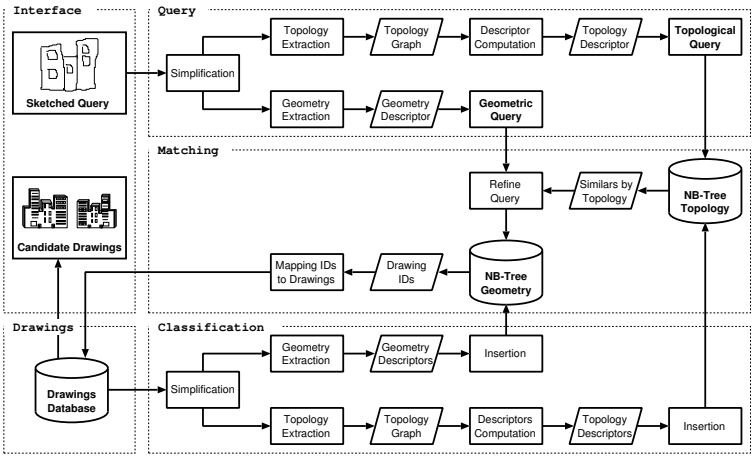
UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO



Sketch-Based Retrieval in Large Sets of Drawings

Manuel João Caneira Monteiro da Fonseca
(Master)

Dissertation for the degree of Doctor of Philosophy in Information Systems and Computer Engineering

Adviser: Doutor Joaquim Armando Pires Jorge

Chairman: Reitor da Universidade Técnica de Lisboa

Members: Doutor João Carlos Rogenmoser Lourenço Fernandes
Doutor Nuno Manuel Carvalho Ferreira Guimarães
Doutor João Afonso Ramalho Sopas Pereira Bento
Doutor Mário Rui Fonseca dos Santos Gomes
Doutor Nuno Manuel Robalo Correia
Doutor Joaquim Armando Pires Jorge

July 2004

Sketch-Based Retrieval in Large Sets of Drawings

Manuel João Caneira Monteiro da Fonseca

A Thesis submitted to the Graduate School
for the degree of

Doctor of Philosophy in
Information Systems and Computer Engineering

Department of

Information Systems and Computer Engineering

Instituto Superior Técnico

Adviser

Prof. Doutor Joaquim Armando Pires Jorge

Professor Auxiliar com Agregação from the
Department of Information Systems and Computer Engineering
Instituto Superior Técnico
Technical University of Lisbon

This work was funded in part by the Portuguese Foundation for Science and Technology, project 34672/99 and the European Commission, project SmartSketches IST-2000-28169.

http://immi.inesc-id.pt/∼mjf

mjf@inesc-id.pt, mjf@ist.utl.pt

Abstract

Thanks to the proliferation of drafting packages, there are a lot of vector drawings available for people to integrate into documents. These come in a variety of formats, such as Corel, Postscript, CGM, WMF, DXF and recently SVG. Moreover, creative designers and draftspeople often re-use data from previous projects, publications and libraries of ready to use components. Usually, retrieving these drawings is a slow, complex and error-prone endeavor, requiring either exhaustive visual examination, a solid memory, or both. While text-driven attempts at classifying image data have been recently supplemented with query-by-image content, these have been developed for bitmap-type data and cannot handle vectorial information.

In this thesis we present a new approach to allow indexing and retrieving vector drawings by content, using information automatically extracted from figures. Our work uses topological relationships, visual elements and high-dimensional indexing to retrieve complex figures from large databases.

Drawings are indexed based on topological relationships among objects. In particular, we produce a multilevel description of drawings using different levels of detail and subparts of drawings. To deal with the large set of descriptors, arising from the classification phase, we developed a new multidimensional indexing structure, the NB-Tree, which outperforms well-known indexing methods and scales well with growing data set size and dimension. Filtering by topology avoids an exhaustive search and comparison through the entire drawing database, thus speeding up retrieval.

Our approach was tested on two different application domains as instances of the general problem of retrieving vector drawings. Experimental evaluation with users showed that our approach presents good results for retrieving CAD drawings, where there is a lot of topological relationships among elements, and clip-art drawings, where geometry provides the most discriminating information.

Keywords

Content-Based Retrieval

Sketch-Based Retrieval

Multilevel Description

High-Dimensional Indexing

Graph Spectrum

Visual Features Extraction

Resumo

Devido à enorme proliferação de programas de desenho, existe uma grande quantidade de figuras vectoriais disponíveis para as pessoas integrarem nos seus documentos. Estes vêm em formatos tão variados, como Corel, Postscript, CGM, WMF, DXF e mais recentemente em SVG. Além disso, os *designers* e os desenhadores reutilizam dados de projectos antigos, de publicações e de bibliotecas de componentes prontos a usar. Normalmente, recuperar estes desenhos é uma tarefa lenta, complexa e sujeita a erros, exigindo um exame visual exaustivo, uma boa memória ou ambos. Embora as abordagens usando descrições textuais tenham sido recentemente suplantadas pela pesquisa baseada no conteúdo das imagens, estas foram desenvolvidas para imagens digitais não conseguindo lidar com informação vectorial.

Nesta tese apresentamos uma nova abordagem que permite a indexação e a recuperação de desenhos vectoriais com base no seu conteúdo, usando informação automaticamente extraída a partir destes. O nosso trabalho usa relações topológicas, elementos visuais e indexação multidimensional para recuperar desenhos complexos guardados em bases de dados grandes.

Os desenhos são indexados com base nas relações topológicas existentes entre os objectos. Em particular, produzimos uma descrição multinível dos desenhos, usando diferentes níveis de detalhe e subpartes do desenho. Para lidar com o enorme conjunto de descritores resultante da fase de classificação, desenvolvemos uma estrutura de indexação multidimensional, a NB-Tree, que apresenta melhores desempenhos que outras estruturas bem conhecidas e escala bem com o tamanho dos conjuntos de dados, assim como com

a dimensão dos descritores. A filtragem por topologia evita pesquisas e comparações exaustivas em toda a base de dados, tornando assim a recuperação mais rápida.

A nossa abordagem foi testada em dois domínios de aplicação diferentes, como instâncias do problema mais geral de recuperar desenhos vectoriais. A avaliação experimental com utilizadores mostrou que a nossa abordagem apresenta bons resultados quer na recuperação de desenhos técnicos, onde existem muitas relações topológicas entre os elementos, quer na recuperação de desenhos *clip-art*, onde a geometria fornece a informação mais discriminante.

