the time series from $TS(u)$.

$$\sum_{u \in N_{TST}} (|MBR(TS(u))| - |\uplus_{i \in TS(u)} R_i|) \rightarrow min! \tag{4.7}$$

$$\forall u, v \in N_{TST}, u \neq v : \begin{cases} Agg(TS(u)) > Agg(TS(v)) \Rightarrow & (MBR(TS(u)).cy > MBR(TS(v)).cy) \vee \\ & (MBR(TS(u)).cx < MBR(TS(v)).cx \wedge \\ & |MBR(TS(u)).cy - MBR(TS(v)).cy| < \varepsilon) \\ Agg(TS(u)) < Agg(TS(v)) \Rightarrow & (MBR(TS(u)).cy < MBR(TS(v)).cy) \vee \\ & (MBR(TS(u)).cx > MBR(TS(v)).cx \wedge \\ & |MBR(TS(u)).cy - MBR(TS(v)).cy| < \varepsilon) \end{cases} \tag{4.8}$$

The *hierarchical rectangular containment* constraint (Equation 4.7) defines incentive to cluster the rectangles R_i in all sets of rectangles corresponding to the time series $TS(u)$ rooted at a tree node u as compact as possible. Compactness is measured by the size of the minimal bounding box MBR around the rectangles $R(u)$. The *hierarchical ordering* constraint (Equation 4.8) is defined similar to the non-hierarchical ordering constraint given in Equation 4.4. It extends the demand of left-to-right/top-to-bottom ordering from single time series rectangles to sets of hierarchically grouped time series rectangles. To this end, the ordering definition is rewritten based on minimum bounding rectangles grouping sets of time series rectangles $TS(u)$ and $TS(v)$ rooted at inner tree nodes u and v. The positional ordering of these minimum bounding boxes should reflect the relation between the aggregated importance measures of the underlying time series, according to a suitably defined aggregation function $Agg()$.

Solving the problem

The above constraints postulate certain properties that a rectangular display tessellation should provide for layout of sets of time series. The constraints are based on the considerations in Section 4.2.1. We recognize there exist conflicts between the objectives. E.g., it will not be possible to find a layout that simultaneously realizes the optimum for the constraints for proportionality (Equation 4.1) and regularity (Equation 4.5) for most data distributions, given that we simultaneously obey the space-filling and non-overlapping constraints (Equation 4.2). So, theoretically, we would only be able to find layouts optimal in the *Pareto* sense, and have to select one from these such that an appropriately combined, scalar error function is optimized.

Practically, finding optimal solutions involving multiple competing objectives of this type is a complex problem for which we do not expect to find efficient algorithms. In the following Section, we therefore propose a heuristic algorithm which is motivated by the above constraints, and which is producing visually satisfying results in interactive time as will be shown. The algorithm is based on partitioning the display space recurringly using homogeneous splitting patterns. The algorithm is designed towards supporting the regularity and aspect ratio error constraints. The space-filling/non-overlapping and hierarchical rectangular containment criteria the algorithm guarantees by design. The performance regarding the proportionality, ordering, and hierarchical ordering criteria are not explicitly addressed in the algorithm definition. It will be up to an experimental evaluation in Section 4.2.6 where we analyze the algorithm behavior with respect to these criteria. In this evaluation, we will come back to the above given constraint definitions, using them as error metrics for analyzing the algorithms performance.

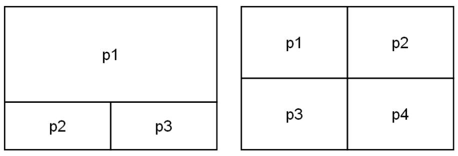
The role of the importance measure I_i (for short: *i*-measure) is to impose the importance relation on the set of time series. Usually, appropriate *i*-measures depend on the specific application context in which the visualization is to be deployed, and will have to be obtained from a domain expert. Suitable *i*-measures may be as simple as the *min* or *max* aggregation functions (e.g., when monitoring for network performance bottlenecks), or they may involve complex time series analysis algorithms (e.g., when searching for certain local patterns in trading data). In our system, we have implemented a set of basic *i*-measures, which already serve well for many applications:

- average, sum, min, max, count, deviation;
- exception count for some preset threshold;
- count of local extreme in the time series;
- the average difference between adjacent values.

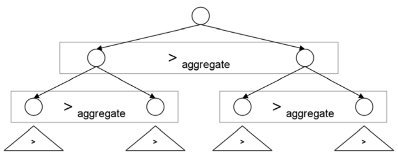
In addition, a number of optional time series preprocessing methods have been implemented, e.g., offset and amplitude normalization, smoothing, and missing value interpolation [56].

4.2.3 Splitting mask selection and splitting policies

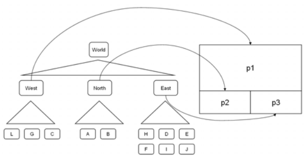
Our algorithm recursively maps ordered subsets of time series data into display partitions, which are constructed by a set of so-called splitting masks. We propose a set of *splitting*



(a) The *uneven* (left) and *even* (right) splitting masks.

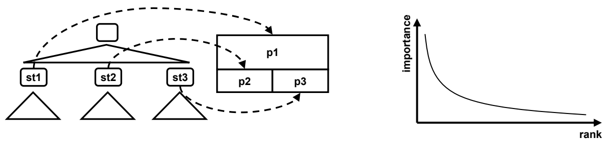


(b) Totally ordered time series tree.

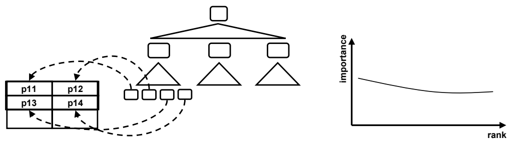


(c) An allocation step in the ID-Map algorithm.

Figure 4.5: ID-Map recursively allocates (c) partitions of the totally ordered time series tree (b) into partitions of pre-defined splitting masks (a).



(a) Uneven allocation of tree level 1.



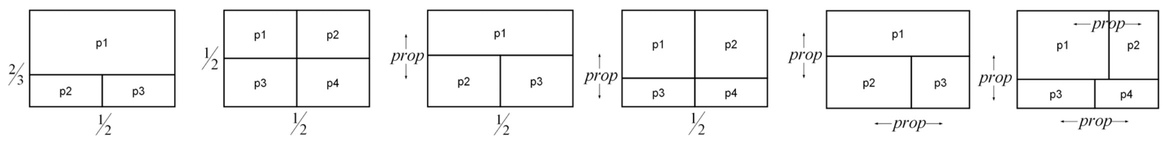
(b) Even allocation on first subtree on level 2.

Figure 4.6: Recursive allocation of time series subtrees using the mask chooser.

masks as a scheme for partitioning any given rectangle into a certain number of sub rectangles, reflecting importance-relations by size and position of the sub rectangles as given by the mask definition (mask structure). The role of the splitting masks is to provide regular layouts, and at the same time communicate importance and hierarchical relationships by size, position, and rectangular containment characteristics in the display. Using fixed display partitioning schemes avoids certain problems that traditional TreeMap algorithms exhibit by design, as they perform *continuous* splitting. Contrarily, using splitting masks, we perform splitting in a *restricted* manner.

For allocating a given set of time series, a *mask chooser* module first analyzes the distribution of *i*-measures present in the data currently to be laid out, and then selects from a set of predefined splitting masks the mask best accommodating the present *i*-measure distribution. We start by defining two masks suited for two salient types of distributions. The *uneven* mask contains three partitions and is appropriate when the distribution of *i*-measures is skewed. The other mask is the *even* mask and is selected if the distribution of *i*-measures is rather uniform. We use Pearson's Mode Skewness (*PMS*) [102] as the skewness scale. Figure 4.5 (a) illustrates the basic splitting masks, and Figure 4.6 illustrates two steps in the recursive layout using the mask chooser.

Considering mask split-point determination, we define three different policies, each one implementing a certain trade-off between the *size-proportionality* and *regularity* constraints. Policy *A* splits an input rectangle at fixed relative positions, irrespective of the *i*-measures underlying the data to be allocated. For the even mask, policy *A* splits both horizontally and vertically at $\frac{1}{2}$ edge length. For the uneven mask, it vertically splits at $\frac{2}{3}$, and horizontally at $\frac{1}{2}$ edge length. Policy *A* results in maximum regularity, but does not guarantee linear reflection of importance-relationships purely by size. In policy *B*, the rectangle is split vertically in linear proportion to sums of the underlying *i*-measures, but horizontally it is split at $\frac{1}{2}$ edge



(a) Split policy A (b) Split policy B (c) Split policy C

Figure 4.7: Splitting policies *A*, *B*, and *C* applied on the uneven (left) and even (right) splitting masks.

length. This results in less regularity, but improves size-proportionality. Finally, policy *C* performs all the splitting in linear proportion to the sums of underlying *i*-measures, guaranteeing linear size-proportionality at the expense of regularity. Figure 4.7 illustrates the three splitting approaches.

We note that for now, we fix the splitting policy based on user preference when generating the layout. We note that it would also be possible to perform the policy selection in a data-dependent way, but leave this for future work. Regarding the size proportionality and regularity tradeoff, we note that the importance relations are always encoded in the overall nesting structure of the display, and specialized techniques supporting nesting structure perception exist [88].

4.2.4 ID-Map algorithm

ID-Map algorithm for unstructured sets of time series

For generating an importance-driven layout for an unstructured set of time series, we first determine the *i*-measure for each time series object, and build a list of time series sorted decreasingly by *i*-measure. Evaluating Pearson's Mode Skewness of the *i*-measure distribution of the list, we select the appropriate splitting mask M_s from a set M of predefined masks. As M_s defines $n = |M_s|$ partitions, we also partition the sorted list of time series into n equal-sized ordered subsets of time series. We then assign each time series subset in order to the respective display partition as defined by M_s, and recursively proceed with all subsets and display partitions, until each time series has been allocated to one display rectangle each.

ID-Map algorithm for hierarchically structured sets of time series

A set of hierarchically organized time series can be held in a *rooted tree*: Inner tree nodes encode the hierarchy; each leaf node of the tree holds one time series. The tree can be totally ordered by *i*-measures. To this end, we first aggregate the *i*-measures from all leaf nodes bottom-up along the hierarchy, until each inner tree node is labeled with an aggregated *i*-measure. We then sort the children of all inner tree nodes by their respective *i*-measure labels, obtaining a totally ordered rooted time series tree. Figure 4.5 (b) illustrates. The algorithm for unstructured sets can then be applied by considering sorted lists of time series tree nodes, instead of sorted lists of time series, as input to the recursive layout. Generation of the layout is initialized by inputting the tree root to the algorithm, and it terminates once all branches

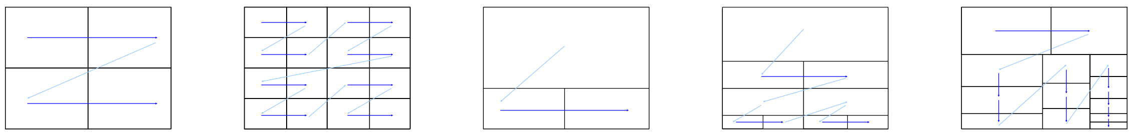

(a) (b) (c) (d) (e)

Figure 4.8: Recursive partitioning of display space using the even (a,b) and uneven (c,d) splitting masks. (e) illustrates a typical ordering result of the Squarified TreeMap.

of the tree have been processed, and all time series have been allocated. Figure Figure 4.5 (c) illustrates the allocation of inner tree nodes from a geo-related hierarchy to the partitions of an uneven splitting mask. The example assumes that the region West has a significantly higher aggregated i-measure than regions North and East. The ID-Map Algorithm for the hierarchically structured case is given in pseudo-code in Algorithm 2. As will be shown next, this scheme is able to produce regular layouts which favor importance-driven perception and fast visual comparison of many different time series simultaneously.

We note that due to the recursive mapping of sets of elements to partitions of the display space, the positioning of consecutively ordered elements follow typical recursive patterns. A user unfamiliar with recursive orderings might need some time to familiarize with such orderings. Figure 4.8 illustrates the ordering patterns in the even and the uneven splitting masks.

```
Procedure ID_Map(list of nodes L, display rectangle R):
/* terminal node reached: draw the time series */
if ($L$ contains exactly one leaf node) then
    drawTimeSeries($L[0]$.timeSeries, $R$)
    return
end if
/* single non-leaf node reached: recursively layout child nodes */
if ($L$ contains exactly one non-leaf node) then
    ID_map( $L[0]$.children, $R$ )
    return
end if
/* list of nodes: select splitting mask; layout chunks of nodes */
$M_s \leftarrow$ choose_mask($M$, $L$)
partition $L$ into $n$ equal-sized, ordered chunks of nodes $c_1, \dots, c_n$, where $n = |M_s|$;
for $chunk = 1$ to $n$ do
    ID_map($c_{chunk}, R_{chunk}(M_s, R)$)
end for
```
Algorithm 2: ID-Map algorithm

4.2.5 Application

Split policy and mask chooser configurations

We have experimented with different masks choosing schemes, such as manipulating the skewness threshold for the PMS selector, restricting the mask chooser to a fixed mask, as well as distinguishing between different choosing rules for inner and leaf nodes in the time series tree. Figure 4.9 shows the results for four different configurations of the algorithm on a dataset of 80 time series from the finance domain, organized in 9 classes (see next Section for details on the data set).

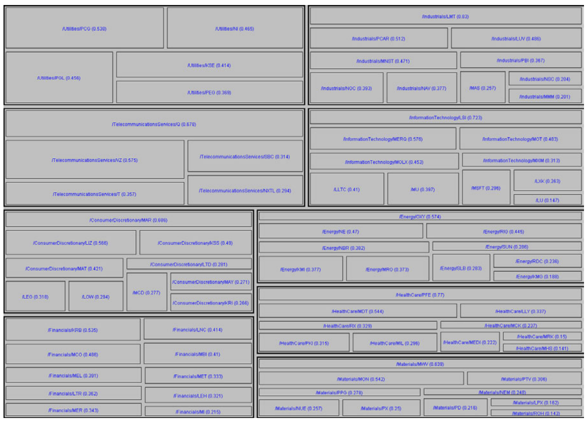
In Figure 4.9 (a), we let the mask chooser select between the even and the uneven splitting masks, based on a volatility measure, to layout both inner and leaf level nodes of the time series tree. The splitting policy is set proportional to sums of underlying *i*-measures. (policy *C*). The display communicates hierarchical relationships between the elements, the size of each rectangle is proportional to the volatility measure, and more volatile Sectors and stocks are positioned top-left in the display. For some time series, due to proportional splitting, we have aspect ratios which are rather square-like, and so, only suboptimal for comparing x and y scales among the different time series elements. In Figure 4.9 (b), we have set the mask chooser to use split policy *A*, which performs splitting always at fixed relative positions, ignoring (sums of) underlying *i*-measures in the layout generation. The result is a more regular display with a reduced number of unique x and y scales throughout the display (specifically, there occur only 3 unique x scales). Size proportionality is not linear anymore, but position still indicates importance relationships among the data, and comparability of x and y scales in the display is improved.

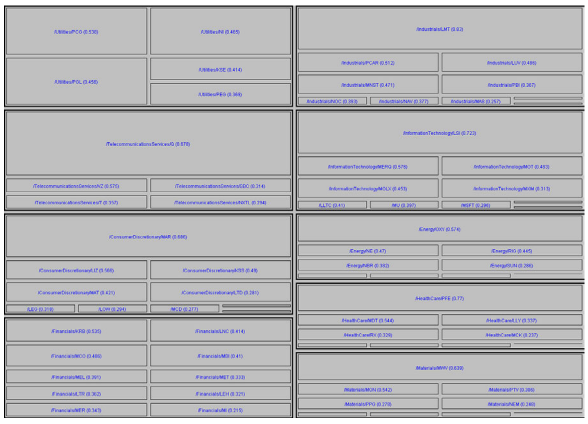
To even further improve regularity, we may restrict the mask chooser to always select a fixed splitting mask for laying out the data. In Figure 4.9 (c), we fix the layout of the leaf level nodes to uneven mask. In Figure 4.9 (d), we have fixed the layout to even on all groups of time series. Both variants share that they disregard distributions of *i*-measures when laying out the leaf level data, so we cannot distinguish anymore between uniform or skewed distributions of *i*-measures among the different sectors; only *ordinal* relationships are perceivable from the display. On the other hand, regularity is further increased.

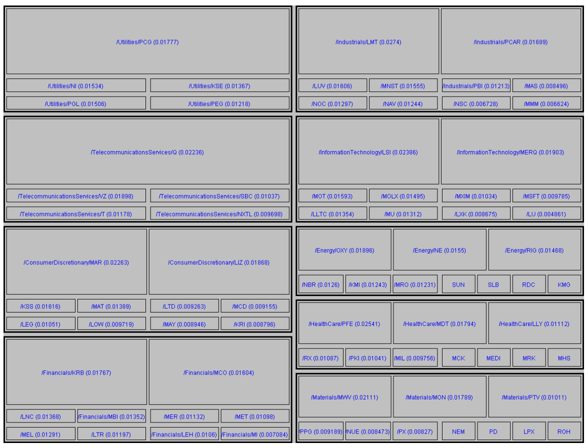
From these results we conclude that already the two basic splitting masks (even, uneven), along with appropriate settings for the mask chooser module, offer a wealth of possible layout results for visualizing sets of time series data. By fixing the above described layout parameters, the user can tune the layout to fit her needs and preferences.

Application on financial datasets

Finance is an application domain where naturally, large sets of hierarchically organized time series data occur. For example, the GICS standard is a multi-level hierarchical classification of the 500 stocks compound in the Standard&Poors 500 index. Investors, in order to make investment decisions, often need to overview, compare, and analyze stocks from specific sectors they are interested in. Our algorithm is suited to provide an effective overview over sets of categorized stocks, where the most important time series are readily perceivable and analyzable. Figure 4.10 shows 30 normalized daily opening prices from October and November


(a)


(b)


(c)

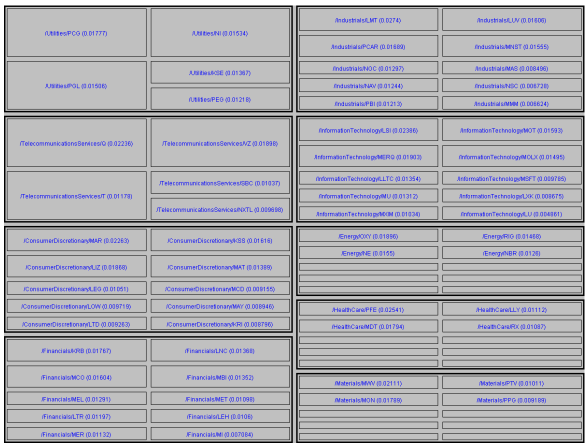

(d)

Figure 4.9: Several different mask chooser configurations along with resulting layouts.

2003 of 80 stocks from 9 different *S&P500* Industries (the data was obtained from [84]). For this example, assume we are interested in comparing the relative volatility of stocks. In order to identify stocks of interest to a risk-seeking investor, we first apply offset- and amplitude normalization [56] as preprocessing. We then select as the *i*-measure the average difference between all adjacent values of each time series, respectively. This *i*-measure rates the volatility of stock prices. For sector level aggregation, we average over the *i*-measures from all time series contained in each of the 9 sectors, respectively.

Using this *i*-measure to guide the layout, the ID-Map algorithm maps each two or three sectors to one partition of an even mask. From the generated layout we learn that the Utilities and Telecommunications Sectors (located top left in the display) are the most volatile in this example, as they are placed into partition $p1$ of an even layout. Sectors Energy, Health Care, and Materials (bottom-right) are the least volatile (placed in partition $p4$). The stocks from Utilities are more volatile than those from Telecommunications on average (they are placed on top of Telecommunications). On the other hand, the distribution of volatility among the stocks from Utilities is more uniform (ranging from 0.018 to 0.012) than those from Telecom (ranging from 0.022 to 0.010). This is readily perceivable, as the mask chooser lays out the Utilities stocks in an even mask, while it chooses an uneven mask for Telecommunication. Using the fixed splitting policy A in the layout algorithm, the overall display partitioning is highly regular. Note that only three different x-axis scales are present; this supports visual comparability of the actual time series data.

As another example, we have applied ID-Map to lay out a set of long bond price time series (cf. Figure Figure 4.11). The display shows daily prices of 170 bonds during roughly 13 years of trading, resulting in up to 3.200 values per bond. As *i*-measure we considered the relative increase in the bond prices over the respective time intervals, and normalized prices were plotted using a pixel-oriented technique similar to [55]. The analyst can not only compare the life spans of the bonds (the portions within the rectangles occupied with values; white is background), but also the total increase in price over all years (the size of the rectangles) and the growth path over the durations (the change in the color map).

4.2.6 Comparison with continuous aspect-ratio optimizing TreeMap algorithm

In this section, we compare the ID-Map layout algorithm with an aspect ratio optimizing variant of the well-known TreeMap layout algorithm. We discuss key features of both algorithms, and present results obtained from batch experiments performed on synthetic data.

Aspect ratio aware TreeMaps

TreeMap [82] is an excellent tool for displaying large volumes of low-dimensional, hierarchically organized data. By design, the original algorithm does not care about the regularity of the display it generates, as split points are placed in linear proportion to sums of underlying measures. While proportionality is guaranteed, there is no guarantee for regularity. Several variants optimize aspect ratios [9, 11, 94]. Here, we extend van Wijk's Stripe-filling method

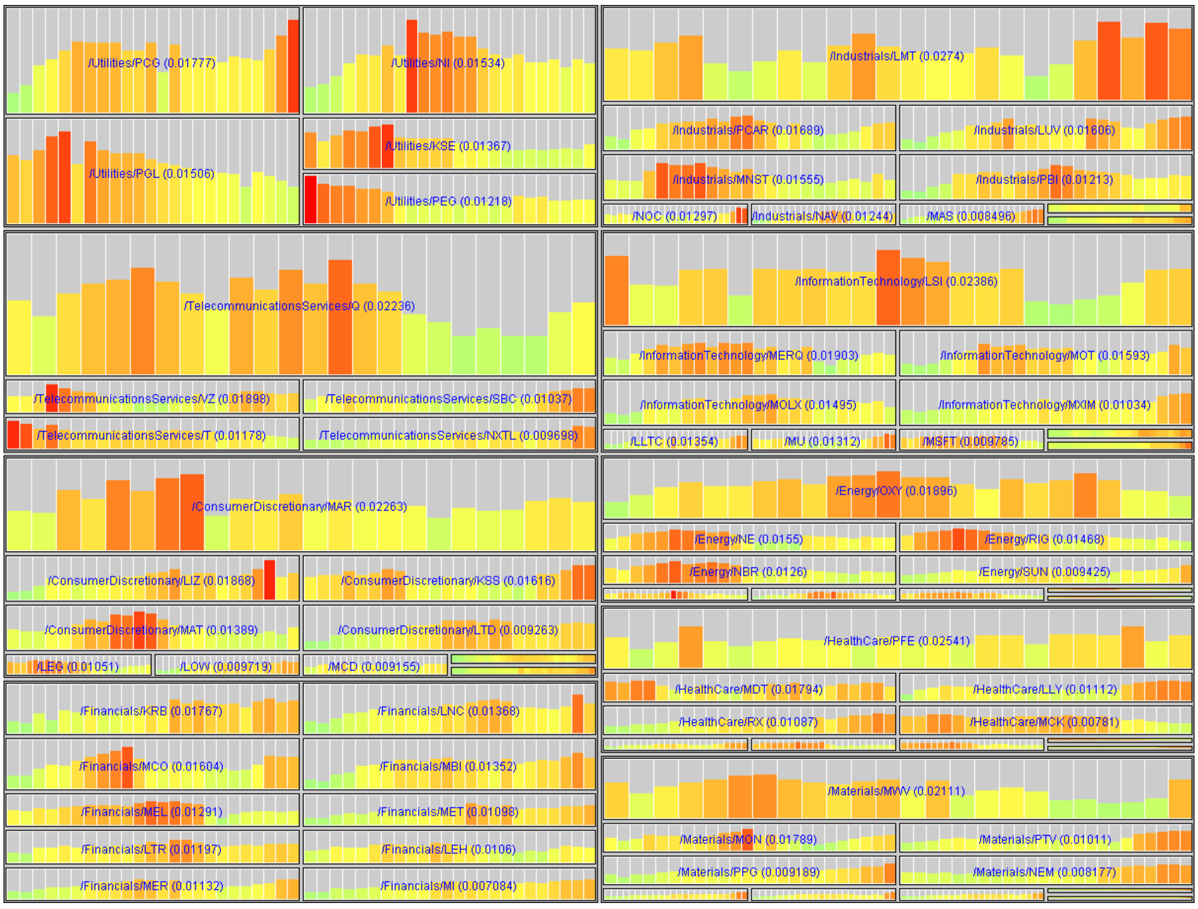


Figure 4.10: Hierarchical ID-Map (configured to split policy A), applied on a set of 80 time series from 9 different S&P500 Industries. The mask chooser uses even and uneven masks to distinguish skewed from uniform stock risk among and within Industries. The *i*-measure used is normalized volatility of stocks; color used in the bar charts redundantly indicates normalized stock open price from green (low) through yellow (medium) to red (high).

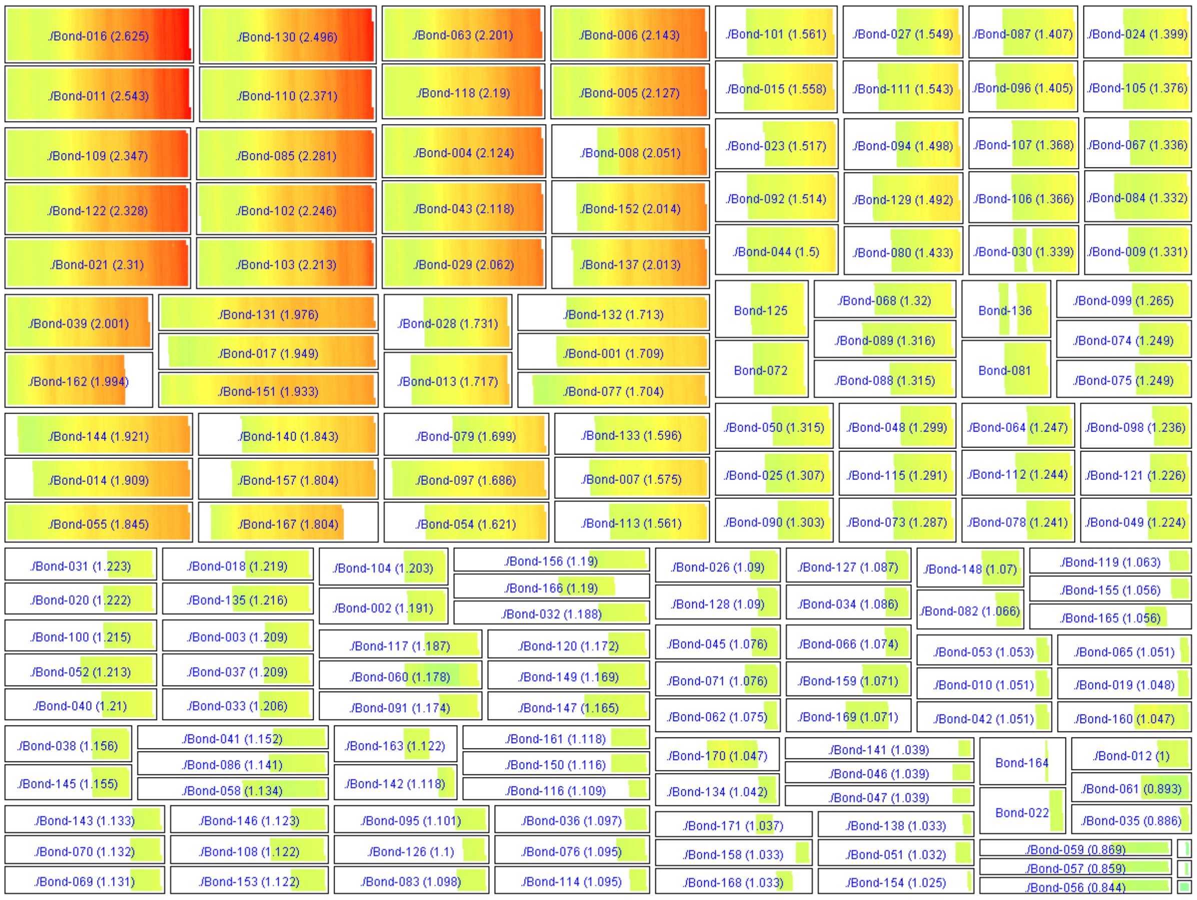
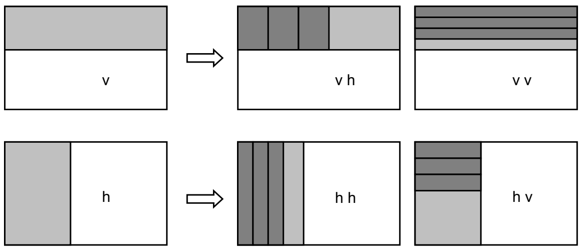


Figure 4.11: Non-hierarchical layout of daily prices of 170 bonds of up to 13 years of trading. Split policy C using the even split mask was fixed for all layout steps. The bond prices were normalized to 100%, starting with the first price record available for each bond. Green denotes 100%, red denotes the maximum (260% in this data set). The growth in bond price over the bonds' life span was selected as the i-measure. The bonds mainly experience smooth, continued growth over their respective life spans. Bonds existing for longer periods of time have grown larger than bonds which are comparably younger. The rectangle sizes are directly proportional to the relative bond growth rates.

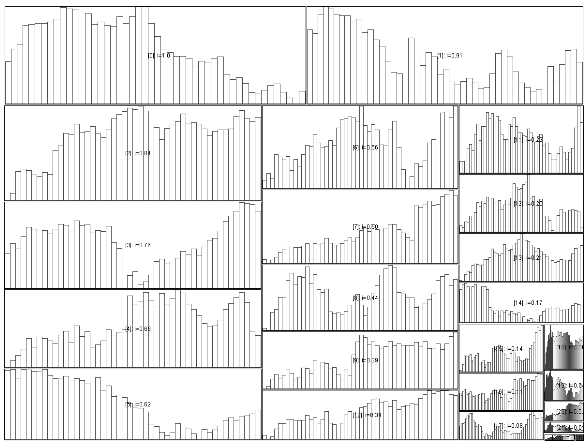
[11] (*Squarified TreeMap* algorithm) to support data-dependent target aspect ratios. The original approach aims to generate square rectangles, and the overall aspect ratio of the rectangle to be filled determines the orientation of the current layout stripe. For our purposes, we consider four possible layout orientations in each step (see Figure 4.12 (a)): We greedily fill each one in four possible orientation stripes with rectangles until the cumulated sum of weighted aspect error (see Constraint in Equation 4.3 in Section 4.2.2) has reached the minimum. We then make the stripe with the lowest overall cumulated error sum permanent, and recursively continue the layout within the remaining rectangle. For evaluation purposes, we applied the modified Stripe TreeMap algorithm to a data set consisting of 24 unstructured time series (48 values each), assuming a quadratic distribution of *i*-measures following $f(x) = x^2, x \in [0..1]$ for the set of time series elements. For the aspect ratio error function, we set parameter c to $\frac{1}{16}$, targeting an aspect ratio of $4:1$ for each time series of 48 values. Figure 4.12 (b) shows the result. While we have good horizontally-oriented aspect ratios around $4:1$, due to continuous splitting in strict proportionality, the number of different x- and y-axis scales is high. Also, the rectangles are not lined up at stripe borders. Furthermore, the ordering of elements may be discontinuous at stripe borders, whenever the algorithm performs a horizontal/vertical (or vice versa) change in stripe orientation. These facts result in medium overall regularity of the display.

ID-Map layout

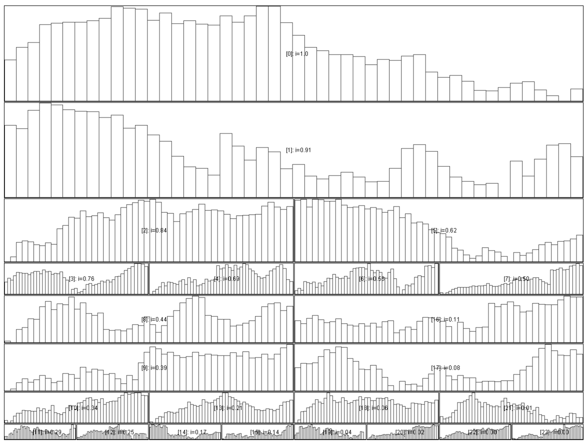
Figure 4.12 (c) shows the application of ID-Map using the uneven mask with splitting policy A (fixed split points) on the dataset. We notice the display's high regularity. Due to the recursive allocation of chunks of time series to display partitions using fixed splitting, rectangle sizes are not guaranteed to be in linear proportion to *i*-measures. We notice that even a reversal of size and *i*-measure relations may occur (note that *i*-measure relations are always perceivable by the overall splitting structure). To improve proportionality, it is possible to re-sort the rectangles by size in a top-down/left-right manner followed by re-assigning time series to rectangles (Figure 4.12 (d)). While we maintain the high regularity of the display, we obtain better proportionality and avoid the reversal case. Splitting policies B and C increasingly trade-off regularity for improved proportionality between *i*-measure and rectangle size. The regularity will be reduced, as either vertical (B), or both vertical and horizontal splitting (C) is performed in proportion to underlying i-measures. For most *i*-measure distributions, this will lead to increasing numbers of unique scales. To moderate the loss in regularity, we can quantize the split points by implementing a "snap to grid"-like splitting function. Figures 4.12 (e) and (f) show the quantized uneven layouts obtained from split policies B and C, respectively. Comparing Figures 4.12 (e) and (f), we note that policies B and C improve the *i*-measure to size proportionality at the expense of reduced regularity, as due to the higher number of different edge lengths occurring, the aspect ratio and regularity constraints (constraints in Equation 4.3 and 4.5 in Section 4.2.2) get more stressed.



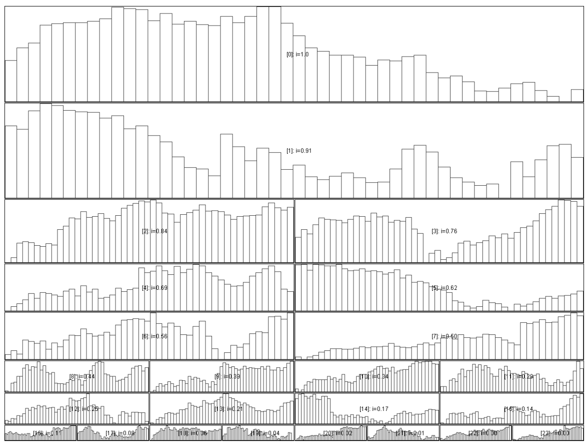
(a) Layout orientations in the adapted Squarified TreeMap algorithm.



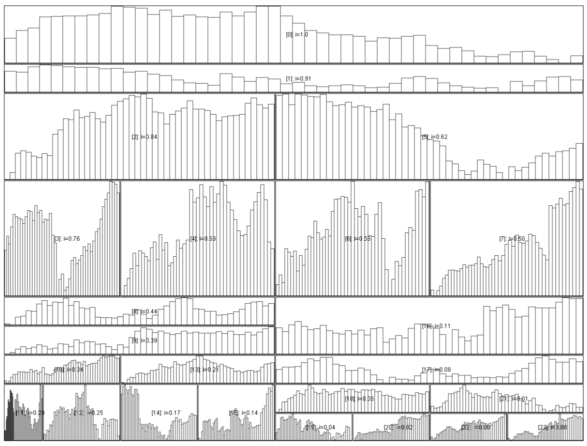
(b) Adapted Squarified TreeMap result.



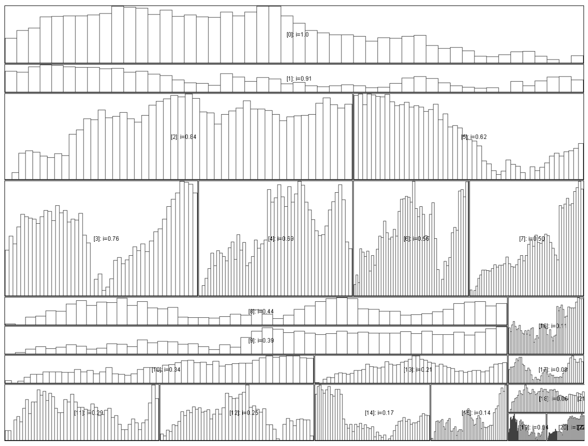
(c) ID-Map, uneven, split policy *A*.



(d) ID-Map, uneven, split policy *A* (resorted).



(e) ID-Map, uneven, quantized split policy *B*.



(f) ID-Map, uneven, quantized split policy *C*.

Figure 4.12: Adapting the Squarified TreeMap algorithm for time series data (a), and comparing the result (b) against ID-Map using different splitting policies of the uneven splitting mask (c – f) on a synthetic data set.

Experimental evaluation

We also conducted a series of experiments on synthetic data to obtain numeric performance results. We used the algorithms to layout unstructured sets of 5 up to 200 time series with 48 values each, within a 1200×900 pixel display. We assume *i*-measures are uniformly distributed in $[0,1]$ for each data set (we also tested square root and square like distributions, but obtained similar results). We restricted ID-Map to use the uneven splitting mask with non-quantized split point determination throughout the experiments. We also tested restricting to the even mask, and again obtained similar results. For the Squarified TreeMap algorithm, we set parameter $c = \frac{1}{16}$ targeting aspect rations of $4:1$ for each time series of length 48. For each generated layout, we evaluated a set of error functions based on the postulations given in Section 4.2.2. Figure 4.13 contains the obtained error numbers. For a sequence of corresponding display snapshots, cf. Figure A.12 in the Appendix.

Figure 4.13 (a) shows the *normalized regularity error* (constraint from Equation 4.5 from Section 4.2.2) as the ratio between unique combinations of rectangle width and height scales (as measured with a 2 pixel threshold), and the number of rectangles. The metric can be interpreted as the number of unique rectangle dimensions per data element in the layout. Lower numbers indicate higher regularity. In this metric, as expected the ID-Map splitting policies A and B perform better than the Squarified TreeMap algorithm, for all data element cardinalities. This is not true for splitting policy C, which becomes worse than the Squarified algorithm for roughly more than 60 rectangles. If we consider just the fraction of unique x scales irrespective of the behavior of y scales, not surprisingly, splitting policies A and B perform similarly, as they split the x axis always at fixed relative positions (see Figure 4.13 (b)). For growing numbers of elements, the Squarified algorithm approaches the ID-Map policies A and B. The good performance of the Squarified algorithm in this metric can be attributed to the fact that the algorithm often chooses *vertical* stripes in this setting (layout orientations *vv* and *hv* in Figure 4.12 (a)). This results in decreasing fractions of unique x scales per element. We can also consider the same metric for the y axis alone (see Figure 4.13 (c)). Here, split policy A outperforms all other algorithms, while polices B and C perform identical and slightly better than the Squarified algorithm.

A second performance metric considers the *normalized ordering error* based on constraint in Equation 4.4 in Section 4.2.2). Here, we measure the fraction of top-down and left-right ordering violations among all pairs of time series, using a 10 pixel threshold for parameter ε. Despite its recursive tessellation scheme, ID-Map with all splitting policies stabilizes at around 12% ordering violations and manages to consistently outperform the Squarified algorithm, which oscillates around 30% ordering violations. Figure 4.13 (d) gives the numbers.

In two other metrics, the Squarified algorithm performs quite well and better than ID-Map. Firstly, by its' continuous splitting policy, it guarantees size proportionality (see constraint in Equation 4.1 in Section 4.2.2). In Figure 4.13 (e), we have evaluated the *normalized proportionality error* and see that the ID-Map splitting policies B and A produce increasing proportionality error. Squarified TreeMap and splitting policy C produce no proportionality errors (up to rounding effects in the pixel display), as both perform continuous, proportional split point determination. Secondly, Squarified TreeMap produces low *weighted aspect ratio error* as measured by constraint in Equation 4.3 in Section 4.2.2. This is not surprising, as weighted aspect ratio error is the optimization criterion in the algorithm. ID-Map ignores the weighted aspect ratio error, and consequentially, produces significant aspect ratio error figures, see Figure 4.13 (f).

As noted above, a potential drawback of the ID-Map split policy *A* is that that for certain pairs of data elements, a reversal of i-measures and corresponding rectangle sizes may occur. It was proposed that to minimize this counter-intuitive effect in the ID-Map/policy *A* layout, a resorting of rectangles can take place prior to assigning time series elements to display rectangles. We also observed the behavior of the ordering error, as well as the number of reversal cases, for the synthetic data set. Figure A.13 in the Appendix reports the respective error numbers. To summarize, resorting is an option to significantly reduce the degree of size to i-measure reversal cases, while also improving the ordering error numbers.