Palavras Chave

Recuperação Baseada no Conteúdo

Recuperação Usando Esboços

Descrição Multinível

Indexação de Dados de Elevada Dimensão

Espectro de Grafos

Extracção de Características Visuais

Acknowledgements

First, I would like to thank my adviser, Prof. Joaquim Jorge, for his continuous support during my PhD. He was always there to meet and talk about my ideas and to provide me with good guidance. His constant comments and the confidence he showed to me and to my research contributed to improve the quality of the developed work.

I wish to thank the members of my thesis committee, Professors Lourenço Fernandes, João Bento, Mário Rui Gomes, Nuno Guimarães and Nuno Correia for valuable contributions and comments to this work.

I would like to thank Eng. Alfredo Ferreira Jr. who implemented the Sketch-Based Retrieval prototype and Engs. Bruno Barroso and Pedro Ribeiro who coded the BajaVista prototype. I would also like to thank all the members of the IMMI (Intelligent MultiModal Interfaces) group for providing me with a good working environment.

Last, I would like to thank the institutions that help me concluding this work, the INESC-ID Lisboa, where I do my research and the IST, for dispensing me from teaching during 3 years to work on my PhD.

I dedicate this thesis to my wife Maria João Belo and my son Rui Pedro who shared all the highs and lows throughout the whole endeavor, and with whom I did not share all the time they deserve.

Lisboa, July 2004

Manuel João Caneira Monteiro da Fonseca

Contents

List of Figures

List of Tables

List of Original Publications

The work described in this dissertation has been the subject of over 15 original publications presented to peer-reviewed journals and scientific meetings. From these publications we selected the more relevant, which in some cases complement the description presented in this dissertation. Publications are organized by publishing venue.

Journals

1. **Manuel J. Fonseca**, Alfredo Ferreira Jr. and Joaquim A. Jorge
   Content-Based Retrieval of Technical Drawings. To appear in a Special Issue of International Journal of Computer Applications in Technology (IJCAT) "Models and methods for representing and processing shape semantics", 2004

2. **Manuel J. Fonseca** and Joaquim A. Jorge.
   Towards Content-Based Retrieval of Technical Drawings Through High-Dimensional Indexing. Computers and Graphics, 27(1):61-69, January 2003 (Reviewed version of the paper from SIACG'02).

3. **Manuel J. Fonseca** and Joaquim A. Jorge.
   Experimental Evaluation of an On-Line Scribble Recognizer. Pattern Recognition Letters, 22(12):1311–1319, 2001 (Reviewed version of the paper from RecPAD'00).

Collections

1. **Manuel J. Fonseca**, Bruno Barroso, Pedro Ribeiro and Joaquim A. Jorge
   Retrieving ClipArt Images by Content. To appear In Proceedings of the 3rd International Conference on Image and Video Retrieval (CIVR'04), Lecture Notes in Computer Science. Springer-Verlag, 2004.

2. **Manuel J. Fonseca**, Bruno Barroso, Pedro Ribeiro and Joaquim A. Jorge
   Retrieving Vector Graphics Using Sketches. To appear In Proceedings of the Smart Graphics Symposium (SG'04), Lecture Notes in Computer Science. Springer-Verlag, 2004.

3. César F. Pimentel, **Manuel J. Fonseca**, and Joaquim A. Jorge.
Experimental Evaluation of a Trainable Scribble Recognizer for Calligraphic Interfaces. In Dorothea Blostein and Young-Bin Kwon, editors, Graphics Recognition Algorithms and Applications, 4th International Workshop, GREC 2001, Selected Papers, volume 2390 of Lecture Notes in Computer Science, pages 81–91. Springer-Verlag, 2002.

4. Joaquim A. Jorge and **Manuel J. Fonseca**.
A Simple Approach to Recognize Geometric Shapes Interactively. In A. K. Chhabra and D. Dori, editors, Graphics Recognition Recent Advances, volume 1941 of Lecture Notes in Computer Science, pages 266–274. Springer-Verlag, September 2000.

International Conferences

1. **Manuel J. Fonseca**, Bruno Barroso, Pedro Ribeiro and Joaquim A. Jorge
Sketch-Based Retrieval of ClipArt Drawings. To appear In Proceedings of the Advanced Visual Interfaces (AVI'04), Gallipoli, Italy, May 2004. ACM Press.

2. **Manuel J. Fonseca** and Joaquim A. Jorge.
Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases. In Proceedings of the 8th Int. Conference on Database Systems for Advanced Applications (DASFAA'03), pages 267–274, Kyoto, Japan, March 2003. IEEE Computer Society Press.

3. **Manuel J. Fonseca** and Joaquim A. Jorge.
Towards Content-Based Retrieval of Technical Drawings through High-Dimensional Indexing. In Proceedings of the 1st Ibero-American Symposium in Computer Graphics (SIACG'02), pages 263–270, Guimarães, Portugal, July 2002.

4. **Manuel J. Fonseca**, César Pimentel and Joaquim A. Jorge.
CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding, pages 51–58, Palo Alto, USA, March 2002.

5. César F. Pimentel, **Manuel J. Fonseca**, and Joaquim A. Jorge.
Experimental Evaluation of a Trainable Scribble Recognizer for Calligraphic Interfaces. In Proceedings of the Fourth Int. Workshop on Graphics Recognition (GREC 01), Canada, September 2001.

6. **Manuel J. Fonseca** and Joaquim A. Jorge.
Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In Proceedings of the 9th Int. Conference on Fuzzy Systems (FUZZ-IEEE 00), volume 1, pages 291–296, San Antonio, USA, May 2000.

7. **Manuel J. Fonseca** and Joaquim A. Jorge.
Experimental Evaluation of an On-Line Scribble Recognizer. In Proceedings of the 11th Portuguese Conference on Pattern Recognition (RecPAD'00), pages 23–30, Porto, Portugal, May 2000.

8. Joaquim A. Jorge and **Manuel J. Fonseca**.
   A Simple Approach to Recognize Geometric Shapes Interactively. In Proceedings of the Third Int. Workshop on Graphics Recognition (GREC'99), pages 251–258, Jaipur, India, September 1999.

1

Introduction

Present–day CAD systems and vector–based drawing applications provide powerful tools to create and edit vector graphics in many domains, such as architecture, mechanics, automobile industry or mould industry (for technical drawings) and presentation charts, clip-art drawings or illustration graphics (for more general drawings). Even though reusing past drawings is common practice in such domains, there are almost no developed mechanisms to support this activity in an automated manner. Thus, it becomes important to develop new systems to support automatic classification and retrieval of vector drawings based on their contents, rather than relying solely on textual annotations or metadata for such purposes.