Regarding the remaining objectives *space-filling*, *overlap-freeness*, and *hierarchical rectangular containment*, we note that all algorithms provide these, considering their definition.

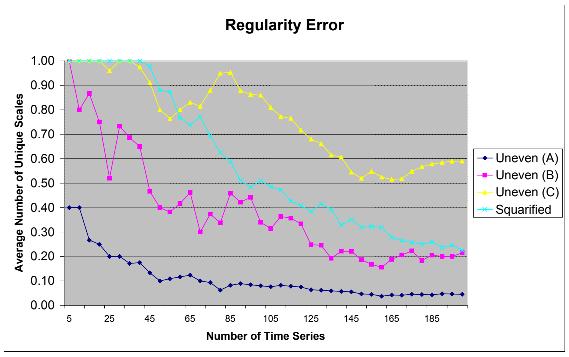
We conclude the experimental evaluation by noting that ID-Map manages to produce regular displays which are suited for comparative analysis of time series data, as it provides a low number of different axis scales (both for individual axes x and y, as well as the combination of scales). Also, it produces low errors regarding the positioning of rectangles according to *i*-measure relationships. Regarding linear proportionality and data-dependent target aspect ratios, the Squarified algorithm seems the better choice, if such criteria are primarily desired.

4.2.7 Conclusions

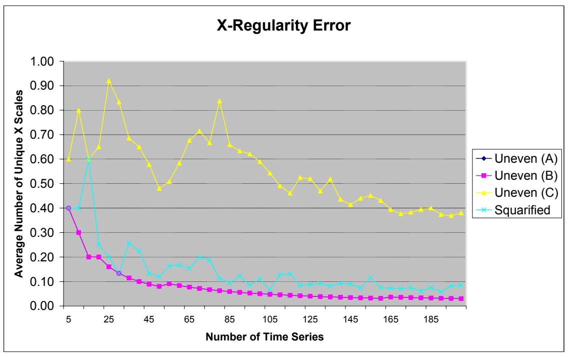
To summarize Section 4.2, we note that ID-Map provides high display regularity in terms of good rectangle alignment and a low number of different rectangle scales. It largely obeys the top-down, left-right ordering of elements in its layout. We conclude that the algorithm is a simple, yet effective layout scheme for analysis tasks on sets of time series data, as high display regularity supports comparing intervals in time and function value of the line and bar charts, and differentiations in position and size serve to indicate data-dependent importance scales.

By configuring the algorithm to different splitting policies, the user can control the tradeoff between size proportionality and display regularity according to preference. We point out that in many applications it is well possible to trade off linear proportionality between time series importance and rectangle sizes for regularity. This is because often, appropriate importance measures are heuristically obtained, and perfect quantitative reflection might not be meaningful anyway. Furthermore, sometimes just ordinal importance relations among time series data are considered by the analyst.

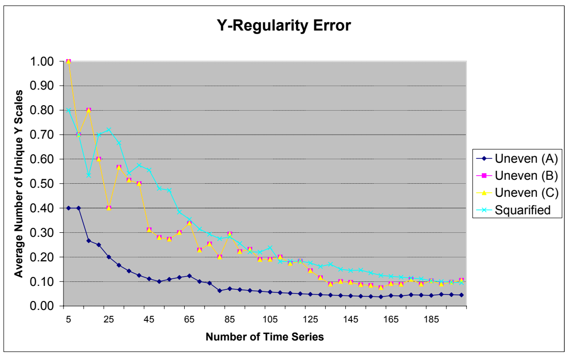
We note that in the experimental comparison, we have considered the non-hierarchical case only. We expect that for sufficiently balanced hierarchical structures, we will obtain good results by recursively applying the ID-Map algorithm throughout the data hierarchy. Of course, application data might be arbitrarily skewed, and we can construct cases where ID-Map runs



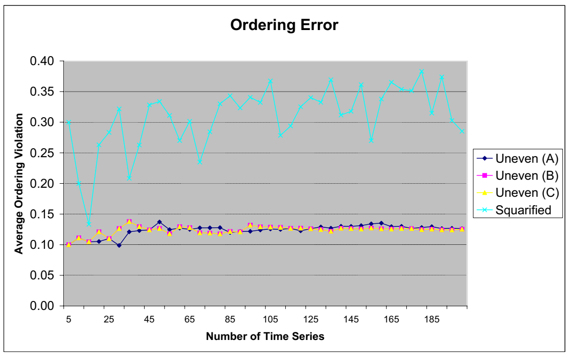
(a) Regularity errors.



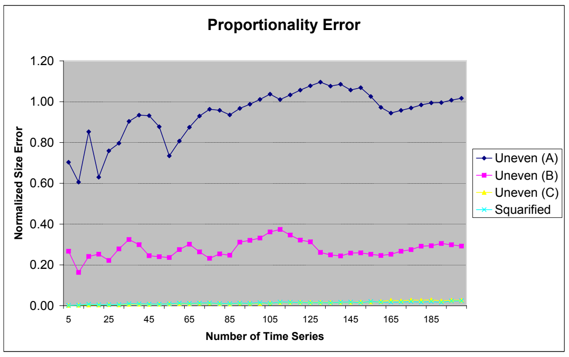
(b) X-regularity errors.



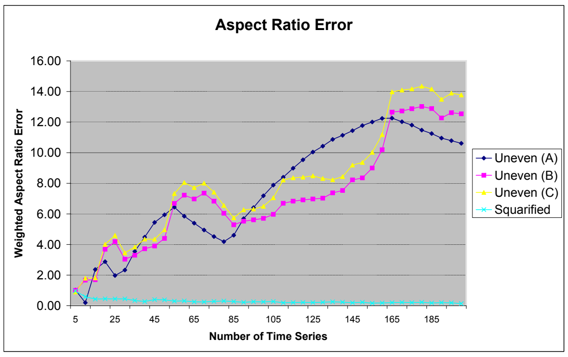
(c) Y-regularity errors.



(d) Ordering errors.



(e) Proportionality errors.



(f) Weighted aspect ratio errors.

Figure 4.13: Error functions for layout of 5 to 200 rectangles, using the uneven splitting masks and assuming *i*-measures uniformly distributed in $[0,1]$.

into degenerate layouts. Consider a totally sorted time series tree, which takes the form of a binary tree degenerated to a right-hand-side list. Using ID-Map split policy A would result in a highly irregular allocation of display space to time series rectangles, such that a small fraction of rectangles consume much of the display space. To circumvent such degenerate situations, we need additional rules to detect and handle the layout of the time series tree, e.g., by using additional and possibly data-dependent layout masks. This point, as well as finding methods for achieving display regularity and better size-proportionality at the same time using data-dependent split point quantization schemes, constitute interesting future research directions.

In certain domains, e.g., in a network monitoring application, high data update rates are given which dynamically change underlying i-measures and thus, call for dynamic updates to the display layout. How to provide good transitions for updates to the ID-Map display is also an open problem which we would like to address in future work.

4.3 Regular layout generation with Grid TreeMaps

In Section 4.2, we introduced the ID-Map algorithm which aimed at improving the regularity of the resulting display partitioning in a TreeMap-like layout. While regularity was improved, still different rectangle sizes and aspect ratios emerged in the display, where rectangle sizes were used to reflect importance relationships. In some applications, regularity needs to be maximized (*all* rectangles need to be of the same dimensions), even at the expense of proportionality (as then, the *size* of the rectangles cannot be used to communicate quantitative relationships any more). The Quantum TreeMap algorithm [9] quantizes the output of the standard TreeMap algorithm to produce layouts where each data element (rectangle) corresponds exactly to one cell on a uniform grid, providing maximum regularity by definition. In this Section, we analyze the space-efficiency characteristic to the Quantum TreeMap algorithm. Based on the analysis, we propose an improved quantized TreeMap variant including three supporting rendering methods. We compare it against the Quantum TreeMap, and conclude that the display is competitive with the Quantum TreeMap, outperforming it in terms of display efficiency for a range of data distributions.

The specific outline of the Section is as follows. In Section 4.3.1, we give a discussion of the disadvantage of the continuous slice-and-dice TreeMap in providing regular displays. We also discuss the Quantum TreeMap algorithm in detail. In Section 4.3.2, we then develop a family of new TreeMap algorithms based on analysis of the Quantum TreeMap. In Section 4.3.3, we apply three instantiations of our layout algorithm family as well as the Quantum TreeMap on a hierarchically structured time series data set, demonstrating the usefulness of our algorithm. In Section 4.3.4, we go on to evaluate the space efficiency characteristics of our algorithms by performing a range of layout experiments based on synthetic data sets modeling different classes of hierarchic structures. Finally, Section 4.3.5 concludes.

4.3.1 Continuous and Quantum TreeMap algorithms

We next illustrate the regularity problem of the Continuous TreeMap by an application example. We then discuss the Quantum TreeMap approach for generating regular dispalys.

Regularity Problem of the Continuous TreeMap

The standard *continuous TreeMap* algorithm (here abbreviated as CTM), as recalled in Section 4.1.3, is a simple yet powerful layout technique supporting hierarchically structured data. However, it has disadvantages regarding the regularity of the produced displays. Due to splitting of rectangles in linear proportion to sums of underlying weights, the continuous TreeMap may produce tessellations consisting of many different rectangle aspect ratios, depending on input data characteristics. Then, it may be hard for the user to select, compare, and trace rectangles throughout the hierarchy [94, 11, 9].

Consider for example the problem of laying out sets of hierarchically structured time series, or more generally, bar chart data. Then, in order to be able to compare time intervals and value magnitudes, there must not be too many different scales present in the display tessellation. Figure 4.14 shows an example of visualizing a set of hierarchically structured time series charts using the CTM algorithm. Clearly, it is hard to compare periods in time and value magnitudes throughout the display, as practically every time series rectangle is assigned unique scales for its *x* and *y* axes.

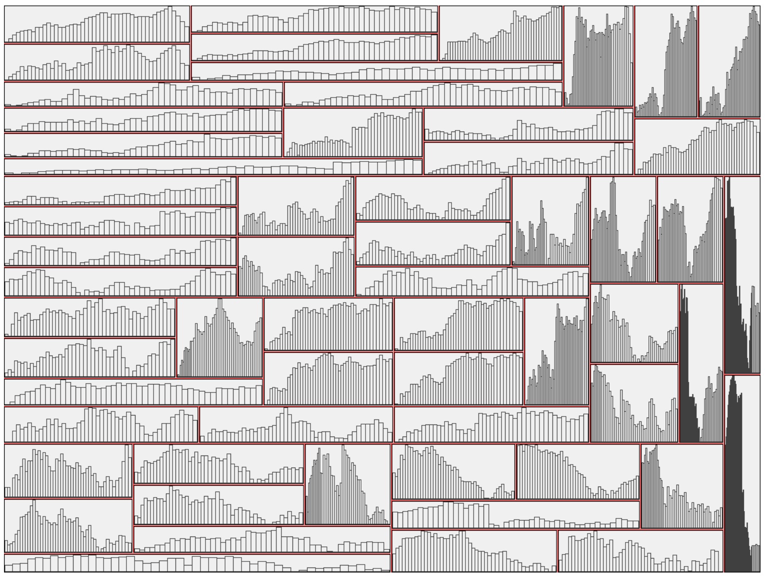


Figure 4.14: In practice, the CTM algorithm produces many different aspect ratios and *x*/*y*-scales in its layout. Thereby, it is difficult to present important data types such as time series inside the obtained rectangles in a useful way. The image shows the CTM layout of the data set used in Section 4.3.3. Clearly, it is hard to effectively compare the data set elements.

Quantum TreeMaps for Guaranteeing Consistent Aspect Ratios

It has been recognized that besides *optimizing* aspect ratios, in many applications it is desirable to *guarantee* constant size and aspect ratios for all of the rectangles to be laid out. This is motivated by supporting visual comparability when displaying multi-dimensional or abstract data types such as images [9] or time series data [26], which call for regularity-providing layout generation algorithms. In order to support regularity in TreeMap displays, in [9, 8] it was proposed to perform a quantization of the output of the TreeMap algorithm, where width and height of the resulting rectangles are allowed to assume only integer multiples of predefined height and width quanta. The so-called *Quantum TreeMap* (here abbreviated as QTM) guarantees consistent rectangle aspect ratios, and by design places all data elements on unique positions on a global regular grid. The basic idea is to first perform continuous splitting, and then, based on this (intermediate) result, search for a good quantization of the split points allowing to lay out the number of data elements requested by the given data subset. It has to be defined what constitutes a good quantization, but usually, the amount of wasted space, or the deviation from the continuous rectangle in terms of aspect ratio or symmetric area difference are candidates for optimization.

The Quantum TreeMap principle can be applied on any intermediate input TreeMap layout. In [9] quantization was done for the so-called Strip TreeMap layout, and in [8] it was done for the so-called Ordered TreeMap layout. Both layouts are TreeMap variants aiming at optimizing rectangle aspect ratios. We here consider quantization of the original slice-and-dice layout. Figure 4.15 illustrates the QTM technique applied on the slice-and-dice TreeMap algorithm.

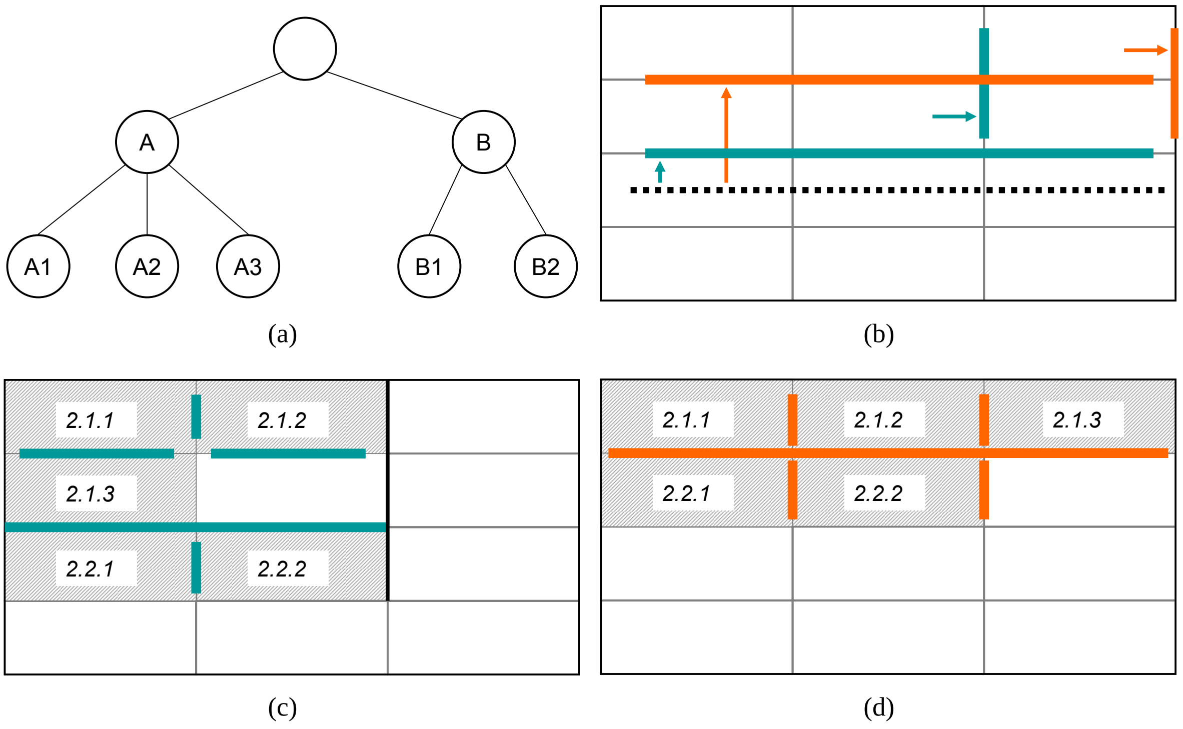


Figure 4.15: The Quantum TreeMap (QTM) quantizes the results of continuous splitting to a suitable amount of rows and columns on a global regular grid. Image (b) shows a global regular discretization grid overlaid on an initial CTM split decision (dashed line), based on the simple data distribution indicated in (a). Based on the concrete searching heuristic used, different allocation outcomes are possible. The blue and orange split lines included in (b) are possible based on different quantization rules. (c) and (d) illustrate the final results. As opposed to the CTM-based result shown in Figure 4.2(b), the quantized displays are not space-filling any more.

By design, QTM does not obey the space-filling property of the original algorithm. In [9], the authors performed an experimental analysis of the *display overhead* metric, which is the amount of display space not occupied by data elements. The authors concluded that display overhead is not critical when laying out data sets consisting of many elements per hierarchical group. In this Section, we will revisit the QTM algorithm in an experimental analysis, and will compare it against our *Grid TreeMap* algorithm developed in the next Section.

4.3.2 Grid TreeMap algorithm

The Quantum TreeMap provides layouts with guaranteed constant size and aspect ratios of the data rectangles, and with consistent alignment of the elements on a global grid. There exist two sources O for potential display overhead (loss in display utilization efficiency):

O_1 QTM is not always able to quantize layout partitions to grid dimensions corresponding to the exact number of elements to be laid out (grid cells may be left unoccupied; first source of potential display overhead).

O_2 QTM performs on-the-fly quantization of split points. During processing of the algorithm, the resulting global layout may grow or shrink in width and height, deviating from the initial root display rectangle. The final result may have to be be (isotropically) scaled back into the original root display rectangle. Whenever the aspect ratios of the root display and the resulting grid differ, then display overhead due to scaling will occur (second source of potential display overhead).

We presume that based on the hierarchic characteristics of the input data set, significant display overhead may occur in QTM. We therefore researched an alternative quantization scheme we call the *Grid TreeMap* (GTM). Instead of first performing the continuous TreeMap on the root display rectangle and then quantizing the result to the grid, we go the other way round. We first decompose the root display rectangle to a grid of sufficient dimensionality, and then perform the TreeMap algorithm directly on the resulting grid. The grid dimensionality is found such that (a) the number of rows and columns is sufficient to hold all the data elements, (b) no more than one row or one column is only partially occupied with data elements, and (c) the aspect ratio of the resulting grid slots matches a predefined (targeted) aspect ratio as closely as possible. We then perform the TreeMap algorithm on this grid by alternatingly scanning rows or columns to assign data elements to slots on the grid (cf. Figure 4.16). We presume this approach to be more space efficient than QTM for certain data set characteristics, at the same time producing regular layouts. We recognize that by using scan based assignment of elements to grid slots, we will encounter “stair-step” effects, thus split boundaries (hierarchical separators) are not guaranteed to be straight lines anymore. It will be the responsibility of the following rendering techniques to compensate for the loss of the straight line property, which by design is provided in the CTM and QTM methods.

Rendering Grid TreeMaps

One property of the Quantum TreeMap not present in the Grid TreeMap is that in the former, groups of elements are separated by straight lines. With QTM, the TreeMap user can trace connections of the straight lines to quickly recognize hierarchical relationships present in the data. With GTM, which performs scan line based partitioning of grid slots, we do not have the straight line separator property, and it is expected that this negatively affects the ready perceivability of hierarchic relationships without providing appropriate visual support. Therefore, we have to pay close attention to the way by which we indicate the hierarchy separators. We have experimented with several different separating schemes, and found three of them effective for rendering GTM layouts of different data hierarchy patterns:

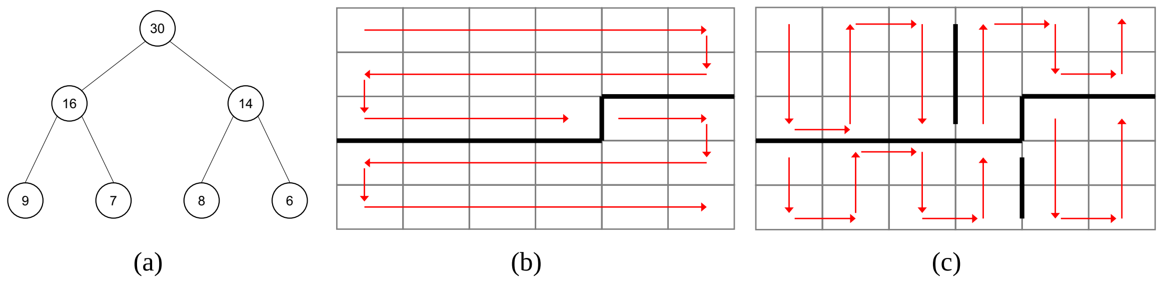


Figure 4.16: With the *Grid TreeMap* (GTM), first the display rectangle is decomposed into a grid of sufficient dimensionality to hold all data set elements. Then, on this grid GTM performs slicing and dicing by alternatingly scanning rows and columns. The right images show the GTM algorithm allocating the first (b) and the second (c) levels of the hierarchy denoted in image (a).

1. *GTM-N* (Nested): Draw enclosing boundaries around groups of elements. Free the required display space by simultaneously scaling down all grid rectangles such that the display can accommodate all boundaries without over plotting.
2. *GTM-S* (Split Lines): Keep the distance between adjacent grid slots constant, and indicate split line depth (hierarchy level) by appropriately setting the drawing attributes color, thickness, and shape of the respective split lines.
3. *GTM-B* (Burst): Indicate hierarchy by varying the distance between groups of elements with the splitting depth. Let greater distance between groups of rectangles indicates higher (closer to the root) hierarchic separations.

GTM-Split keeps the amount of display space dedicated to hierarchic information constant, but is not expected to scale for arbitrary deep hierarchies due to the fixed amount of display space reserved for the split lines. GTM-Nest and GTM-Burst do not guarantee a bound for the fraction of display space allocated for indicating arbitrarily deep or wide hierarchical relationships, but we will see in the next Sections that the techniques work well in practice. The proposed rendering techniques are illustrated in Figure 4.17. In the next sections, we will apply the GTM variants on a real-world data set, and perform experiments on synthetic data assessing the quality of the layout algorithms.

4.3.3 Application

We here present the application of the QTM and GTM algorithms on a data set consisting of daily stock price time series obtained from [84]. We like to render the data using the familiar bar chart drawing technique, which requires a high degree of regularity in the display to support effective comparative analysis. To allow the analyst to gain a quick overview over a set of bar charts, the corresponding layouts should produce regular tessellations. Note that a continuous layout as provided by the classical CTM (c.f. Figure 4.14) is clearly inappropriate for this task, and we recognize the need for quantized layouts.

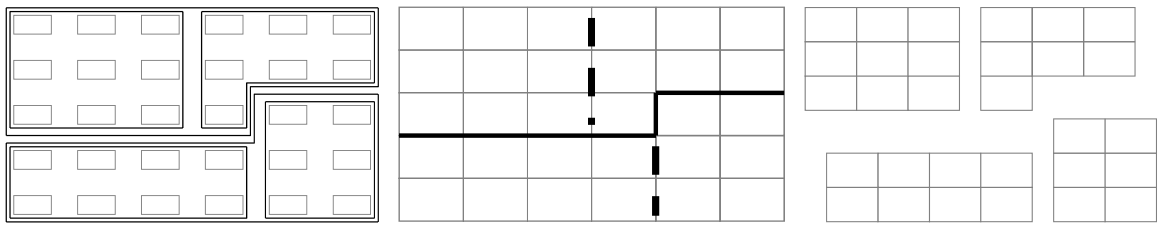

(a) Nest (b) Split Lines (c) Burst

Figure 4.17: Three rendering methods for visualizing hierarchical relationships for Grid TreeMap layouts. Cf. also Figure 4.16.

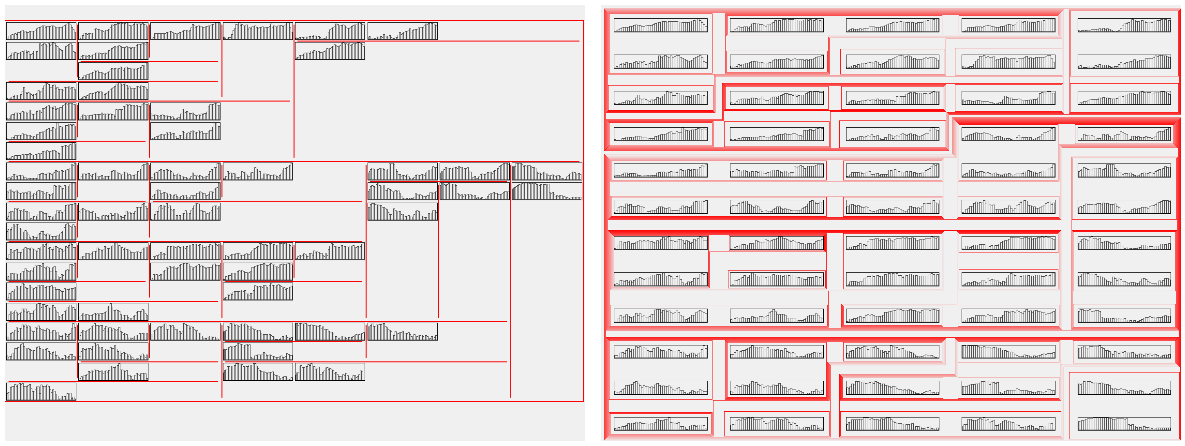
In order to impose a meaningful hierarchical ordering on the data set, as a preprocessing step we apply the *Hierarchical Agglomerative Clustering Algorithm* (HAC) [37] on the data set. The HAC iteratively merges pairs of sets of elements which exhibit highest similarity at the given iteration. The result is a binary tree which when properly visualized is a nice tool for analyzing similarity relationships in a set of objects on which a meaningful similarity scale can be defined. Note that binary-tree hierarchies are a stress test for the layout algorithms. The height h of a balanced tree with fanout f containing n leaves (data elements) logarithmically depends on f, as $h = log_f n$ holds. This in turn means that binary trees give deepest hierarchies (more hierarchy levels) as compared to trees with fanout larger than 2. Thereby, the display has to communicate more information and consequentially, has to allocate more display space resources for hierarchy visualization.

For generating the layouts, we used a subset of the time series database consisting of 60 series with 48 values each. We configured the HAC algorithm to use the Euclidean distance between time series normalized for offset and scale. Such normalization is a useful preprocessing step when calculating time series similarity [56]. The resulting tree is of height 11, which leads to a high degree of data partitioning, stressing the layout algorithms. We set the targeted aspect ratio (as width : height) to 4 : 1, which seems reasonable for rendering bar charts of the considered length. We set the root display to 1200×900 pixels. Where needed, we set indention of grid rectangles and thickness of splitting / nesting lines to reasonable parameters which try to be space efficient, and at the same time support perception of hierarchical separation relationships.

In the following, we consider as *display overhead* the fraction of the root display rectangle which is not occupied by time series (bar chart) rectangles. Specifically, empty space as well as space occupied by split lines or used for element separation contributes to display overhead. In case of QTM and GTM-B which may return layouts violating the aspect ratio of the root display rectangle , we isotropically scale back the layout to fit the root display rectangle prior to measuring display overhead.

Quantum TreeMap

Figure 4.18 (a) gives the result of the QTM layout of the hierarchically clustered data set. The display aligns all time series rectangles on a global grid, and guarantees that all rectangles



(a) Quantum TreeMap (QTM) (b) Nested Grid TreeMap (GTM-N)

Figure 4.18: Quantum TreeMap (a) and nested Grid TreeMap (b) layouts of 60 time series, organized by similarity using the hierarchic agglomerative clustering algorithm (HAC) for preprocessing.

have the same size and aspect ratio. Due to the binary branching in the HAC tree, the tree height is significant and results in a complex set of hierarchical relationships among the time series. This in turn stresses the layout in terms of display overhead. When allocating inner tree nodes, QTM searches for grid layouts which locally minimize display overhead. Therefore, the total display overhead of a given data partition not only depends on the current quantization decision made for the given tree node. It also depends on the quantization steps performed when subsequently laying out the subtrees rooted at the given node, as well as the layout of the siblings of the considered tree node. By design, QTM is a greedy layout algorithm which does not consider global effects in the local layout decisions it makes, and consequently, the display overhead is expected to be high if the tree structure is complex and deep. The overall overhead in the QTM display of this data set amounts to 71%, which is significant. On the other hand, the QTM retains the straight line property when separating groups of elements in the display. This makes it rather easy for the analyst to trace partition borders in order to understand the hierarchical relationships in the data set.

A more subtle problem coming along with the uneven distribution of elements to embedding rectangles is that of potentially reduced perceivability of *balancing relationships*. Note that in the QTM display, the size of rectangular display partitions does not have to be proportional to the number of contained data elements. E.g., consider in Figure 4.18 (a) the first top-level partition of the data set indicated along the straight horizontal line at about $\frac{1}{3}$ height. Within this partition, the data is subsequently separated into two groups containing 15 and 3 elements, respectively (the left and the right partition in the topmost display partition). As a result of the final quantization layout, the second-level (vertical line) separator between the two groups is placed at $\frac{1}{2}$ display width. At first, this might mislead to assuming both partitions to be balanced, as the display areas in the left and right hand side partitions are equal. Yet, the population of these areas with data elements is quite uneven (15 and 3 elements, respectively).

Nested Grid TreeMap (GTM-N)

Figure 4.18 (right) gives the result of the GTM-N (nested) layout of the data set. The algorithm first generates the Grid TreeMap layout, and then draws thin boundaries around groups of elements. Here, we drew boundaries of 1 pixel width around groups of elements which belong together along the hierarchy. We also indent adjacent lines by 1 pixel. The deeper a given hierarchy, the more grouping lines have to be drawn. Consequently, the bar chart boxes have to be scaled down to free up the space required for border drawing. We chose to simultaneously scale down all rectangles by the same amount as required by the hierarchy, to keep the data elements at the same size and consistently positioned on the global grid.

The display overhead metric amounts to roughly 70%, which is comparable to the QTM result. The reason is the depth of the hierarchy. As the HAC tree is of height 11, and considering we have to allocate at least 2 pixel per hierarchic boundary on each side of the respective rectangles, it is not surprising that the rectangles are scaled down significantly. On the other hand, as compared to QTM most of the space not occupied with bar charts is occupied by nesting boundaries, indicating hierarchic information. We note that the topmost separation levels are clearly perceivable, as there is a visual cumulative effect when many lines are drawn in parallel. Also, the number of elements contained in subtree partitions is better perceivable, as there is no “dead space” like in the QTM. It is somewhat more difficult to trace the lower separation borders. Eventually, we have to rely on the pixel-level to completely assess the full hierarchical structure.

Hierarchic Split Lines Grid TreeMap (GTM-S)

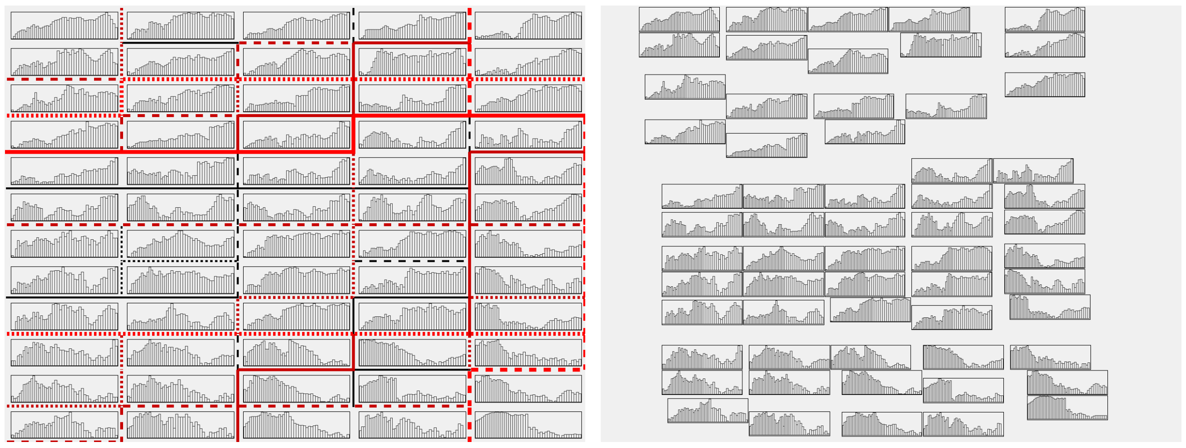

(a) Split Line Grid TreeMap (GTM-S) (b) Burst Grid TreeMap (GTM-B)

Figure 4.19: Splitline-based (a) and Burst Grid TreeMap (b) layouts of the same data as in Figure 4.18.

The Grid TreeMap with hierarchic split lines (GTM-S) uses visual attributes of split lines of fixed maximal width in order to communicate hierarchical separations throughout the data tree. Designing split lines which are visually discriminating and at the same time are capable to encode ordinal relationships (tree levels) is not an easy task. We experimented with several different settings, and found the scheme given in Figure 4.20 a good compromise, although other schemes are possible. We employ color, split line width, and dashing as visual attributes. Figure 4.19 (a) shows the resulting Grid TreeMap layout using hierarchic split lines. We have indented the rectangles by 10 Pixels each to free space for drawing the hierarchical split lines. We notice that the top separation levels in the hierarchy (i.e., the bright most split lines) are best perceived. Tracing the lower-level split lines is somewhat harder, as the lines get increasingly thinner, but still it is possible to do so using the legend. The overall display overhead of 31% is reasonably smaller than the one resulting from QTM and GTM-N.

Burst Grid TreeMap (GTM-B)

GTM-N and GTM-S draw enclosing boundaries and hierarchic split lines to communicate hierarchic relationships among subsets of data elements. Another alternative we tested is to visualize the separation relationships by depth-dependent horizontal and vertical spacing between the individual data element groups. In our strategy, groups of objects get "burst apart" such that groups separated on higher hierarchy levels get separated by more horizontal or vertical space between them. We implemented a policy which during the layout generation performs the bursting by translating rectangle partitions: In a sequence of scanning rows (columns), the partitions get burst apart along vertical (horizontal) directions. We model the "bursting distance" $d(l)$ separating the sibling element groups rooted at an inner tree node of level l in a tree of height h as linearly depending on the respective hierarchy level l:

$$d(l) = \frac{h-l}{h} \times D^{max}$$

It results that the topmost partitions (rooted at hierarchy level $l = 0$) get allocated the distance $d(l = 0) = 1 \times D^{max}$ for separation, while the consecutive split levels are allocated a linearly decreasing amount of maximal bursting distance. In practice, we get good results by setting $D^{max} = min(width, height)$ of the grid cells.

Figure 4.19 (b) shows the GTM-B layout of the clustered data set. Translating the groups proportional to the splitting level supports perceivability of the splitting hierarchies. There is a tradeoff between the space-efficiency and the usage of bursting: Larger burst distances (implemented by higher $D^{max} settings$) consume more display space leading to higher display overhead rates for a given data set. Still, display utilization is reasonable. In this example, the space used for bursting was 53% of display space. We note that other than in the previous variants, the individual rectangles are not aligned on a global grid anymore due to the continuous calculation of the amount of translation. Note also that the resulting layout may need to be scaled back into the original root display area (like QTM), possibly resulting in scaling overhead O_2 (c.f. Section 4.3.2).

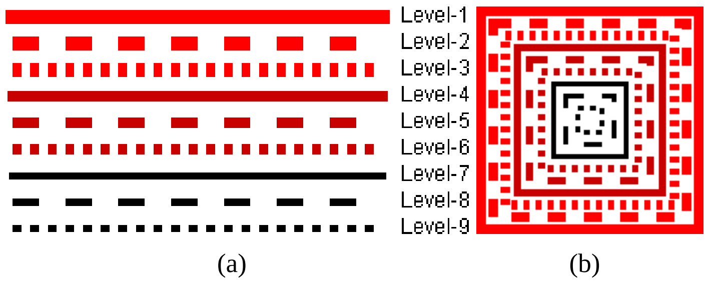


Figure 4.20: Split line legend for the GTM-S layout in Figure 4.19 (a).

4.3.4 Evaluation

We recognize two important criteria for evaluating the quality of the proposed layout algorithms. One criteria is the *effective perceivability of hierarchical relationships* by the user. Evaluating this metric formally is difficult as it would require a carefully designed user study which at present is beyond our resources. We note that we regard the discussion in Section 4.3.3 as supporting the effectiveness of the GTM techniques in visualizing hierarchical relationships in regular displays. A second obvious evaluation criterion is the *display utilization* which contrary to the hierarchy perceivability can be automatically evaluated using the display overhead metric. We experimentally measured the display overhead during batch experiments to layout synthetically generated data sets of different size and hierarchy characteristics. Specifically, we considered different balanced and unbalanced hierarchy models for laying out 5 up to 200 time series objects inside a 1200×900 root display area.

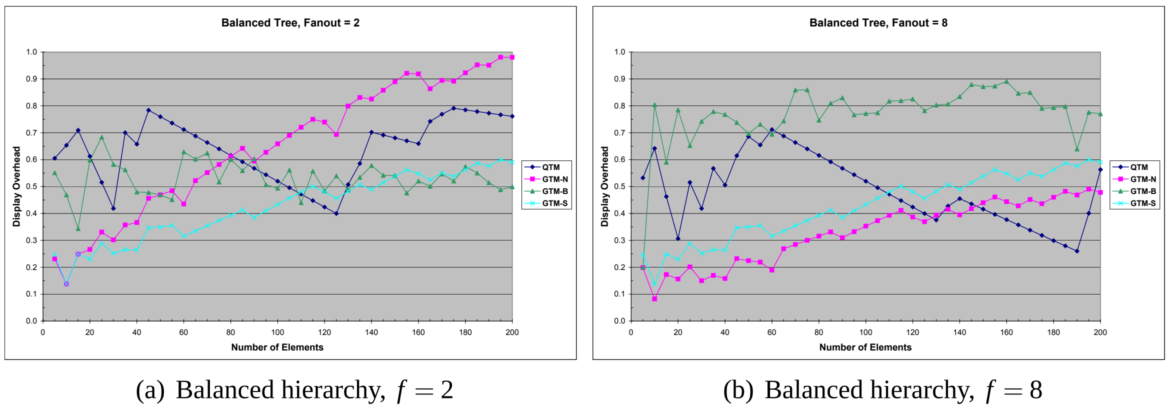


Figure 4.21: Display overhead for GTM-{N,B,S} and QTM, for balanced hierarchies with tree fanouts of 2 (a) and 8 (b), respectively.

Balanced hierarchies

We first evaluate the display overhead modeling the hierarchy as a *balanced* tree with constant fanout f at each inner tree node. Figure 4.21 gives the display overhead results for fanout settings $f = 2$ (left chart) and $f = 8$ (right chart), respectively.

GTM-Nest and GTM-Split clearly show decreasing space efficiency for growing numbers of elements. As the tree height grows with the number of elements, GTM-Nest has to allocate more space for boundary drawing, which is freed by scaling down the grid cells. In case of the binary tree ($f = 2$), already at 100 elements, space required for boundaries consumes more than 60% of display resources. Practically, rendering more that 100 elements in the given tree structure and root display (1200×900) using GTM-N thereby is problematic. Again, as noted in the preceding Section, using binary trees is a stress test for the Grid TreeMap. In case of trees of higher fanout this problem diminishes, as then tree height grows less aggressive. In case of $f = 8$, GTM-N's space efficiency improves to at most 50% of display overhead for 200 elements.

GTM-Split allocates constant inter-cell distances irrespective of the hierarchy structure (note the identical display overhead curves in Figures 4.21 and 4.22). As the number of cells grows, so does the amount of inter-cell spacing required for drawing hierarchic split lines. The amount of space allocated for hierarchy visualization amounts to 20% for few elements, up to a maximum of 60% for 200 elements.

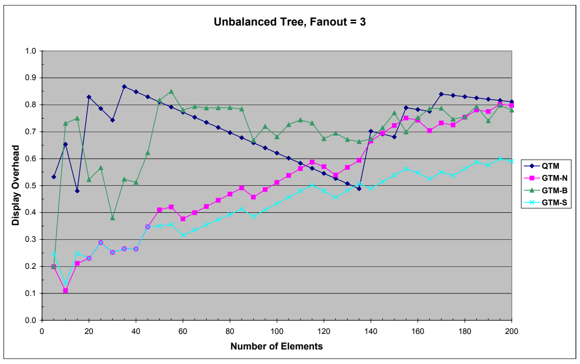
GTM-Burst indicates hierarchical separations by translating partitions of elements apart along opposite directions. The amount of translation negatively depends of the hierarchy level. Therefore, trees with high fanout at higher tree levels (close to the root) consume the most space for bursting. For $f = 2$ we observe reasonable display overhead numbers around 50% for most data sizes. With $f = 8$ the amount of space required is rather high (up to 80%) for most data sizes.

The Quantum TreeMap results show an oscillating display usage pattern. In our implementation, we let QTM search for a good local quantization by either extending the number of columns *or* the number of rows to accommodate the number of elements to lay out in each iteration, and using the *area* of the alternatives as the selection criteria. In the hierarchy model considered here, a large part of the display overhead in QTM is due to scaling overhead (overhead source O_2 in Section 4.3.2). In our experiments, many resulting QTM layouts are significantly larger in width than in height, so isotropically scaling back the result into the 1200×900 display sacrifices vertical space. Due to the balanced structure of the hierarchy, overhead source O_1 makes up only a smaller part of the loss in display space. The latter finding is in accordance with the results from [9]. With the data considered here, QTM overhead oscillates around 60% (50%) at $f = 2$ ($f = 8$), with significant variance.