The remainder of this chapter describes the problem, enumerates the key objectives of our dissertation research, presents a short overview of our work, summarizes the main contributions and finally presents the dissertation outline.

1.1 Problem Description

Research in the domain of technical drawings [Do 98], indicates that project libraries with old case studies are crucial to help designers identify relevant features to include or problems to avoid in new designs. Moreover, in some design firms, designers often work by making or copying diagrams from colleagues in their design team for further development [Do 95]. Furthermore, during task analysis performed in the context of ongoing research projects [Consortium 00] and in informal conversations with draftspeople, we

found out that industrial designers often include elements from libraries of ready-to-use components. Indeed, designers also re-use old drawings during the creation phase of a new project, to get at ideas or review insights from previous problems and their solutions.

For more general vector drawings, such as clip-arts, currently there are a lot of figures available for integration into documents, either on the Internet or on collections sold in optical media. Typically, such drawings tend to be archived and accessed by categories (*e.g.* food, shapes, transportation, etc.). Although people find it useful to incorporate such figures in their documents (presentations, thesis, reports, etc.), finding a particular drawing among hundreds of thousands is not an easy task. Granted, reusing drawings often saves time. However, manually searching for them is usually slow and problematic, requiring users to browse through large and deep file directories or navigate a complex maze of menus, dialogs and categories for component libraries or drawings. Moreover, CAD systems and vector-based editing tools while making the creation of new drawings easier, exacerbate the retrieval problems, because they do not provide adequate search mechanisms. Indeed, present-day CAD systems rely on conventional database queries and direct-manipulation to retrieve information. Furthermore, such search becomes humanly impossible when the number of drawings increases to tens of thousands, except for very well established ancilliary classification schemes.

Some solutions to this problem use textual databases to organize the information [Bakergem 90, Clayton 91]. These classify drawings by keywords and additional information, such as designer name, style, date of creation/modification and a textual description. However, solutions based on textual queries are not satisfactory, because they force users to know in detail the meta-information used to characterize drawings. Worse, these approaches also require humans to produce such information when cataloging data. Moreover, textual description is not adequate to describe layout, shape and topology [Goodrum 00], suffers from low term agreement across indexers [Markey 88] and also between indexers and user queries [Enser 93, Seloff 90].

In contrast to the textual organization, we propose a visual classification scheme based

on shape, geometry and spatial relationships, which are better suited to this problem, because they take advantage of users' visual memory and explore their ability to sketch as a query mechanism. Indeed, recent studies [Gross 95, Do 98] show that designers use a small set of common graphical elements to describe the same drawings, validating our approach of using sketches to specify queries to databases of drawings. This visual classification scheme is combined with an indexing method that efficiently supports large sets of drawings, and classification schemes that allow us to hierarchically describe figures by level of detail and graph-based techniques to compute descriptors for such drawings in a form suitable for machine processing.

1.2 Research Goals

The primary goal of this work is to show that a reduced number of topological relationships combined with geometric information can be successfully used to describe and retrieve vector drawings based on their content. To that end we provide a Sketch-Based Retrieval (SBR) approach that allows users to quickly enter a sketch, as a query to find similar drawings in a database.

Drawings can be very complex and contain lots of details. However, when users want to search for a drawing, they might want to omit details and just specify a sketchy representation of its salient features. Or, in other situations, users just recall a specific part of the drawing. As a consequence, another goal of this work is to offer a multilevel description scheme that supports searching both by level of detail and by subparts of drawings.

Topological relationships among elements in drawings naturally translate to topology graphs. However, both graph and subgraph isomorphism are NP-complete problems, making the matching process complex, computationally very costly and hard to scale. Therefore, another goal of the present research is to find an alternative and more efficient method to graph matching.

Currently, any designer, draftsperson or ordinary user has ready access to thousands of drawings and any small company can have dozens of thousands of technical drawings in digital form. If we apply a multilevel description scheme to classify drawings, we can end up with hundreds of thousands or even millions of descriptors (depending on the complexity of drawings). As a consequence, another goal of this research is to develop a multidimensional indexing structure, to deal with large data sets of descriptors, offering mechanisms to compute measures of similarity between descriptors of drawings.

Finally, our last objective is to develop and evaluate, with potential users, a working prototype application for sketch-based retrieval of vector drawings by content, combining all implemented techniques, in order to validate our approach and its components.

1.3 Overview of the Present Work

In this thesis, we present a new approach to support classification, indexing and retrieval of drawings by content. Our method supports efficient processing of hand-sketched queries and automatic feature extraction from drawings. These are indexed based on the topological relationships between objects, avoiding an exhaustive search through the entire database, during the matching process. Additionally, we also store geometric information about each drawing. For simplicity, drawings are described in terms of two topological relationships, Inclusion and Adjacency. For geometry, we use a set of geometric features, such as ratios between areas and perimeters of some special polygons, like the convex hull, the enclosing rectangle or the largest inscribed triangle. The set of topological relationships between picture elements are converted into a topology graph. Before extracting this information from drawings, we apply a simplification step, where useless objects are removed.

Since graph matching is computationally costly, we convert graphs into descriptors by computing their spectrum, which is defined by the eigenvalues of its adjacency matrix. Then, we perform ”graph matching”, by comparing graph descriptors, since similar

graphs have similar descriptors.

To offer flexibility in specifying queries, we developed a new multilevel description technique, which describes drawings and subparts of it using different levels of detail. This way, users can retrieve drawings by providing either a complete sketch, a coarse representation of the drawing, or a subpart of it.

Descriptors created during the classification phase are then inserted into a multidimensional indexing structure, the NB-Tree, which provides a very efficient nearest neighbor search algorithm. This indexing structure also supports descriptors of variable dimension and can deal with large data sets. Using this indexing structure avoids the exhaustive search and comparison with all elements in the descriptor set.

We developed two prototype applications to index and retrieve drawings by content using our approach. One prototype works with technical drawings from the mould industry and other can search for clip-art drawings. We chose these two domains to validate our approach, because mould drawings have very rich topological information, while clip-arts are characterized mainly by geometric information. Figure 1.1 shows sample screen-shots of the two developed prototypes.