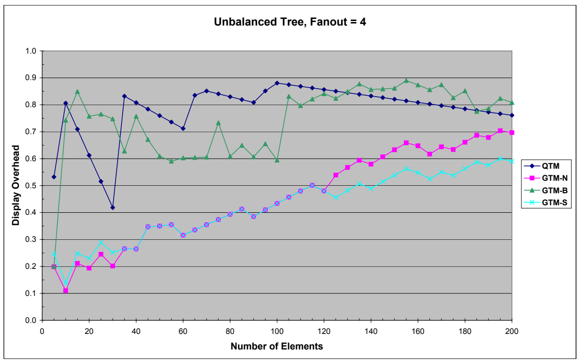
Unbalanced hierarchies

We also evaluated the display overhead when modeling the hierarchy as an *unbalanced* tree. We keep fanout f at each inner tree node constant but populate the leftmost child of each inner tree node with the triple number of leaf elements rooted at the respective node, as compared to its right hand side siblings. Figure 4.21 gives the results for fanout settings $f = 3$ (left chart) and $f = 4$ (right chart), respectively, for this tree model. We chose these fanout settings to get tree structures of sufficient depth for our experiments. We obtain unbalanced trees where data is unevenly distributed along the tree paths.

We observe the same metric readings for GTM-Split, as by design its space occupancy is invariant with respect to tree structure. GTM-Nest, due to the unbalanced hierarchy structure, has to dedicate less display space to the group boundaries as compared to the low-fanout tree results in Figure 4.21 (left). We observe maximal hierarchy overhead of about 80% (70%) for 200 elements using a fanout of $f = 3$ ($f = 4$). We note that for element counts up to about 140, in both hierarchies we still have around $100\% - 60\% = 40\%$ space left for element rendering. GTM-Burst is significantly impacted by the deeper hierarchies resulting from the unbalanced tree structure, as it has to bust apart more element partitions, and $d(l)$ is diminishing at a slower rate (cf. Section 4.3.3). Also, O_2 overhead occurs for GTM-Burst. Maximum overhead is in the range of 80% for both fanout settings, and for most data sizes. An exception is the interval between 40 and 100 elements with $f = 4$, where overhead is around 60%.



(a) Unbalanced hierarchy, $f = 3$



(b) Unbalanced hierarchy, $f = 4$

Figure 4.22: Display overhead for GTM-{N,B,S} and QTM, for unbalanced hierarchies with tree fanouts of 3 (a) and 4 (b), respectively.

Experiments Summary

We can summarize the experimental findings as follows. GTM-Burst delivers efficient layouts in terms of the hierarchy overhead metric for hierarchical structures of comparably low fanout, as then, less space is allocated for bursting apart partitions (cf. Section 4.3.3). GTM-Nest delivers efficient layouts in the opposite case, namely, when fanout is high and resulting tree depth is comparably low. The bottleneck factor in GTM-Nest is the tree depth, as the maximal hierarchic level in the data set dictates the amount of down scaling that has to be applied on all grid cell rectangles (recall that we target regular layouts, so we demand uniform scales for all rectangles). GTM-Burst and GTM-Nest therefore are recommended for complementary use depending on the hierarchical structure to be visualized. GTM-Split provides a middle ground and is promising in case the number of elements in the data set is not too large, irrespective of the hierarchical structure present, up to a limit dictated by the number of levels we can visually discriminate using split line drawing attributes. We estimate the latter limit to be around 10 levels, which should be sufficient to support many real-world applications.

We also evaluated the Quantum TreeMap algorithm as a base line competitor for our algorithms. As the charts show, there are data constellations where either one of the Grid TreeMap algorithms outperforms the QTM, in term of the display overhead metric. We conclude that Grid TreeMaps are a practical option for visualizing regularity-requiring, hierarchical data sets. Depending on the nature of the data in terms of hierarchical structure and data set size, the most appropriate GTM variant may be selected interactively by the user or automatically by the visualization system as based on the display overhead metric. Figure 4.23 summarizes the experimental findings in a table.

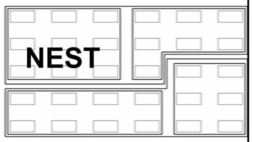
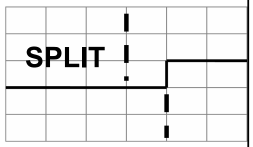
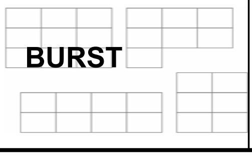
Method	Deep Tree (f=2)	Shallow Tree (f=8)	Summary
	Bad due to constant indention 50% overhead @ 60 series	Good due to less indention 40% overhead @ 120 series 50% overhead @ 200 series	Good for shallow trees Hierarchy perception rather easy
	Invariant 50% overhead @ 140 series 60% overhead @ 200 series	Invariant dito	Independent of hierarchic structure Effectiveness probably limited to ~5-10 levels
	Good due to lower numbers of partitions on higher levels Roughly around 50% for all data set sizes considered	Bad (conversely) >80% overhead @ 60 series	Good for shallow and low-fanout trees Room for additional visual clues

Figure 4.23: Summary of the display overhead experiments.

Regarding the rather high degree of display overhead in QTM, we note that the numbers may be due to our chosen implementation (cf. Section 4.3.4). Other quantization heuristics are possible, and we have not attempted to optimize the searching strategy to the considered data set characteristics. We note that traditionally, TreeMap algorithms perform greedy optimization, and this is rather a feature than a drawback of the techniques. The TreeMap philosophy is to provide fast algorithms suited for online layout generation. We therefore have not attempted to further optimize the quantization in our QTM implementation in a back-tracking manner, as we feel this would not be in accordance with the tradition of TreeMap algorithms. We rather take QTM as a reference for assessing the performance of our Grid TreeMap variants, which are designed to give compact representations based on the linear nature of the TreeMap algorithm family.

4.3.5 Conclusions

We have presented novel layout techniques based on the idea of applying the recursive TreeMap algorithm on a regular grid of display cells. The introduced Grid TreeMap layouts provide a high degree of regularity, and thereby are suited for layout of hierarchically structured, complex data sets requiring regularity such as time series data, images, and multidimensional data, among others. Depending on the nature of the data set in terms of hierarchy fanout, degree of tree balance, and data set size, different Grid TreeMap rendering variants using boundaries, split lines, or translation-based separation, can be recommended. We applied the techniques for visualizing hierarchically clustered time series data, and we performed synthetic experiments allowing to evaluate the algorithm performance in terms of perceivability of hierarchical relationships and display overhead.

In this Section we have focussed on designing the basic layout generators, and will address improved rendering for the different Grid TreeMap variants in the future. Several options for visually supporting the perceivability of hierarchical structures using cell connectors, indication of bursting directions, usage of shading and cushions and so on are possible. Also, interaction techniques such as mouse-over-based highlighting functionality like implemented in the SequoiaView system [80] seem a promising extension for supporting hierarchical perception and navigation by the user. Evaluation of the effectiveness of hierarchical structures perception should be addressed in a formal user study. In this Section we assumed all the data elements to require the same layout (size and aspect ratio) for their enclosing element rectangles. Another interesting problem would be to design regularity-providing layout generators for data where the elements require differing element layouts.

5 Thesis conclusions

This thesis started by considering the Feature Vector approach for the management of databases of multimedia objects. The FV approach describes complex multimedia objects by points in high-dimensional feature space. Using an appropriate metric defined in FV space, distances in FV space are associated with similarity relationships in object space, which in turn are input to important applications such as content-based retrieval, clustering, and visual database analysis.

For many data types, there is a wealth of FV extractors available to perform the mapping to FV space, and it is usually not clear what the best FV extractors in terms of effectiveness and efficiency of the representation are. Therefore, benchmarking and FV space analysis are needed to address the FV selection and engineering problems. The FV selection problem refers to identifying the most appropriate FV spaces for a given data type and application. The FV engineering problem refers to finding the most efficient representation for a given FV space. In this thesis we have addressed these problems for the 3D model retrieval domain. We have proposed supervised and unsupervised methods for FV selection and engineering, and for increasing the effectiveness of multimedia applications by building appropriate combinations of FVs. Also, novel visualization methods were devised for effectively supporting retrieval and visual analysis in multimedia databases at the FV space and at the object levels. We next recall the main results in each of these thesis areas. We also summarize related problems considered interesting for future research in this context.

Effective retrieval with Feature Vectors (Chapter 2)

In this chapter, we studied FV extraction for 3D models. First, the 3D model data type was introduced, and key desirable properties of 3D FV extractors were identified. An extraction process model was set up that allows systematization of the wealth of different 3D extractors which have been proposed recently. The model considers the most important preprocessing and transformation steps, and accommodates statistical, FV and graph-based descriptions. We then introduced the Konstanz 3D benchmark, a collection of 3D models manually classified according to geometric shape. This benchmark was used to evaluate the discrimination power (effectiveness) of a range of structurally different 3D FV extractors, identifying benchmark-average discrimination power, as well as robustness and efficiency of FV extraction. An image-based extractor was found to deliver best results.

Based on an analysis of the distribution of retrieval performance over subsets of the benchmark (individual query classes), we then researched combinations of FVs as a powerful means of boosting the discrimination performance. Thorough experiments were conducted exploring the combination space with respect to combination structure, combination cardinality, and mode of combining the selected FVs. A scheme based on sums of maximum-normalized distances was found to deliver best results. Furthermore, an approach for building dynamically weighted combinations was proposed. Using a certain amount of supervised information, a query processor builds weights for each FV in the system to execute dynamically weighted queries based on the given query object.

Several promising directions for future research in this context can be identified. Regarding descriptor definition, the problem of how to best achieve rotational invariance is obvious. Solutions to date include using PCA-based object normalization on one hand, and defining feature extractors considering implicit rotational invariant features only on the other hand. Quantifying the tradeoff between both approaches should be done in a careful experiment including an orientation ground truth.

On a larger scale, the specialization of FVs for certain application domains such as CAD or medical applications, where peculiarities of the database (model characteristics) can be exploited, seems promising. Current extractors consider more general, global geometric features, suited for general-purpose VRML models found on the Internet today, like represented in the benchmarks considered in this thesis. Exploiting domain-dependent knowledge should allow design of FV extractors better supporting specific applications.

Also in this context, graph-based descriptors are suited to capture structural information, which is difficult to achieve with FVs. How to robustly extract graph-based information from 3D objects is not clear to date, and also, how these could possibly be encoded in a FV. Furthermore, current FV extractors consider the global model, aiming at calculating global geometric similarity. Local (partial) similarity calculation is a largely unsolved problem.

Regarding the implementation of effective retrieval systems, we see potential for the dynamic FV selection and weighted combination approach. The theoretic analysis showed that significant improvement potential exists over the static combination approach, if the best suited FVs to answer the given query can be found. Heuristics to estimate the suitability are needed. They can be based on supervised information, which could be dynamically collected from the system by monitoring relevance feedback provided by the users. Also, unsupervised estimators based on distance and component distributions are possible. In Section 3.2 we leveraged such a scheme globally for different FV spaces. How to adapt these estimators for the query-local case is an interesting problem which should be addressed.

Visual feature space analysis (Chapter 3)

The second contribution of this thesis was to research visualization techniques suitable for supporting retrieval, browsing, and FV selection and engineering in multimedia databases. First, the Self-Organizing Map (SOM) algorithm was shown to be applicable to organize large multimedia databases described by high-dimensional FVs in a semantically meaningful way. SOMs were generated for collections of 3D models, E-Mails, and time series objects; these data types have not been previously organized elsewhere using SOMs to the best of our knowledge. The obtained SOM visualizations were improved by representing density distribution information by scaling appropriate color space attributes. The proposed SOM views are useful to quickly overview previously unknown databases, and to perform scatter browsing to obtain query objects, and visual database analysis in the wider sense. Also, we proposed SOMs as a suitable means for implementing visual relevance feedback by mapping relevant answer objects back to the map, encouraging the exploration of neighboring SOM regions. This approach is expected to lead to retrieval of more relevant answer objects by the user.

SOMs were then researched as unsupervised estimators of global discrimination power to expect for a given database represented in different FV spaces. The estimators are based on the heterogeneity of cluster distances and FV component distributions, and allow unsupervised selection of suitable FV spaces for a given application in the absence of benchmarking (supervised) information. Regression experiments validated the usefulness of the estimators on real as well as on synthetic data. Unsupervised FV selection and engineering is desirable, as the supervised approach requires a large test database, therefore is expensive, and may not be stable for dynamically changing database content.

We finally explored a new metaphor for projection visualization, the convex hull metaphor. This visualization improves over the symbol cloud approach usually adopted in 2D/3D projection visualization, and supports supervised visual benchmarking, class discrimination analysis, and FV engineering in the wider sense. By a series of systematic experiments, the validity of the proposed metaphor for visual analysis was statistically supported.

There exist promising future research directions in visual analysis support for multimedia retrieval. The integration of standard retrieval interfaces with the SOM view should be further developed. Specifically, more advanced methods for mapping relevant objects onto the SOM are needed, considering also local SOM node characteristics such as node population density, relative discrimination to surrounding SOM regions, and so on in the visualization. Such mapping encourages the user to explore further on the SOM, potentially finding more relevant objects. The performance of such interactive SOM-based exploration in terms of precision/recall statistics should be assessed by appropriate experiments, allowing to compare the SOM-approach to more standard relevance feedback methods from machine learning and information retrieval. Parts of these aspects are currently being worked on [29]. Also, it is planed to further research these directions within the DELOS EU Network of Excellence on Digital Libraries [27] (there: task 4.5a ‘Visual Relevance Feedback’).

Regarding the proposed unsupervised discrimination power estimators, it would be interesting to research additional estimation functions based on entropy, or other information-theoretic measures rating the heterogeneity of distances and components in SOM space. Pixnostics [45] are expected to be a suitable starting point. Also, adaption of these estimators to work in a dynamic query processor should be researched. The idea is to use local SOM-estimators for query-dependent FV selection and weighting. In context of this idea, it would also be interesting to combine the unsupervised estimators with supervised information obtained e.g., during a relevance feedback cycle.

Finally, regarding the convex-hull based projection approach, we feel that the display could be improved by researching more complex visualization metaphors. Such metaphors could include free-form shapes modeling also the local density of the point clouds, and the degree of overlap throughout the shape. An interesting starting point to this end would be to implement the so-called Enridged Contour Maps [91] in our system.

Visual object space analysis (Chapter 4)

The last part of the thesis dealt with visualization support for the object level. Regular layout algorithms were researched which support the structured visualization of database content. Use cases are exploration of database content, and visualization of search results. Regularity providing layout algorithms are needed for many object-level representations of multimedia content, such as images, time series, and text documents.

First, we surveyed the popular TreeMap family of layout algorithms. We classified the algorithms according to their approach to the display regularity objective into continuous, optimizing, and guaranteeing layout algorithms. The main layout objectives the algorithms are based on were analytically compared.

We then proposed a regularity-optimizing algorithm (the ID-Map algorithm), which recursively allocates sets of time series elements into partitions of predefined splitting masks. The splitting masks provide regularity and ordering properties, supporting comparative visual analysis of time series data. Choosing between different prototype mask templates, the display is able to communicate the distribution of certain interestingness measures along a given hierarchy. The algorithm was applied on a number of data sets, demonstrating its usefulness. The technique was experimentally compared with an aspect-ratio optimizing algorithm by means of certain display metrics, outperforming the competitor on a number of metrics.

We finally proposed a regularity-guaranteeing algorithm (the Grid TreeMap algorithm) which operates by slicing and dicing a regular grid of display space. Three different rendering methods were designed supporting the visualization of deep or wide hierarchical structures on the grid. The algorithm was applied on a hierarchically clustered time series data set, and experimentally shown to outperform the slice-and-dice Quantum TreeMap algorithm.

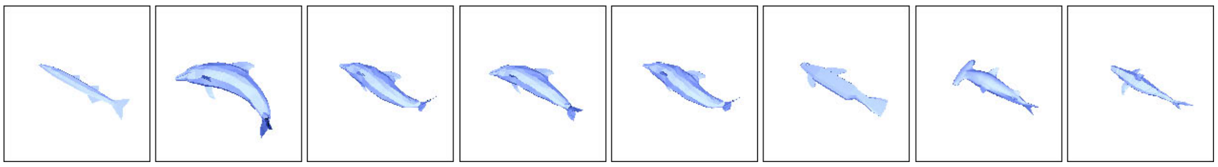
Again, a number of interesting research directions can be identified. First, the ID-Map algorithm could be improved with respect to size-proportionality. Data-dependent discretization of split points is an option to improve this characteristic. As size-proportionality shares a trade-off with the regularity property, precisely quantifying this tradeoff would be helpful for improving importance-driven time series layouts.

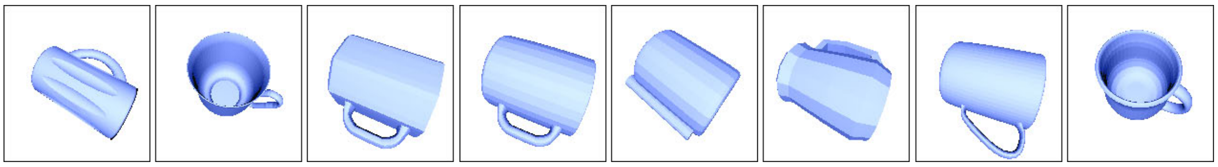
Regarding the Grid TreeMap approach, the three proposed rendering methods split, nest, and burst could also be improved by combination with other hierarchy-supporting visualization techniques. E.g., the cushion metaphor [88] has been shown to work well for supporting perception of complex hierarchical structures. We plan to use cushions or 2.5D-like shading to support the burst and nest approaches. Also, further researching effective combinations of color and shape attributes should lead to improved designs of hierarchical split lines of fixed width to support the split rendering method. Generally, it is a challenging problem to design hierarchical visual clues for inclusion in TreeMap-like displays. Recent results from the IEEE Symposium on Information Visualization 2006 indicate that the potential for improvement in this direction is not yet exhausted.

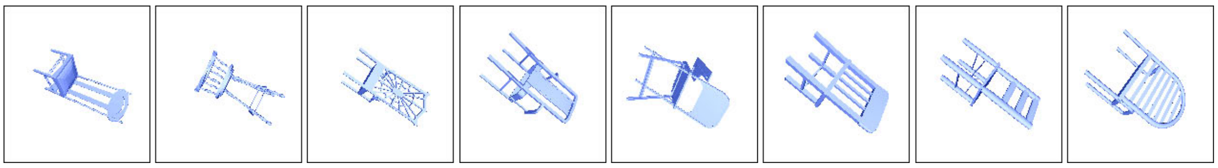
A Appendix

Table A.1: Query classes defined in the KN-DB benchmark. The database contains 1838 objects, out of which 472 were manually assigned to one of 55 query classes.