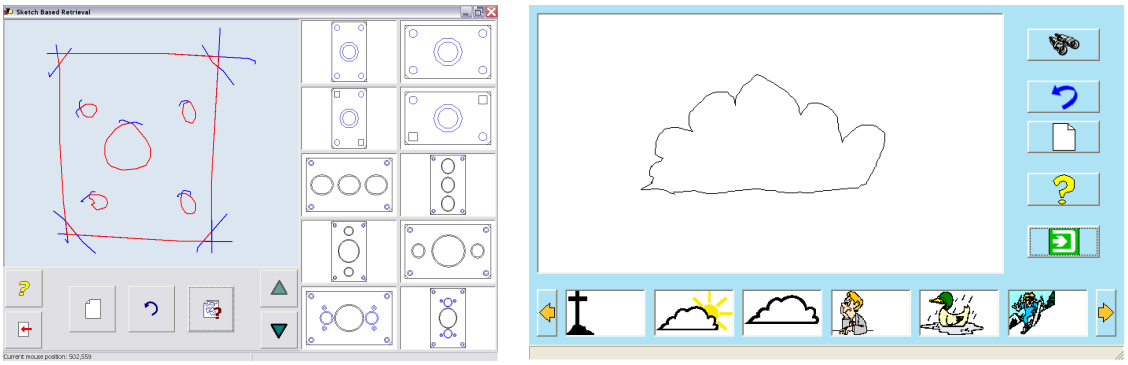


Figure 1.1: Two prototypes using our approach, one for mould drawings (left) and another for clip-art drawings (right).

1.4 Thesis and Contributions

The work described in this dissertation provides new mechanisms to classify, index and retrieve large databases of drawings by content. This is achieved by integrating multilevel description methods to support indexing different details of the same drawing, multidimensional indexing structures to speed up search and graph spectrum techniques to compute descriptors from topology graphs, in a modular architecture for sketch-based retrieval systems. The contributions of this research include the following:

- **New data structure for multilevel description**

  One original concept explored in our work is the description by level of detail [Fonseca 04d]. This technique explores different views of the same drawing, where the number of details increases while we "zoom-in" figures. This multilevel scheme is also applied to subparts of drawings, producing an hierarchical description. All these views of the same drawing are then mapped into multidimensional descriptors, providing us with a way to describe and retrieve either by partial matching or by coarse representation of drawings. This method of describing drawings by level is new in the area of content-based retrieval, since most approaches describe drawings as a whole.

- **Heuristic based on graph spectrum to speed-up search**

  Topology graphs describing drawings need to be compared during matching. However, graph isomorphism is an NP-complete problem. To overcome this, we have developed heuristics, computationally lighter than graph matching, that provide a mechanism to quickly filter-out unwanted drawings during the matching process [Fonseca 04d]. Topology graphs constructed during the classification phase are converted into multidimensional points by computing graph spectra. These descriptors are then used to index drawings by their topology and to compute the similarity between graphs. While similar approaches have been proposed elsewhere, ours is unique in using the full spectrum of the graph. We have found that this yields

superior performance.

- **Indexing structure for multidimensional data points of variable dimension**

  We developed the NB-Tree [Fonseca 03a, Fonseca 03b], a new multidimensional indexing structure based on dimension reduction. The NB-Tree offers simple, yet efficient indexing mechanisms for high-dimensional data points of variable dimension, while existent structures only support data of fixed dimension. Our technique offers methods for point, range and nearest-neighbor queries, presents good results for large data sets (around 1 million) and for data points of high dimensionality (greater than 200) and outperforms well known indexing structures.

- **Library of features to describe geometry**

  To acquire geometric information from drawings we developed a recognition library called CALI [Jorge 99, Fonseca 00b, Fonseca 00d, Fonseca 00c, Jorge 00, Fonseca 01, Fonseca 02b]. This library is able to identify a set of geometric shapes and gestural commands. However, instead of using CALI to recognize shapes or gestures from polygons, we use it to compute a set of geometric features, such as area and perimeter ratios from special polygons. This set of features are computed for each polygon in the drawing to yield a multidimensional feature vector that describes its geometry. While these features describe objects independently of their orientation, scale or line type, the set of describable objects is not limited by imposing *a priori* classification. Furthermore, our method presents better precision values than well established techniques.

- **Generic architecture for content-based retrieval**

  The architecture embodied in our approach [Fonseca 04d, Fonseca 02a, Fonseca 03c] also introduces a marked departure from previous techniques. Indeed, it combines the "normal" components presented in this systems (feature vector computation, feature vector matching and returning of similar results) with a multidimensional indexing structure, to deal with large sets of drawings, and with a pre-filtering step that uses topology graphs and graph spectrum. Additionally, our approach is de-

signed to deal with vector drawings, while almost all existent methods and systems process digital images (bitmaps).

1.5 Dissertation Outline

The rest of this dissertation is organized into five chapters and three appendices.

Chapter 2 surveys the work related to our research, which includes content-based retrieval of images, drawings and 3D objects. We also present research on multidimensional indexing structures, graph matching and shape representation.

In Chapter 3 we describe our approach for sketch-based retrieval of drawings by content. We also present our method based on graph spectra to perform topology matching and a new multilevel scheme for hierarchical description and indexing using levels of detail . Finally, we describe our method for representing geometric information and detail our matching algorithm.

Chapter 4 introduces our multidimensional indexing structure, the NB-Tree, describing its associated algorithms and the analysis to determine its computational complexity. A complete set of experimental tests, comparing its performance to other well known indexing structures is also presented.

In Chapter 5 we describe two prototypes developed using our approach. One system retrieves mould drawings while the other aims at retrieving clip-art drawings. We discuss experimental results garnered from using these systems and present the main outcomes from corresponding usability tests.

Chapter 6 presents an overall discussion of our work and proposes conclusions on applying our approach to retrieve drawings by content. We discuss possible extensions to the present work and propose new directions for further research.

Finally, we complete this dissertation with additional materials contained in three appendices. Appendices A and B present the protocol used during experimental evaluation

of our prototypes. Appendix C describes our recognition library, CALI, presenting an experimental evaluation of its recognition rate using representative data.

2

Related Work

Recently there has been considerable interest in querying Multimedia databases by content. However, most such work has focused on image databases as surveyed by Shi-Kuo Chang [Chang 99]. Another relevant survey by Yong Rui [Rui 99] analyzes work on image retrieval systems that use color and texture as main features to describe image content. Notwithstanding drawings in electronic format are represented in different structured form, (vector graphics), which requires wholly different approaches from image-based methods. While some early work [Bakergem 90, Clayton 91] attempted to index technical drawings through textual databases, such approaches are not suited to use the rich visual association mechanisms and draftspeople use of sketches to recover information.

Sketch-Based Retrieval of vector drawings is closely related to a number of structural techniques aimed at image understanding and pattern recognition. Indeed, our work spans a cluster of related fields ranging from retrieval of images by content, to indexing techniques used to speed up search, to graph and tree isomorphism to perform topology matching and to shape representation and matching (pattern recognition) to describe and compare visual elements in both queries and candidate drawings. In this chapter we first look briefly at the previous work on Content-Based Retrieval of digital images. Then we review the main approaches to Content-Based Retrieval of vector drawings, examining their methods and domains of application. Then, we briefly survey recent work on Content-Based Retrieval of 3D objects and review methods for shape representation. Finally, we study techniques for graph and tree isomorphism and analyze in depth current indexing structures for multidimensional data points.

2.1 Content-Based Retrieval

A current trend in multimedia information processing is toward Content-Based Retrieval. Rather than manually generate text-based descriptions, content-based retrieval works by matching the query against an automatically generated representation of the content of the element to retrieve. There are multiple types of visual entities that can be retrieved based on their content. In this section we analyze content-based retrieval of digital images, vector drawings and 3D objects.

2.1.1 Digital Images

Content-based image retrieval (CBIR) relies on characterizing primitive features such as color, shape and texture that can automatically be extracted from images using simple processing techniques. Queries to CBIR systems are expressed mainly by providing visual exemplars of the type of image or image attribute to search for. In such systems users may specify queries either by entering an existing image, by submitting a sketch, by clicking on a texture palette or by selecting a particular shape of interest. The system then returns stored images which exhibit the highest degree of similarity to requested features.

In the remainder of this section, we briefly describe prevailing features used for CBIR, namely Color, Texture, Shape and Color Layout. Finally, we present recent and relevant surveys in the CBIR area.

2.1.1.1 Extracted Features

Features extracted from the whole image are called global features. These include color, texture or image layout. While global features convey information about the image as a whole, local features are extracted from an object, or a segment of an image to convey information just about that object. Local features can describe the location of a particular object within an image, its intersection with neighboring features, its shape, etc. When querying large collections of images, global features can be very useful to narrow down

the search space. On the other hand they cannot provide enough information to estimate the similarity between images. For this, they must be supplemented by local features. This subsection briefly presents the features most often used to describe digital images.

Color

Color is one of the most widely used visual features in Image Retrieval. It is relatively robust to background complication and independent of image size and orientation. Typically, retrieving images based on color similarity is achieved by computing color features for each image in the database. Color Histograms are the most commonly used color feature representation since it is insensitive to small object distortions and is easy to compute. Color histograms represent the proportion of pixels within an image holding specific colors. They are computed by splitting the range of the data into equally sized bins, where the value of each bin is the number of pixels that have the same color. Besides Color Histograms, other color feature representations have been applied in Image Retrieval, such as Color Moments [Striker 95] and Color Sets [Smith 95]. Although these color features describe approximations of color histograms, they have the advantage of producing more compact representations.

Color moments are the statistical measures that characterize the histogram distribution. Resulting descriptors include the first (average), the second (variance) and third-order (skewness) moments of the color histogram. Color Sets are an approximation to Color Histograms, providing a compact alternative for representing color information. Because color sets are binary feature vectors, a binary search tree can be used to allow fast search in large-scale image collections.

Texture

Texture refers to visual patterns having properties of homogeneity that do not result from the presence of a single color or intensity [Smith 96, Ma 95]. Indeed, texture is an innate property of virtually all surfaces, including clouds, trees, bricks, hair, fabric, etc. It con-

tains important information about the structural arrangements of surfaces and their relationship to the surrounding environment. Methods for texture feature extraction generally fall into four categories: statistical methods, geometric methods, model-based methods and signal processing methods. The statistical methods are among the earliest proposed in the literature. They often use spatial frequency, co-occurrence matrices, edge frequency, among other techniques. From these many simple features such as energy, entropy, homogeneity, coarseness, contrast, correlation, etc., are derived to construct descriptors. Although the selection of features to use in Image Retrieval is application dependent, the majority of existing CBIR systems use a combination of color and texture features to describe image content.

Shape

The primary mechanisms used for shape retrieval, resorts to identifying features such as lines, boundaries, aspect ratio, circularity, etc. Some of these shape representations are invariant to translation, rotation and scaling, while others are not. Furthermore, they can be divided into two categories, boundary-based and region-based. The former uses only the outer boundary of shapes while the latter techniques use the entire shape region. The most successful techniques in these two categories are respectively Fourier Descriptors [Rui 96] and Moment Invariants [Hu 77]. Fourier Descriptors use the Fourier transformed boundary as the shape feature, while Moment Invariants use region-based statistical moments. Shape queries are generally achieved by computing feature vectors from an example image provided by the system or from an user sketched shape.

Color Layout

Although global color features are simple to calculate and can provide reasonable discriminating power in Image Retrieval, they tend to yield too many false positives when the image collection is large. Many research results suggested that color layout, (by combining color features to spatial relations between regions), can provide a better solution to Image Retrieval [Carson 97]. The extension of global color features to local regions, is naturally

achieved by dividing the whole image into sub-blocks, as suggested in [Chua 97].

2.1.1.2 Surveys

Aigrain et al's survey provides a comprehensive overview of approaches to image similarity matching for database retrieval [Aigrain 96], while Idris and Panchanathan examine in detail approaches to image indexing and retrieval based on shape, color, texture and spatial location [Idris 97].

Gudivada and Raghavan provided a taxonomy for CBIR approaches describing their characteristics and limitations [Gudivada 97]. The authors examine a number of image database applications to identify their retrieval requirements and to structure them from a domain independent perspective. From this study, the authors define a taxonomy for image attributes and identify a number of generic query operators. Finally, they propose a novel system architecture for CBIR that supports generic query operators. Their architecture is structured in a way to enable systems to inherit only those query operators that are useful to the specific domain of the application. Authors also demonstrate the versatility and effectiveness of their architecture by configuring a prototype for two image retrieval applications: realtor information system and face retrieval system.

Eakins et al most complete report [Eakins 99] reviews the current state of the art in CBIR. Their findings were based both on a review of the relevant literature and on discussions with researchers and practitioners in the field. Authors identify three levels of abstraction to characterize images, namely, primitive features, such as color, texture or shape, logical features, such as the identity of objects shown, and abstract attributes, such as the significance of the scenes depicted. Although current CBIR systems operate only on the lowest of these levels, authors found that most users demand higher levels of retrieval. Authors conclude that despite its current limitations, CBIR is a fast developing technology with considerable potential that should be exploited. Finally, this report presents a set of recommendations for users, managers of image collections, UK government agencies, software developers, etc., concerning the research in this field.