Class id	Description	# of models	Class id	Description	# of models
1	ants	6	29	submarines	5
2	rabbits	4	30	warships	5
3	cows	7	31	beds	7
4	dogs	4	32	chairs	24
5	fish-like	13	33	office chairs	6
6	bees	5	34	sofas	4
7	CPUs	4	35	benches	3
8	keyboards	8	36	couches	11
9	cans	4	37	axes	4
10	bottles	14	38	glasses	7
11	bowls	4	39	knives	3
12	pots	4	40	screws	3
13	cups	8	41	spoons	3
14	wine glasses	9	42	tables	6
15	teapots	4	43	skulls	3
16	biplanes	5	44	human heads	8
17	helicopters	9	45	human masks	4
18	missiles	16	46	books	4
19	jet planes	18	47	watches	2
20	fighter jet planes	26	48	sand clocks	4
21	propeller planes	10	49	swords	25
22	other planes	4	50	barrels	3
23	zeppelins	6	51	birches	4
24	motorcycles	5	52	flower pots	9
25	sport cars	6	53	trees	11
26	cars	23	54	weeds	9
27	Formula-1 cars	9	55	human bodies	56
28	galleons	4			


(a) Class 5 (fish-like)


(b) Class 13 (cups)


(c) Class 32 (chairs)

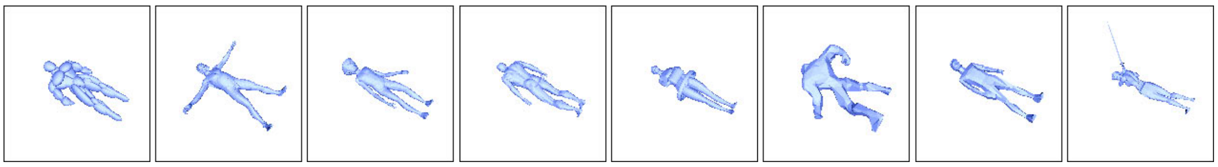
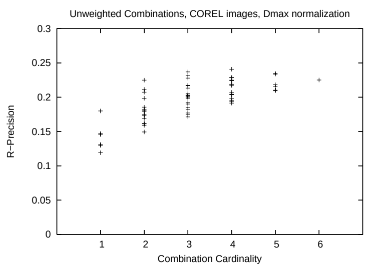
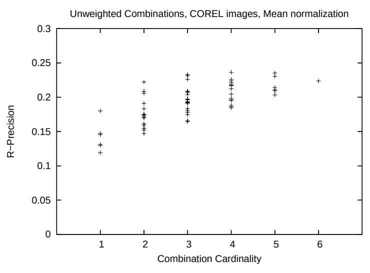

(d) Class 55 (human bodies)

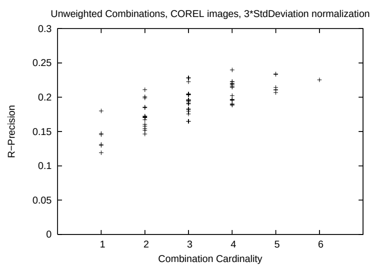
Figure A.1: Members of four different query classes from the Konstanz 3D Benchmark.



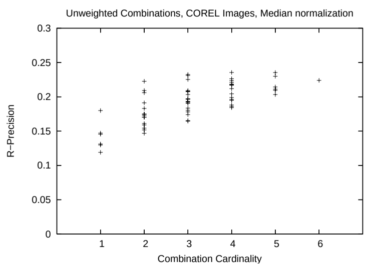
(a) d_{max} normalization



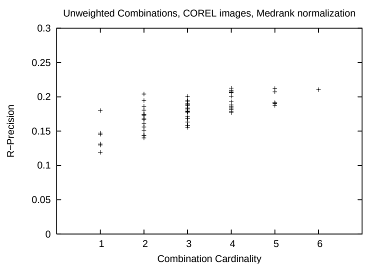
(b) mean normalization



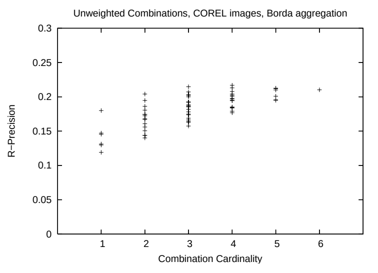
(c) variance normalization



(d) median normalization



(e) Medrank aggregation

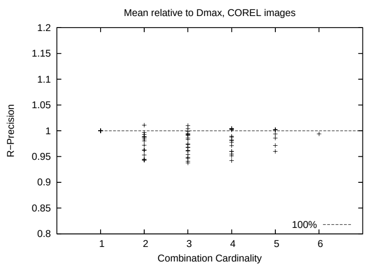


(f) Borda aggregation

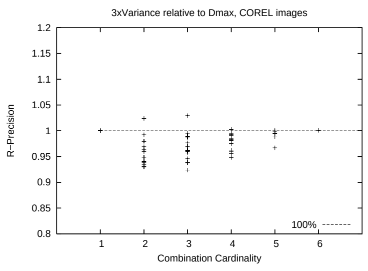
Figure A.2: Combination results for the COREL images database.

Table A.2: R-precision statistics for the COREL images combinations.

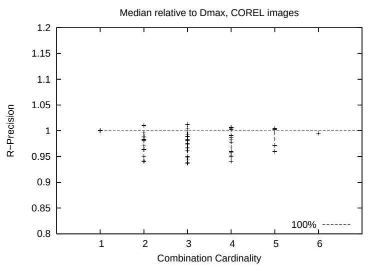
Normalization		1	2	3	4	5	6
d_{max}	min	11.9	14.9	17.1	19.1	20.9	22.5
d_{max}	mean	14.2	18.1	20.1	21.3	22.0	22.5
d_{max}	max	18.0	22.5	23.7	24.1	23.5	22.5
mean	min	11.9	14.7	16.5	18.5	20.3	22.4
mean	mean	14.2	17.7	19.6	20.8	21.7	22.4
mean	max	18.0	22.2	23.3	23.6	23.5	22.4
variance	min	11.9	14.6	16.5	18.9	20.7	22.5
variance	mean	14.2	17.4	19.5	20.9	21.8	22.5
variance	max	18.0	21.1	22.8	24.0	23.4	22.5
median	min	11.9	14.7	16.5	18.4	20.3	22.4
median	mean	14.2	17.7	19.6	20.8	21.7	22.4
median	max	18.0	22.3	23.3	23.5	23.5	22.4
Medrank	min	11.9	14.0	15.5	17.7	18.7	21.0
Medrank	mean	14.2	16.8	17.8	19.5	19.7	21.0
Medrank	max	18.0	20.4	20.1	21.3	21.2	21.0
Borda	min	11.9	14.0	15.7	17.7	19.5	21.0
Borda	mean	14.2	16.8	18.4	19.6	20.4	21.0
Borda	max	18.0	20.4	21.5	21.7	21.3	21.0



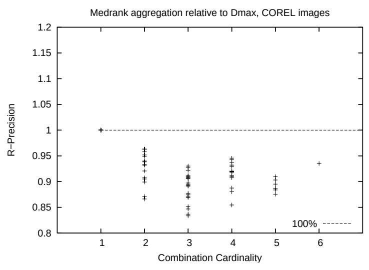
(a) Mean/d_{max}



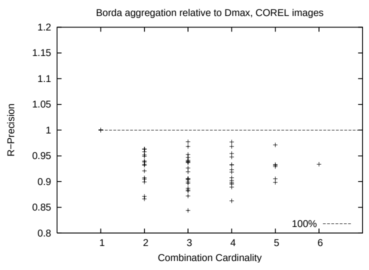
(b) 3xVariance/d_{max}



(c) Median/d_{max}



(d) Medrank/d_{max}



(e) Borda/d_{max}

Figure A.3: Performance of the aggregation methods relative to d_{max}, on the COREL FV database.

Size	COR	SD2	H3D	RIN	SHA	3DDFT	SIL	CPX	VOX	DBF	R-Precision
	(17.2)	(19.3)	(22.1)	(22.6)	(23.8)	(26.2)	(27.3)	(29.7)	(30.3)	(32.2)	
2	1	0	0	0	0	1	0	0	0	0	22.27%
2	1	1	0	0	0	0	0	0	0	0	22.71%
2	1	0	0	0	0	0	1	0	0	0	24.05%
2	1	0	0	1	0	0	0	0	0	0	25.11%
2	0	1	1	0	0	0	0	0	0	0	25.54%
2	1	0	0	0	1	0	0	0	0	0	25.73%
2	1	0	1	0	0	0	0	0	0	0	25.79%
2	0	1	0	1	0	0	0	0	0	0	26.15%
2	1	0	0	0	0	0	0	0	0	1	27.40%
2	0	0	1	0	0	1	0	0	0	0	27.48%
2	0	0	0	1	0	1	0	0	0	0	27.60%
2	0	1	0	0	0	1	0	0	0	0	27.96%
2	0	1	0	0	0	0	1	0	0	0	28.05%
2	1	0	0	0	0	0	0	1	0	0	28.17%
2	0	0	1	1	0	0	0	0	0	0	28.30%
2	0	0	1	0	0	0	1	0	0	0	28.39%
2	0	0	0	1	0	0	1	0	0	0	28.89%
2	0	0	1	0	1	0	0	0	0	0	29.67%
2	0	0	0	0	1	0	0	1	0	0	30.85%
2	0	0	0	1	0	0	0	0	0	1	30.88%
2	0	0	1	0	0	0	0	0	0	1	30.89%
2	0	0	0	1	1	0	0	0	0	0	30.90%
2	0	1	0	0	0	0	0	0	0	1	31.11%
2	0	0	0	0	0	1	0	0	1	0	31.17%
2	0	1	0	0	1	0	0	0	0	0	31.31%
2	0	0	1	0	0	0	0	1	0	0	31.40%
2	0	1	0	0	0	0	0	1	0	0	32.36%
2	0	0	0	0	1	1	0	0	0	0	32.41%
2	1	0	0	0	0	0	0	0	1	0	32.50%
2	0	0	0	0	1	0	1	0	0	0	33.54%
2	0	0	0	0	0	1	1	0	0	0	33.58%
2	0	0	0	1	0	0	0	1	0	0	33.62%
2	0	0	0	0	0	0	1	0	0	1	33.84%
2	0	1	0	0	0	0	0	0	1	0	34.23%
2	0	0	1	0	0	0	0	0	1	0	34.57%
2	0	0	0	1	0	0	0	0	1	0	34.58%
2	0	0	0	0	0	1	0	0	0	1	35.02%
2	0	0	0	0	0	0	1	1	0	0	35.32%
2	0	0	0	0	0	0	1	0	1	0	35.72%
2	0	0	0	0	1	0	0	0	0	1	35.97%
2	0	0	0	0	1	0	0	0	1	0	36.35%
2	0	0	0	0	0	1	0	1	0	0	36.65%
2	0	0	0	0	0	0	0	1	0	1	37.40%
2	0	0	0	0	0	0	0	0	1	1	37.50%
2	0	0	0	0	0	0	0	1	1	0	38.85%

Figure A.4: Participation histogram for combinations of cardinality 2 (KN-DB). The columns are sorted increasingly by single FV benchmark scores (R-precision), and the rows are sorted by increasing R-precision of the combinations. Dark shaded cells indicate that the respective FV participated in the combination. FVs which perform good in single usage participate more often in good performing combinations.

Size	COR	SD2	H3D	RIN	SHA	3DDFT	SIL	CPX	VOX	DBF	R-Precision
	(17.2)	(19.3)	(22.1)	(22.6)	(23.8)	(26.2)	(27.3)	(29.7)	(30.3)	(32.2)	
3	1	1	0	0	0	1	0	0	0	0	26.84%
3	1	1	0	0	0	0	1	0	0	0	27.34%
3	1	1	0	1	0	0	0	0	0	0	27.71%
3	1	0	0	0	0	1	1	0	0	0	28.08%
3	1	1	1	0	0	0	0	0	0	0	28.79%
3	1	0	0	1	0	1	0	0	0	0	28.84%
3	0	1	0	1	0	1	0	0	0	0	29.25%
3	1	0	0	1	0	0	1	0	0	0	29.75%
3	0	1	1	1	0	0	0	0	0	0	29.84%
3	1	0	1	0	0	1	0	0	0	0	29.95%
3	1	1	0	0	1	0	0	0	0	0	30.04%
3	0	1	0	1	0	0	1	0	0	0	30.04%
3	0	1	1	0	0	0	1	0	0	0	30.10%
3	1	0	0	0	1	1	0	0	0	0	30.10%
3	1	0	1	1	0	0	0	0	0	0	30.41%
3	1	0	1	0	0	0	1	0	0	0	30.47%
3	1	1	0	0	0	0	0	0	0	1	30.74%
3	0	1	1	0	0	1	0	0	0	0	30.79%
3	1	0	0	0	0	0	1	0	0	1	30.94%
3	1	0	0	1	0	0	0	0	0	1	31.11%
3	1	0	0	0	1	0	1	0	0	0	31.31%
3	1	0	0	1	1	0	0	0	0	0	31.36%
3	1	0	0	0	1	0	0	1	0	0	31.43%
3	1	0	1	0	1	0	0	0	0	0	31.44%
3	1	0	0	0	0	1	0	0	0	1	31.46%
3	0	0	1	1	0	1	0	0	0	0	31.65%
3	1	0	1	0	0	0	0	0	0	1	31.66%
3	1	1	0	0	0	0	0	1	0	0	31.87%
3	0	1	0	1	1	0	0	0	0	0	31.93%
3	0	0	0	1	0	1	1	0	0	0	31.94%
3	0	0	1	1	0	0	1	0	0	0	31.95%
3	0	1	1	0	0	0	0	0	0	1	32.10%
3	0	0	1	0	0	1	1	0	0	0	32.31%
3	0	1	0	1	0	0	0	0	0	1	32.38%
3	0	1	0	0	0	1	1	0	0	0	32.38%
3	0	1	1	0	1	0	0	0	0	0	32.40%
3	1	0	0	0	0	0	1	1	0	0	32.75%
3	1	0	1	0	0	0	0	1	0	0	32.81%
3	1	0	0	0	0	1	0	1	0	0	32.90%
3	0	0	1	1	0	0	0	0	0	1	33.40%
3	0	1	0	0	0	0	1	0	0	1	33.42%
3	0	0	0	1	0	1	0	0	0	1	33.43%
3	1	0	0	1	0	0	0	1	0	0	33.45%
3	1	0	0	0	1	0	0	0	0	1	33.47%
3	0	0	0	1	0	0	1	0	0	1	33.51%
3	0	0	1	0	1	0	0	1	0	0	33.56%
3	0	0	1	1	1	0	0	0	0	0	33.65%
3	0	1	0	1	0	0	0	1	0	0	33.66%
3	0	1	1	0	0	0	0	1	0	0	33.74%
3	0	0	1	0	1	1	0	0	0	0	33.87%
3	0	0	1	0	0	1	0	0	0	1	33.99%
3	0	0	1	0	1	0	1	0	0	0	34.03%
3	1	0	0	0	0	1	0	0	1	0	34.05%
3	0	0	1	0	0	0	1	0	0	1	34.08%
3	1	0	0	0	0	0	0	1	0	1	34.35%
3	0	0	0	1	1	1	0	0	0	0	34.61%
3	0	0	0	1	1	0	1	0	0	0	34.64%
3	0	1	0	0	1	0	0	1	0	0	34.86%
3	0	1	0	0	0	1	0	0	0	1	34.86%
3	0	1	0	0	1	0	1	0	0	0	34.91%

(a)

Size	COR	SD2	H3D	RIN	SHA	3DDFT	SIL	CPX	VOX	DBF	R-Precision
	(17.2)	(19.3)	(22.1)	(22.6)	(23.8)	(26.2)	(27.3)	(29.7)	(30.3)	(32.2)	
3	1	1	0	0	0	0	0	0	1	0	34.99%
3	0	0	1	0	0	0	1	1	0	0	35.02%
3	0	1	0	1	0	0	0	0	1	0	35.11%
3	0	0	0	1	1	0	0	1	0	0	35.18%
3	1	0	0	1	0	0	0	0	1	0	35.21%
3	0	0	1	1	0	0	0	1	0	0	35.23%
3	0	1	0	0	0	0	1	1	0	0	35.32%
3	0	0	1	0	0	1	0	1	0	0	35.38%
3	0	0	0	1	1	0	0	0	0	1	35.43%
3	0	0	0	0	1	0	1	1	0	0	35.61%
3	0	1	1	0	0	0	0	0	1	0	35.78%
3	0	1	0	0	0	1	0	0	1	0	35.79%
3	0	0	1	0	1	0	0	0	0	1	35.80%
3	0	0	0	1	0	1	0	0	1	0	35.90%
3	1	0	0	0	0	0	1	0	1	0	35.92%
3	1	0	1	0	0	0	0	0	1	0	35.97%
3	0	0	1	0	0	1	0	0	1	0	36.00%
3	0	0	0	1	0	0	1	1	0	0	36.13%
3	0	0	0	0	1	1	0	1	0	0	36.23%
3	0	0	0	0	0	1	1	0	0	1	36.33%
3	0	0	0	0	0	1	1	0	1	0	36.41%
3	0	0	1	1	0	0	0	0	1	0	36.42%
3	0	1	0	0	1	0	0	0	0	1	36.45%
3	0	1	0	0	0	0	1	0	1	0	36.51%
3	0	1	0	0	1	1	0	0	0	0	36.57%
3	0	0	0	1	0	0	1	0	1	0	36.76%
3	0	0	1	0	0	0	0	1	0	1	36.77%
3	0	0	0	1	0	1	0	1	0	0	37.01%
3	0	1	0	0	0	0	0	1	0	1	37.09%
3	0	0	0	0	1	1	0	0	1	0	37.13%
3	0	1	0	0	0	1	0	1	0	0	37.14%
3	0	0	0	1	0	0	0	1	0	1	37.14%
3	0	0	0	0	1	0	0	1	0	1	37.26%
3	0	0	1	0	0	0	1	0	1	0	37.36%
3	0	0	0	0	1	0	1	0	0	1	37.53%
3	1	0	0	0	1	0	0	0	1	0	37.67%
3	1	0	0	0	0	0	0	0	1	1	37.72%
3	0	0	0	1	0	0	0	0	1	1	37.80%
3	0	0	0	0	0	1	0	0	1	1	38.12%
3	0	0	0	0	1	1	1	0	0	0	38.23%
3	0	0	1	0	0	0	0	0	1	1	38.30%
3	0	1	0	0	0	0	0	0	1	1	38.34%
3	0	0	0	0	0	0	1	1	0	1	38.34%
3	0	0	0	0	0	1	1	1	0	0	38.80%
3	0	0	0	1	1	0	0	0	1	0	39.04%
3	0	0	0	0	0	0	1	0	1	1	39.05%
3	0	0	0	0	1	1	0	0	0	1	39.21%
3	1	0	0	0	0	0	0	1	1	0	39.23%
3	0	0	0	0	1	0	0	1	1	0	39.39%
3	0	0	0	0	0	1	0	1	1	0	39.46%
3	0	1	0	0	1	0	0	0	1	0	39.57%
3	0	0	1	0	0	0	0	1	1	0	39.58%
3	0	0	1	0	1	0	0	0	1	0	39.71%
3	0	0	0	0	1	0	1	0	1	0	39.76%
3	0	0	0	0	0	1	0	1	0	1	40.00%
3	0	0	0	0	0	0	1	1	1	0	40.38%
3	0	0	0	1	0	0	0	1	1	0	40.48%
3	0	1	0	0	0	0	0	1	1	0	40.53%
3	0	0	0	0	1	0	0	0	1	1	41.59%
3	0	0	0	0	0	0	0	1	1	1	41.76%

(b)

Figure A.5: Participation histogram for combinations of cardinality 3 (KN-DB).

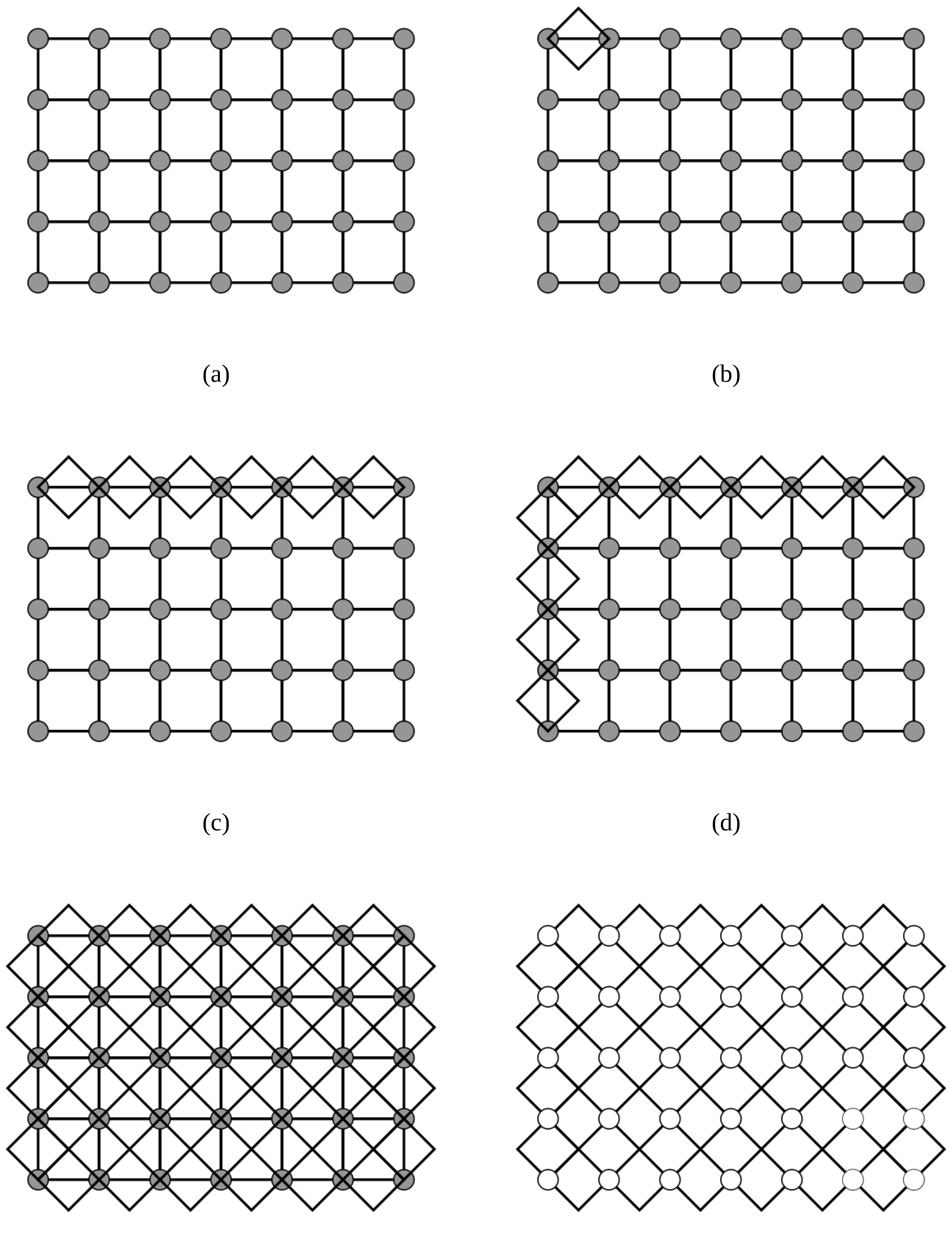
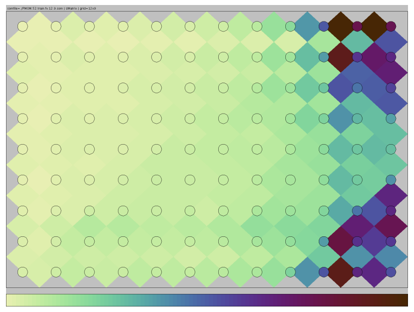
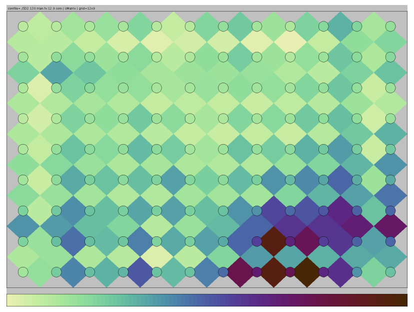
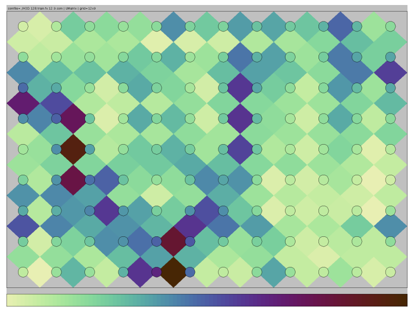
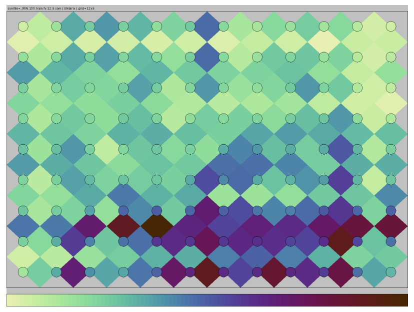


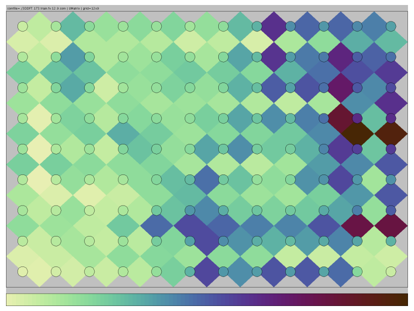
Figure A.6: The diamond-shaped U-Matrix is constructed by representing the distances between adjacent nodes on the rectangular grid by diamonds (we thank Dietmar Saupe for giving the idea). The diamonds are color-coded by respective distance. The SOM nodes are represented by circles, with the circles being color-coded according to the average distance to all neighboring nodes.

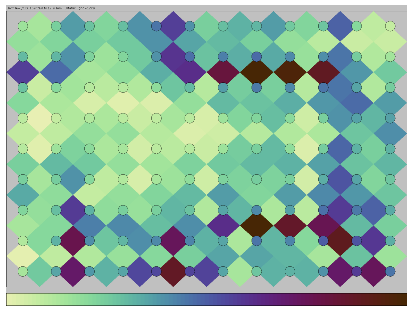

(a) PMOM (15%, 1.16)

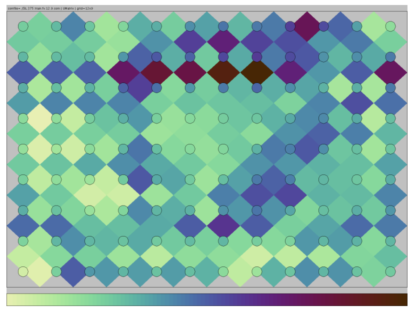

(b) SD2 (18%, 0.97)

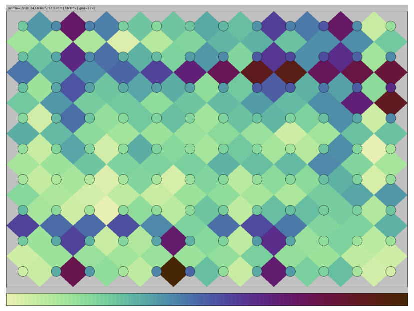

(c) H3D (20%, 0.81)

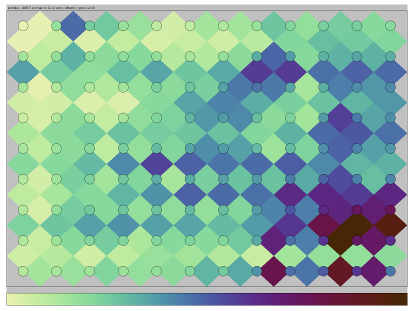

(d) RIN (23%, 0.70)


(e) 3DDFT (25%, 0.78)


(f) CPX (27%, 0.75)


(g) SIL (28%, 0.83)


(h) VOX (31%, 0.73)


(i) DBF (31%, 0.72)

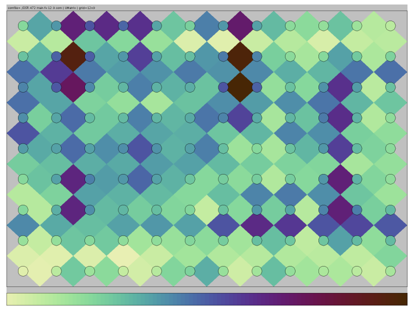
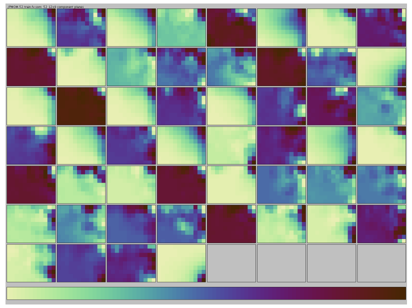
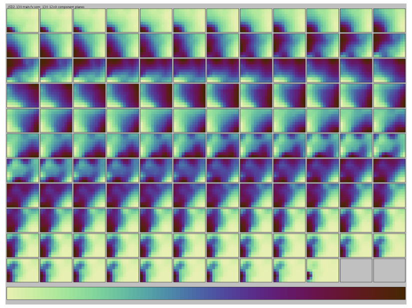
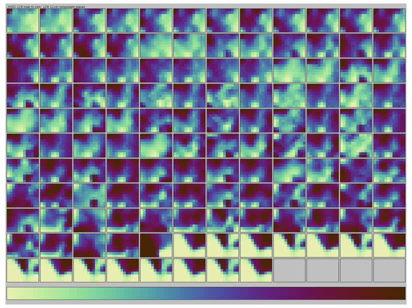
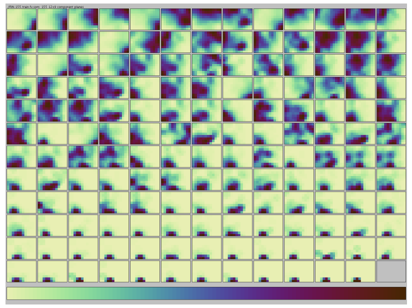

(j) DSR (43%, 0.73)

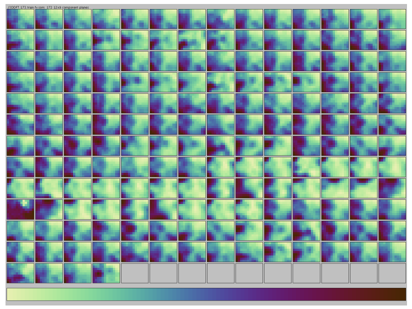
Figure A.7: U-Matrices for the ten FV spaces from Section 3.2.4. R-precision and uniformity scores are given in brackets.

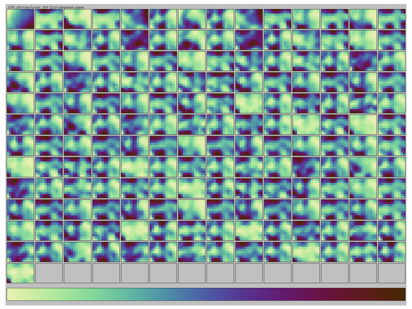

(a) PMOM (15%, 52d)

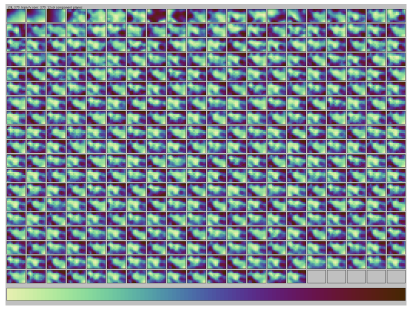

(b) SD2 (18%, 130d)

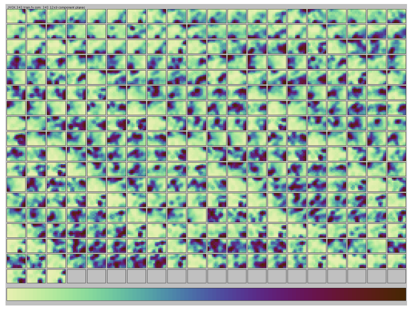

(c) H3D (20%, 128d)

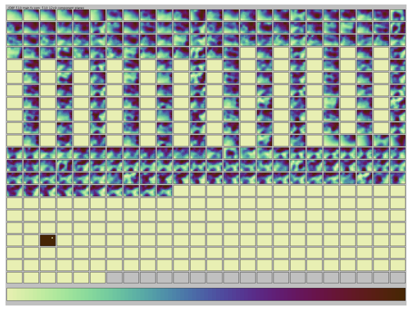

(d) RIN (23%, 155d)


(e) 3DDFT (25%, 173d)


(f) CPX (27%, 169d)


(g) SIL (28%, 375d)


(h) VOX (31%, 343d)


(i) DBF (31%, 510d)

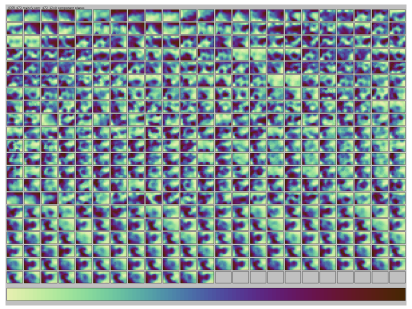

(j) DSR (43%, 472d)

Figure A.8: Component plane arrays for the ten FV spaces from Section 3.2.4. R-precision and FV dimensionality are given in brackets.

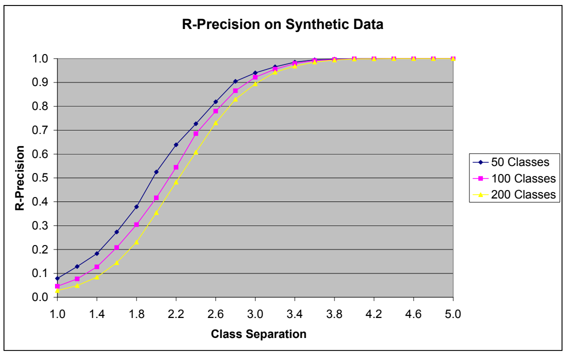
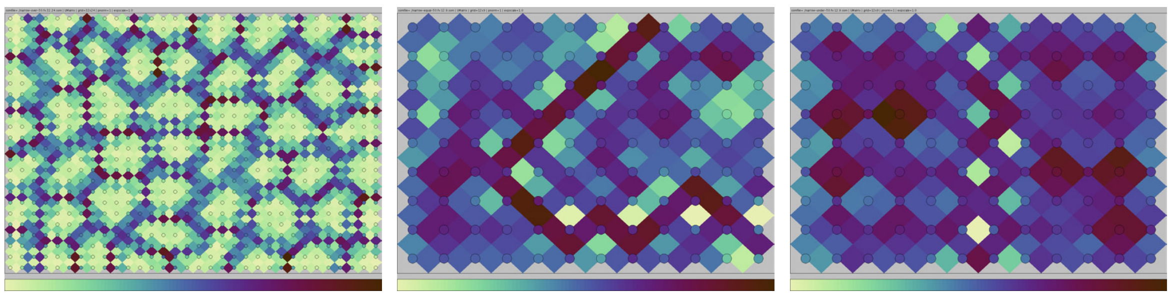


Figure A.9: R-precision plots for the synthetic data sets used in Section 3.2.5. The R-precision ranges between 0% and 100%, modeling R-precision score ranges encountered also in real benchmarks.



(a) Over-sampling (b) Equal-sampling (c) Under-sampling

Figure A.10: U-Matrices for the synthetic datasets used for experiments in Section 3.2.5 with d_span settings of 5, providing good cluster separation properties (cf. also Figure A.9).

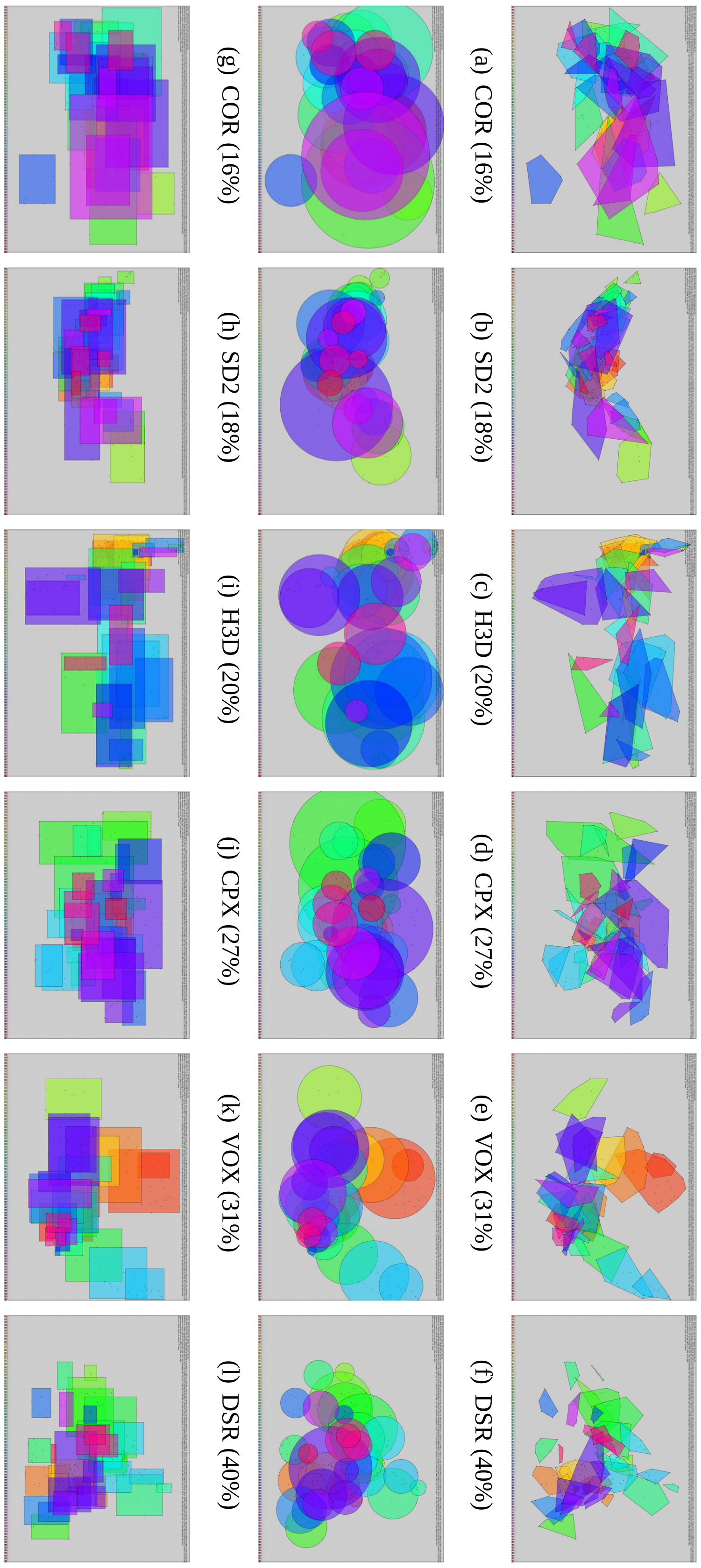


Figure A.11: Convex hulls (top row), and minimum bounding discs (middle row) and rectangles (bottom row) over the FV spaces discussed in Section 3.3.4.

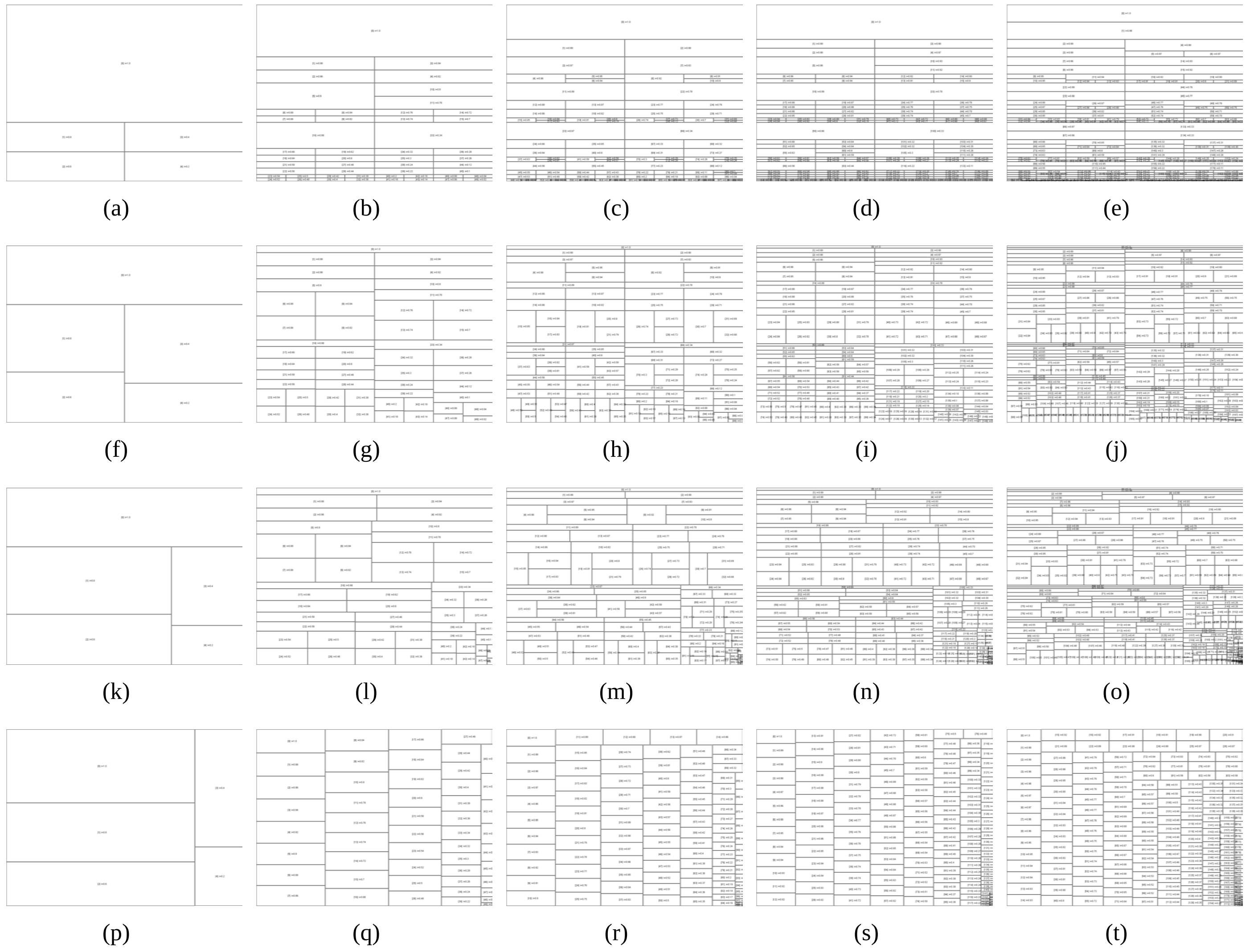
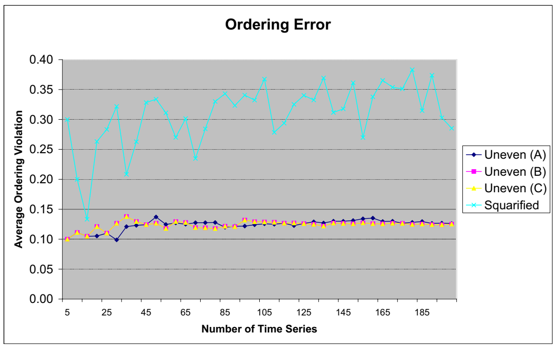
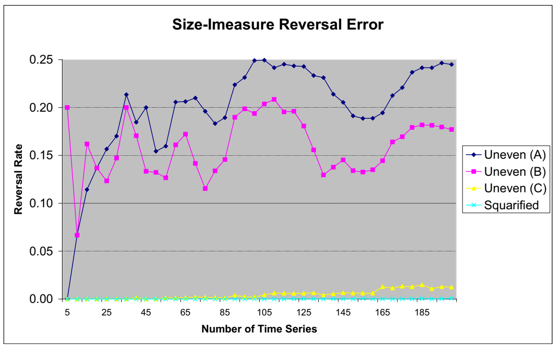


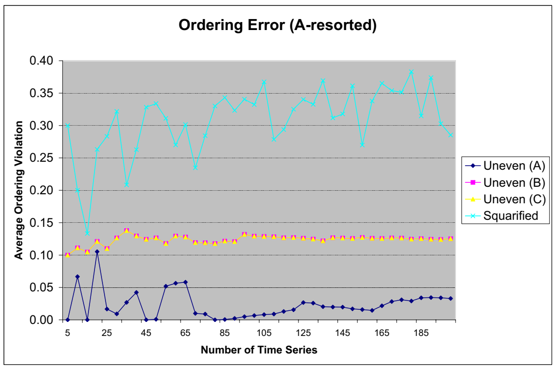
Figure A.12: Snapshots of layouts using ID-Map. From top to bottom rows, the uneven splitting mask using splitting policies *A*, *B*, and *C*, as well as the Squarified TreeMap algorithm are shown for layout of 5 up to 200 rectangles in a 1200×900 display (refer to Section 4.2.6).