Rui and Chang produced an extended survey [Rui 99], from more than 100 papers, covering technical achievements in the research area of CBIR. The authors address research aspects of image feature representation and extraction, multidimensional indexing and system design. Authors concluded the survey by identifying open research issues and by suggesting future promising research directions, such as the inclusion of human in the image retrieval process (*e.g.* relevance feedback), the use of high-level concepts to describe images, the evolution to web-based search engines, the development of effective high dimensional indexing techniques to deal with large collections of images, the definition of performance evaluation criterion and standard testbed, the study of human perception of image content and finally, the integration of disciplines and media.

Smeulders et al presented a review of 200 references in CBIR [Smeulders 00]. They start by discussing the scope of the content-based retrieval, analyzing the characteristics of the domain and sources of knowledge. After, authors analyze methods to describe image content, such as color texture and local shape. The interpretation of a single image, the similarity between a pair of images, query definition, result displaying and interaction are also addressed. Finally, the authors present their view at the system level, discussing indexing techniques, system architecture and evaluation performance.

More recently Goodrum provide an overview of current research in image information retrieval, focusing on three aspects of image research: text-based retrieval, content-based retrieval and user interactions with image information retrieval systems [Goodrum 00]. The authors also suggest an outline of areas for future research, such as cross-disciplinary approaches utilizing both text and image features for retrieval, efficient indexing structures, classification mechanism, vocabulary control, users needs, similarity measures and presentation of retrieval results. Finally, they conclude their review with a call for image retrieval evaluation studies.

2.1.2 Vector Drawings

In the last decade, the majority of indexing and retrieval systems were developed for digital images. Vector drawings, which required different approaches, only in recent years received attention from researchers. In the remainder of this section we present systems and approaches related to the retrieval of vector drawings.

Gross's Electronic Cocktail Napkin [Gross 95, Do 95, Gross 96a] addressed a visual retrieval scheme based on diagrams, to indexing databases of architectural drawings. Users draw sketches of buildings, which are compared with annotations (diagrams), stored in a database and manually produced by users. Even though this system works well for small sets of drawings, the lack of automatic indexation and classification makes it difficult to scale the approach to large collections of drawings.

Nabil et al [Nabil 96] presented a set of techniques for similarity retrieval based on the 2D Projection Interval Relationships representation (2D-PIR), including methods for dealing with rotated and reflected images. 2D-PIR is a symbolic representation of directional as well as topological relationships among spatial objects. It adapts three existing representation formalisms (Allen's temporal intervals [Allen 83], 2D-Strings [Chang 87] and topological relationships [Egenhofer 91]) and combines them in a novel way to produce a unified representation of pictures. Authors claim that their method offers more information about spatial relationships between objects in a picture than traditional methods. However, during matching the query gets compared to all the symbolic representations stored in the database, making this work difficult to scale up for large collections of images.

The S3 system [Berchtold 97b] supports managing and retrieving industrial CAD parts, described by their geometry (2D contour) and thematic attributes. S3 retrieves parts using bi-dimensional contours drawn using a graphical editor. Contours can also be taken from sample parts stored in a database. S3 supports four similarity algorithms, to describe and match queries against parts in the database, namely a modified version of the

Mehrotra-Gary algorithm [Mehrotra 93] that determines angles and lengths of segments in a region, the Angular profile [Badel 92], which computes angles between line segments defined by sample points from the contour, Section coding that determines the proportion of the contour inside each sector resulting from equally dividing the surrounding circle of the polygon, and finally, a Fourier-based method [Berchtold 97a] to allow partial similarity. Authors also used this system as a testbed for developing and testing new indexing algorithms, such as the R*-Tree and the X-Tree. S3 relies exclusively on matching contours, ignoring spatial relations and shape information, making this method unsuitable for retrieving complex multi-shape drawings.

Müller and Rigoll presented a novel approach [Müller 99] to retrieve engineering drawings based on stochastic models. Drawing databases are searched using sketches or shapes which represent details in mechanical parts. Their approach aims to retrieve images containing certain specified details and locating these details in the retrieved images. Authors represent drawings and queries using a pseudo 2-D Hidden Markov Model augmented with filler states, which describe the remaining part of the engineering drawing apart from the query shape. However, their method only supports simple queries, representing a single element. More complex queries including several elements with spatial relationships between them are not contemplated. Furthermore, the search mechanism is not appropriate for large collections of drawings, since they perform a sequential scan through the database comparing the query with all indexed drawings.

Park and Um described an approach to retrieve complex 2D drawings of mechanical parts based on the dominant shape [Park 99]. Objects are described by recursively decomposing its shape into a dominant shape, auxiliary components and their spatial relationships (Inclusion and adjacency). Their approach breaks, a complex drawing into many adjacent closed loops or blocks, each with an arbitrary polygonal shape. Spatial relationships among blocks are extracted to create a graph. Then, each block is decomposed and recursively divided into many fragments of primitive shapes, yielding another type of graph that describes the geometry of blocks. The combination of these two types

of graphs produces a complex graph structure, where each node of the graph describing spatial arrangement, has another graph describing the geometry of the correspondent block. The small set of base geometric primitives and the not-so-efficient matching algorithm, based on the breadth-first tree matching, make it hard to handle large databases of drawings.

Huet et al devised two alternative methods for content-based retrieval of technical drawings representing patents [Huet 01]. One approach describes drawings using histograms of attributes and retrieves similar occurrences by comparing histograms. To make their representation insensitive to rotation, translation or scaling, authors describe each line of the drawing in relation to all other lines, using the relative angle and the relative position of the two lines. Also, to reduce the number of attributes and to increase the influence of local visual information, the authors construct a n-nearest neighbor attributed graph from the line-segment set. Nodes in the graph represent lines, while edges represent the n closest lines. From this graph, the histograms are computed, by considering the neighborhood of each line. The other approach uses these attributed graphs to describe drawings and computes a graph similarity measure in the retrieval process. Searching from similar drawings takes a considerable amount of time (in both algorithms), which is directly dependent on the size and complexity of schemes stored in the database. Worse, each of the retrieval algorithms requires comparing the query to each drawing in the database.

Leung and Chen proposed a sketch retrieval method to store and retrieve unstructured free-form hand-drawings stored in the form of multiple strokes [Leung 02]. They use shape information from each stroke exploiting the geometric relationship between multiple strokes for matching. Their approach then computes a matching score between the query and each sketch in the database. More recently, authors improved their system by also considering spatial relationships between strokes [Leung 03b, Leung 03a]. Authors use a graph based description, which is similar to ours [Fonseca 02a], but their method only considers inclusion, while we also describe adjacency between drawing elements.