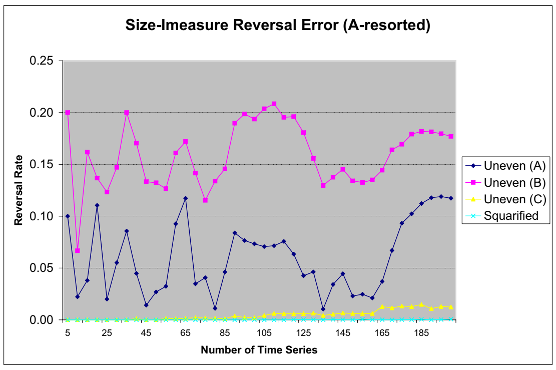
(a) Ordering error.



(b) Reversal error.

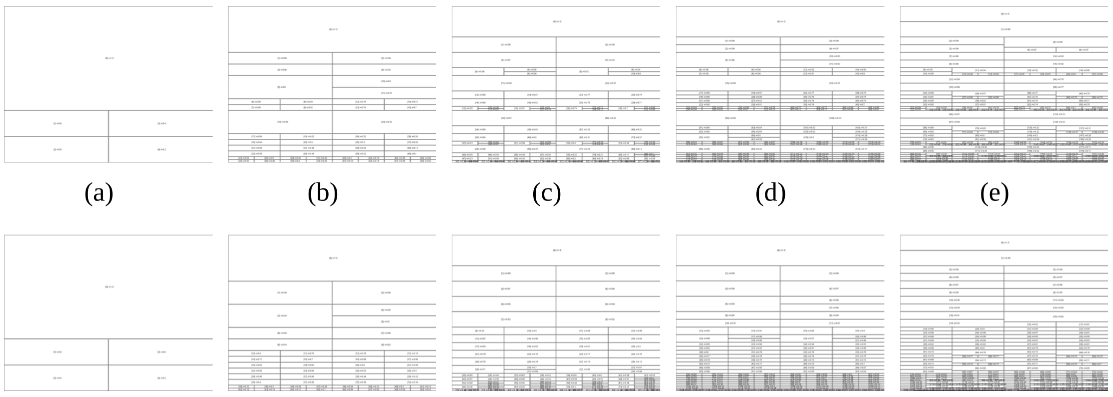


(c) Ordering error (resorting).



(d) Reversal error (resorting).

Figure A.13: Position and size errors without (first row) and with resorting of rectangles (second row). Resorting reduces the amount of size/imeasure reversals to about 5% on average, and also improves the (positional) ordering error.



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

(f) (g) (h) (i) (j)

Figure A.14: The uneven layout snapshots from Figure A.12 (first row), and the same layouts using resorting (second row).

Bibliography

[1] J. Abello, H. Schumann, and C. Tominski. Axes-based visualizations for time series dat. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*. IEEE, 2003.

[2] C. Aggarwal. On the effects of dimensionality reduction on high dimensional similarity search. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 256–266, New York, NY, USA, 2001. ACM Press.

[3] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 420–434, 2001.

[4] A. Aris, P. Buono, A. Khella, C. Plaisant, and B. Shneiderman. Interactive pattern search in time series. In *Proceedings of the SPIE Conference on Visualization and Data Analysis*, 2005.

[5] B. Shneiderman's TreeMap History Website. `http://www.cs.umd.edu/hcil/treemap-history/index.shtml/`.

[6] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[7] M. Balzer and O. Deussen. Voronoi treemaps. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis), Minneapolis, MN, USA, October 23-25*. IEEE Computer Society, 2005.

[8] B. Bederson. Photomesa: A zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'01)*, pages 71–80, 2001.

[9] B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Trans. Graph.*, 21(4):833–854, 2002.

[10] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

[11] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, 2000.

[12] B. Bustos. *Index Structures For Similarity Search in Multimedia Databases*. PhD thesis, University of Konstanz, Germany, 2006. submitted.

[13] B. Bustos, D. Keim, C. Panse, and T. Schreck. 2D maps for visual analysis and retrieval in large multi-feature 3D model databases (poster paper). In *Proceedings of the IEEE Visualization Conference (VIS'2004)*. IEEE Press, 2004. Poster paper.

[14] B. Bustos, D. Keim, D. Saupe, T. Schreck, and A. Tatu. Methods and user interfaces for effective retrieval in 3D databases (in German). *Datenbank-Spektrum - Zeitschrift für Datenbank Technologie und Information Retrieval*, (20):23–32, 2007.

[15] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Automatic selection and combination of descriptors for effective 3D similarity search. In *Proceedings of the IEEE International Workshop on Multimedia Content-based Analysis and Retrieval*, pages 514–521. IEEE Computer Society, 2004.

[16] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. An experimental comparison of feature-based 3D retrieval methods. In *Second International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'2004). Thessaloniki, Greece, September 6-9.*, pages 215–222. IEEE, 2004. Poster paper.

[17] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Using entropy impurity for improved 3D object similarity search. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), Taipei, Taiwan, June 27-30*, pages 1303–1306. IEEE, 2004.

[18] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Feature-based similarity search in 3D object databases. *ACM Computing Surveys (CSUR)*, 37:345–387, 2005.

[19] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranic. An experimental effectiveness comparison of methods for 3D similarity search. *International Journal on Digital Libraries, Special Issue on Multimedia Contents and Management*, 6(1):39–54, 2006.

[20] B. Bustos, D. Keim, and T. Schreck. A pivot-based index structure for combination of feature vectors. In *Proceedings of the 20th Annual ACM Symposium on Applied Computing. Multimedia and Visualization Track, Santa Fe, New Mexico, March 13 -17, 2005*. ACM, 2005.

[21] R. Campbell and P. Flynn. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding*, 81(2):166–210, 2001.

[22] S. Card, J. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kauffman, 1999.

[23] S. Card and R. Rao. Exploring large tables with the table lens. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. ACM, 1995.

[24] M. Cox and M. Cox. *Multidimensional Scaling*. Chapman and Hall, 2001.

[25] C. Blake D. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.

[26] U. Dayal, M. Hao, D. Keim, and T. Schreck. Importance driven visualization layouts for large time-series data. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis), Minneapolis, MN, USA, October 23-25*. IEEE Computer Society, 2005.

[27] DELOS Network of Excellence on Digital Libraries, funded under the EU Sixth Framework Programme. `http://www.delos.info/`.

[28] I. S. Dhillon, D. S. Modha, and W. S. Spangler. Class visualization of high-dimensional data with applications. *Computational Statistics and Data Analysis*, 4(1):59–90, 2002. Special issue on Matrix Computations and Statistics.

[29] H. Dolfing. A discretized projection-based framework for visual analytics. Master's thesis, University of Konstanz, Germany, 2007.

[30] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley-Interscience, New York, 2nd edition, 2001.

[31] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proc. ACM WWW Conference*, 2001.

[32] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proc. ACM SIGMOD Conference*, pages 301–312, 2003.

[33] C. Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[34] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3D models. *ACM Transactions on Graphics*, 22(1):83–105, 2003.

[35] G. Furnas. Generalized fisheye views. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. ACM, 1986.

[36] R. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Proc. Letters*, 1(4):132–133, 1972.

[37] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kauffman, 2001.

[38] P. Hanrahan, C. Stolte, and D. Tang. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Transactions on Visualization and Computer Graphics (TVCG)*, 8(1), 2002.

[39] D. Heesch, A. Yavlinsky, and S. Rueger. Performance comparison between different similarity models for cbir with relevance feedback. In *CIVR '03: Proceedings of the International Conference on Image and Video Retrieval*, pages 456–466. Springer-Verlag, 2003.

[40] R. Heilmann, D. A. Keim, C. Panse, and M. Sips. Recmap: Rectangular map approximations. In *IEEE Symposium on Information Visualization 2004 (InfoVis04), October 10-12, Austin, Texas, USA*. IEEE Computer Society, October 2004.

[41] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proc. ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'01)*, pages 203–212. ACM Press, 2001.

[42] A. Hinneburg, C. Aggarwal, and D. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 506–515, 2000.

[43] T. Honkela, S. Kaski, K. Lagus, and T. Kohonen. WEBSOM—self-organizing maps of document collections. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6*, pages 310–315. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[44] P. Howarth and S. Rueger. Evaluation of texture features for content-based image retrieval. In *CIVR '04: Proceedings of the International Conference on Image and Video Retrieval*.

[45] D. Keim J. Schneidewind, M. Sips. Pixnostics: Towards measuring the value of visualization. In *IEEE Symposium on Visual Analytics and Technology (VAST 2006), Baltimore, Maryland, USA, October 29 - November 3*, 2006.

[46] I. Jolliffe. *Principal Components Analysis*. Springer, 3rd edition, 2002.

[47] T. Kato, M. Suzuki, and N. Otsu. A similarity retrieval of 3D polygonal models using rotation invariant shape descriptors. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, pages 2946–2952, 2000.

[48] L. Kaufman and P. Rousseeuw. *Finding groups in data*. Wiley, New York, 1990.

[49] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proc. Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP'03)*, pages 156–164. Eurographics Association, 2003.

[50] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Shape matching and anisotropy. *ACM Transactions on Graphics*, 23(3):623–629, August 2004.

[51] D. Keim. Efficient geometry-based similarity search of 3D spatial databases. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 419–430, New York, NY, USA, 1999. ACM Press.

[52] D. Keim, F. Mansmann, and T. Schreck. Analyzing electronic mail using temporal, spatial, and content-based visualization techniques. In *LNI Tagungsbände Informatik*. Gesellschaft für Informatik (GI), 2005. Invited paper.

[53] D. Keim, F. Mansmann, and T. Schreck. Mailsom - visual exploration of electronic mail archives using Self-Organizing Maps. In *Second Conference on Email and Anti-Spam (CEAS 2005), Stanford University, Palo Alto, CA, USA, July 21-22*, 2005. Short paper.

[54] D. Keim, T. Nietzschmann, N. Schelwies, J. Schneidewind, T. Schreck, and H. Ziegler. A spectral visualization system for analyzing financial time series data. In *Proceedings of the EuroVis 2006: Eurographics/IEEE-VGTC Symposium on Visualization, Lisbon, Portugal, May 8-10 , 2006*. IEEE Computer Society, 2006.

[55] Daniel A. Keim, Mihael Ankerst, and Hans-Peter Kriegel. Recursive pattern: A technique for visualizing very large amounts of data. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 279, Washington, DC, USA, 1995. IEEE Computer Society.

[56] E. Keogh. Tutorial on data mining and machine learning in time series databases. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM)*. IEEE, 2004.

[57] E. Keogh and T. Folias. The UCR time series data mining archive, 2002.

[58] H. Kestler, A. Mueller, T. Gress, and M. Buchholz. Generalized venn diagrams: A new method of visualizing complex genetic set relations. *Bioinformatics*, 21(8):1592–1595, 2005.

[59] T. Kohonen. *Self-Organizing Maps*. Springer, Berlin, 3rd edition, 2001.

[60] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. Som_pak: The self-organizing map program package. Technical Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science, FIN-02150 Espoo, Finland, 1996.

[61] Konstanz 3D Model Database. `http://merkur01.inf.uni-konstanz.de/CCCC/`.

[62] Y. Koren and L. Carmel. Visualization of Labeled Data Using Linear Transformation. In *IEEE Symposium on Information Visualization (InfoVis)*, pages 121–128, 2003.

[63] J. Laaksonen, M. Koskela, S. Laakso, and E. Oja. PicSOM—content-based image retrieval with self-organizing maps. *Pattern Recogn. Lett.*, 21(13-14):1199–1207, 2000.

[64] S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8):983–1001, 1998.

[65] H. Müller, S. Marchand-Maillet, and T. Pun. The truth about corel - evaluation in image retrieval. In *CIVR '02: Proceedings of the International Conference on Image and Video Retrieval*, pages 38–49, London, UK, 2002. Springer-Verlag.

[66] M. Novotni and R. Klein. A geometric approach to 3D object comparison. In *Proc. International Conference on Shape Modeling and Applications*, pages 167–175. IEEE CS Press, 2001.

[67] M. Novotni and R. Klein. Shape retrieval using 3D zernike descriptors. *Computer Aided Design*, 36(11):1047–1062, 2004.

[68] US National Institute of Standards and technology. Text retrieval conference, `http://-trec.nist.gov/`.

[69] R. Ohbuchi and Y. Hata. Combining multiresolution shape descriptors for 3D model retrieval. In *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'2006)*, 2006.

[70] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape distributions. *ACM Transactions on Graphics*, 21(4):807–832, 2002.

[71] E. Pampalk. Islands of music. Master's thesis, Technical University of Vienna, 2001.

[72] E. Pampalk, A. Rauber, and D. Merkl. Using smoothed data histograms for cluster visualization in self-organizing maps. In *Proc. Int. Conf. on Artifical Neural Networks (ICANN'02)*, volume 2415 of *Lecture Notes in Computer Science*. Springer, 2002.

[73] E. Paquet, M. Murching, T. Naveen, A. Tabatabai, and M. Rioux. Description of shape information for 2-D and 3-D objects. *Signal Processing: Image Communication*, 16:103–122, 2000.

[74] Pebbles Circular TreeMap Project. `http://lip.sourceforge.net/ctreemap.html`.

[75] M. Pickering and S. Rüger. Evaluation of key frame-based retrieval techniques for video. *Computer Vision and Image Understanding*, 92:217–235, 2003.

[76] PicSOM project homepage. `http://www.cis.hut.fi/picsom/`.

[77] T. Schreck, D. Keim, and F. Mansmann. Regular treemap layouts for visual analysis of hierarchical data. In *Spring Conference on Computer Graphics (SCCG'2006), April 20-22, Casta Papiernicka, Slovak Republic*. ACM Siggraph, 2006.

[78] T. Schreck, D. Keim, and C. Panse. Visual feature space analysis for unsupervised effectiveness estimation and feature engineering. In *IEEE International Conference on Multimedia and Expo (ICME'2006). Toronto, Canada, July 9-12*, 2006.

[79] T. Schreck and C. Panse. A new metaphor for projection-based visual analysis and data exploration. In *Proceedings of the IS&T/SPIE Conference on Visualization and Data Analysis (VDA)*, 2007.

[80] SequoiaView Homepage. `http://www.win.tue.nl/sequoiaview/`.

[81] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The princeton shape benchmark. In *Proc. International Conference on Shape Modeling and Applications (SMI'04)*, pages 167–178. IEEE CS Press, 2004.

[82] B. Shneiderman. Tree visualization with tree-maps: 2D space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992.

[83] B. Shneiderman, editor. *Sparks of Innovation in Human-Computer Interaction*. Ablex Publishing Corporation, 1993.

[84] S&P500 stock price archive. `http://kumo.swcp.com/stocks/`.

[85] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *Proc. International Conference on Shape Modeling and Applications (SMI'03)*, pages 130–142. IEEE CS Press, 2003.

[86] J. Tangelder and R. Veltkamp. Polyhedral model retrieval using weighted point sets. *International Journal of Image and Graphics*, 3(1):209–229, 2003.

[87] J. Tangelder and R. Veltkamp. A survey of content based 3D shape retrieval methods. In *Proc. International Conference on Shape Modeling and Applications (SMI'04)*, pages 145–156. IEEE CS Press, 2004.

[88] H. van de Wetering and J. van Wijk. Cushion treemaps: Visualization of hierarchical information. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, 2005.

[89] C. van Rijsbergen. *Information retrieval*. Butterworths, London, 2nd edition, 1979.

[90] J. van Wijk. Personal communication, January 2006.

[91] J. van Wijk and A. Telea. Enridged contour maps. In *Proceedings of the IEEE Visualization Conference*, 2001.

[92] J. van Wijk and H. van de Wetering. Cushion treemaps. In *Proceedings IEEE Symposium on Information Visualization (InfoVis)*, pages 73–78, 1999.

[93] R. Veltkamp and M. Tanase. Content-based image retrieval systems: A survey. Technical Report UU-CS-2000-34, University Utrecht, 2000.

[94] F. Vernier and L. Nigay. Modifiable treemaps containing variable shaped units. In *Work in Progress Proceedings of the IEEE Information Visualization (InfoVis)*, 2000.

[95] J. Vesanto. SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126, 1999.

[96] D. Vranić. An improvement of rotation invariant 3D-shape descriptor based on functions on concentric spheres. In *Proc. IEEE International Conference on Image Processing (ICIP'03), Volume III*, pages 757–760, 2003.

[97] D. Vranić. *3D Model Retrieval*. PhD thesis, University of Leipzig, 2004.

[98] D. Vranić, D. Saupe, and J. Richter. Tools for 3D-object retrieval: Karhunen-Loeve transform and spherical harmonics. In *Proc. IEEE 4th Workshop on Multimedia Signal Processing*, pages 293–298, 2001.

[99] C. Ware. *Information Visualization*. Morgan Kaufmann, 2004.

[100] M. Wattenberg. Visualizing the stock market. In *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, pages 188–189, New York, NY, USA, 1999. ACM Press.

[101] M. Wattenberg. A note on space-filling visualizations and space-filling curves. In *IEEE Int. Symposium on Information Visualization (InfoVis)*, 2005.

[102] E. Weisstein. Pearson mode skewness. From MathWorld – A Wolfram Web Resource `http://mathworld.wolfram.com/PearsonModeSkewness.html`.

[103] J. Van Wijk and E. Van Selow. Cluster and calendar based visualization of time series data. In *INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization*, page 4, Washington, DC, USA, 1999. IEEE Computer Society.

[104] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of The Twentieth International Conference on Machine Leaning (ICML-03)*, pages 856–863, 2003.

[105] T. Zaharia and F. Prêteux. 3D shape-based retrieval within the MPEG-7 framework. In *Proc. SPIE Conference on Nonlinear Image Processing and Pattern Analysis XII*, volume 4304, pages 133–145, 2001.