Their approach has two drawbacks. First, they use a restricted number of basic shapes (circle, line and polygon) to classify strokes. Second, their approach is difficult to scale to large databases of drawings, since they explicitly compare the query with all the figures in the database.

Looking at the majority of existing content-based retrieval systems for drawings, we can observe three things. The first is scalability: most published works use databases with few elements (typically less than 100). The second is complexity: figures stored in the database are simple elements not representing sets of real drawings. Third, complex matching schemes (graph matching) make it difficult to adopt efficient algorithms.

2.1.3 3D Objects

Although the retrieval of 3D objects is not directly related to our work, we decided to examine current research approaches in this area to understand the new trends and challenges. Furthermore, we can see this domain of application as a new area where the core of our approach can be extended with small changes in our algorithms.

Elad et al present a technique to searching for similar objects in a database of 3D objects [Elad 01], specified using VRML. Authors tried to address the subjective matter of object similarity, by providing an approach based not only on geometric feature similarity but also letting the user influence subsequent searches by marking some of the results as ”good” or ”bad”. This way users indicate personal preferences to prescribe both content and semantics. This algorithm, which is both iterative and interactive, uses statistical moments computed from 3D objects’ surfaces as features to define object signatures. To make the similarity measure invariant with changes in spatial position, scale and rotation, all feature vectors are normalized. Experiments exhibit promising results, since after very few iterations (usually, 1-4) searching seems to converge to objects that users had in mind. However, it is not clear that this approach scales well for large collections of 3D Objects, since they do not use any indexing structure to avoid a sequential scan of the complete database. Authors suggest that their work can be extended with new features, such as

topological traits, color and texture.

Lau and Wong surveyed some representative work on 3D model retrieval [Lau 02], focusing their analysis in feature matching, dividing existing methods into three approaches: geometry-based, frequency-based and topology-based. They also show how these approaches were implemented and compare their relative performance using a common geometry database. One key finding is that-frequency based methods perform significantly better than geometry-based. Moreover, authors propose using Haar wavelets for 3D model matching and evaluated it. Experimental results revealed that they are easy to implement and have a good retrieval performance at a minimal feature size.

Chen et al proposed a system for retrieving 3D models based on the visual similarity among objects [Chen 03], using 2D projections. The main idea is that if two 3D objects are similar then they will also look similar from all viewing angles. To apply this idea, the authors encode multiple orthogonal projections of a 3D object using both Zernike moments and Fourier descriptors. To create query models authors provide an interface allowing users to use an existent 3D model or to draw 2D shapes. Queries specified in 2D are easy to depict and can be compared to 3D models readily, since in their approach descriptors for 3D models are composed of 2D shapes. Authors demonstrate that their approach is robust against similarity transformation, noise, model degeneracy, etc., and provides better performance than other competing approaches. Although authors use a scheme to speed up the retrieval process, they can not avoid the comparison between the query and each of the models in the database. For databases with a large number of models this procedure is impracticable, requiring some kind of indexing. Moreover, according to authors, partial matching is still a problem for their approach.

Funkhouser et al [Funkhouser 03] describe a method for retrieving 3D shapes using textual keywords, 2D sketched contours, 3D sketches or a combination of both. The authors developed a new 3D shape descriptor based on spherical harmonics that is descriptive, concise, efficient to compute and invariant to rotations. Authors find that queries combining both text and shape produce better results than either one alone and that users

prefer to specify queries in 2D than in 3D. While their approach relies on silhouettes and their fitting to projections of 3D images, our method, which is based on structural matching of graphical constituents, uses both shape and spatial relations.

Shokoufandeh et al presented a framework, in its preliminary stage, for shape matching through scale-space decomposition of 3D models [Bespalov 03]. Their algorithm is based on efficient hierarchical decomposition of metric data using its spectral properties. 3D objects are mapped into binary rooted trees, thus recasting the problem of finding a match between 3D models as the much simpler technique of comparing rooted trees. The authors are planning to use the structural parameters of tree representation as signatures for indexing purposes. This way, they expect to avoid the comparison between a query and every model in the database.

More recently Lou et al presented an approach for searching 3D engineering shapes [Lou 04]. Their system incorporates multiple feature vectors, relevance feedback, query by example and browsing. Additionally, they use a multidimensional indexing structure based on the R-Tree to support efficient execution. The system extracts four feature vectors, namely moment invariants, geometric ratios, principal moments and eigenvalues of the skeletal graph, to represent 3D objects. The extraction of these feature vectors is performed after applying different transformations, namely, shape normalization, voxelization, skeletonization and skeletal graph construction. A drawback of using various types of features vectors to describe shapes, is the space needed to store descriptors. Authors utilized the system to perform extensive experimentation, using a database of 113 real engineering shapes to test the effectiveness of the four methods. Experimental tests revealed the following descending order of the average precision of feature vectors: principal moments, moments invariants, geometric parameters and eigenvalues. Authors also proposed and evaluated a multistep similarity search strategy, that combines moment invariants and geometric parameters, to improve effectiveness. Experimental evaluation revealed that this technique presents a precision value 51% higher than the best individual feature vector (principal moments). Another drawback of this system is that it only

supports query by example, requiring the use of shape models, either created using a 3D modeling tool or picked from a set of shapes sampled from the database. It is not possible to specify a query using sketches or a rough representation of the desired shape. Finally, results are presented in a 3D view, allowing users to manipulate models and offering a better way to evaluate the similarity of results.

2.2 Shape Representation and Matching

Since shape is one of the primary low level image features used in content-based image retrieval, shape representation became a fundamental issue in this type of applications. The main objective of shape description is to measure geometric attributes of an object, that can be used for classifying, matching and recognizing objects. Moreover, a shape representation scheme should be affine invariant, robust, compact, easy to derive, easy to match and perceptually meaningful.

There is a lot of related work on shape representation. Mehtre et al grouped existing techniques into two categories: boundary-based and region-based [Mehtre 97]. The former use only the contour or border of the object shape, which is crucial to human perception in judging shape similarity, and completely ignore its interior. The latter exploit shape interior information, besides the boundary. More recently Safar et al presented a taxonomy [Safar 00b] that complements Mehtre's classification.

As examples of boundary-based methods we have Fourier descriptors [Persoon 77], chain code [Freeman 78], autoregressive models [Kauppinen 95], polygonal approximation [Gu 95], curvature scale space [Mokhtarian 96] and shape signature [Davies 97]. In region-based methods, we encountered geometric moments [Hu 62], Zernike moments [Mehtre 97], grid representation [Lu 99] and area.

Kupeev and Wolfson presented an algorithm for the detection of perceptual similarity among planar shapes [Kupeev 94, Kupeev 96]. Their technique divides a contour by straight lines which are simultaneously parallel to the X axis and tangent to the contour.

The area restricted by these lines and by the contour, called a lump, will correspond to the vertices of a special weighted graph used to represent shapes, while edges correspond to the adjacency relationships between lumps. During comparison graphs are reduced using a ”small leaf” trimming procedure until the resulting graphs are isomorphic. From this representation they compute a similarity measure via a polynomial-time complexity algorithm. Although this method is effective in recognizing shapes represented by closed planar curves without self-intersections, it is not effective when we consider convex shapes.

Safar et al presented a study where they measured the effectiveness of the similarity retrieval of four boundary-based methods for shape representation, under different conditions [Safar 00a]. They compared a Fourier descriptor method, a grid-based method, a Delanay triangulation method and their Touch-point-vertex-angle-sequence (TPVAS) technique. Their TPVAS method describes the shape of an object using touch points and vertex angle sequence, extracted from the objects' minimum bounding circle. Experimental results demonstrated that their technique outperformed all the other methods and that it is the most robust.

Zhang and Lu studied and compared several shape descriptors [Zhang 01], namely, Fourier descriptors (FD), curvature scale space descriptors (CSSD), Zernike moment descriptors (ZMD) and grid descriptors, against properties like robustness, compactness, computation complexity, etc. Authors concluded that Zernike moment descriptors and Fourier descriptors get more credits than the other two methods and suggested to include Fourier descriptors as a shape representation method in MPEG-7[1], like CSSD and ZMD were.

Shock trees [Kimia 95] are another method to describe and compare shapes. Pelillo presented a solution to matching two shock trees by constructing the association graph [Pelillo 99]. Authors illustrate the power of this approach by matching articulated and

[1]MPEG-7 is an international standard, also known as ”Multimedia Content Description Interface”, that specify a set of descriptors (features) that can be used to describe various types of multimedia information [Jeannin 00].

deformed shapes described by shock trees. Shokoufandeh et al developed another approach to perform shock tree matching based on graph spectrum and Voronoi diagrams [Shokoufandeh 99]. While these approaches use trees (graphs) to describe the contour of simple shapes, we use graphs to represent the spatial structure of complex drawings.

More recently, Zhang and Lu performed two more studies, one comparing three region-based methods [Zhang 02] and another comparing two contour-based methods [Zhang 03]. In both studies authors used standard principles and standard MPEG-7 databases [Kim 00]. In the former, authors compared Zernike moment descriptors, grid descriptors and geometric moments descriptors. Results showed that the ZMD outperformed the other two methods, being the most suitable for effective and efficient shape retrieval. The latter study compared Fourier descriptors and curvature scale space descriptors. Experimental results showed that FD outperformed CSSD in terms of robustness, low computation, hierarchical representation, retrieval performance and suitability for efficient indexing.

Although contour-based methods, such as the Fourier descriptors, presented good results in these studies, they have limited applications. These methods cannot capture shape interior content and cannot deal with disjoint shapes, where the boundary may not be available. On the other hand, region-based methods can be applied to more general shapes, but usually require more computation resources.

2.3 Graph Matching and Related Techniques

Most of the techniques described so far, resort to statistic properties of image content. Spatial organization in a figure is a very important feature used to describe content. Since spatial relationships extracted from pictures are, most of the times, represented symbolically by a tree or a graph, we acn decide whether two pictures are similar by comparing the corresponding graph representations.

However, graph isomorphism in the general case is non polynomial (NP), while sub-

graph isomorphism is known to be NP-Complete [Garey 79]. Thus, comparing a graph with a set of graphs or finding occurrences of a subgraph in a set of graphs stored in a database becomes impracticable, mainly when these databases have thousands to millions of graphs. Despite the intensive research for over three decades still no efficient (polynomial-bound) algorithm for graph isomorphism is known. However, by finding approximate results or by imposing certain restrictions on the underlying graphs it is possible to derive algorithms of polynomial-time complexity. In the remainder of this section, we present a number of alternative solutions proposed in the past few years.

In 1976 Ullman presented an algorithm for both graph and subgraph isomorphism detection [Ullmann 76], based on backtracking and on a refinement procedure. Although this method presents an exponential behavior with the size of graphs, it is considered one of the fastest algorithms for the subgraph isomorphism problem, and still is one of the most used.

Messmer and Bunke proposed a new approach based on a preprocessing step in which model graphs are used to create a decision tree [Messmer 99, Messmer 95]. To that end they generate the set of all permutations of the adjacency matrix of a model graph and then organize it in a decision tree. At run time, subgraph isomorphisms are detected by means of decision tree traversal. The computational complexity of their algorithm is only polynomial (quadratic) in the number of vertices of the input graph, remaining independent of the number of model graphs in the database and of the number of edges in any of the graphs. The major drawbacks of their approach are the size of the decision tree and the preprocessing step, that grow exponentially with the size of the model graphs. To overcome these problems, authors presented some pruning techniques to cut down the size of the decision tree, with the cost of introducing some limitations to the original algorithm. In summary, this approach works well on small graphs, but does not scale well to larger graphs or large databases of graphs.

Jiang and Bunke introduced the concept of ordered graphs and ordered graph isomorphism [Jiang 99]. These graphs have the particularity of having all edges incident to

a vertex uniquely ordered. Authors show that the ordered graph isomorphism problem can be optimally solved in quadratic time by utilizing particular properties of ordered graphs. Experimental evaluation demonstrated that their, simple and easy to implement, algorithm outperforms Ullman's approach. However, this technique cannot be applied to general graphs.

In 1998 Bunke and Shearer proposed a new graph distance measure, based on the maximal common subgraph of two graphs [Bunke 98]. Authors formally proof that this new distance measure is a metric and arg that it is superior to graph edit distances, since it does not depend on edit costs. However, algorithms to compute the maximum common subgraph (MCS) of two graphs are conceptually simple, but are NP-complete. Several algorithms for approximate and exact MCS detection have been developed in the last years. Bomze et al surveyed a set of approximate algorithms, including analysis of their complexity and potential applications [Bomze 99].

Shearer et al presented a new algorithm to solve the largest (maximum) common subgraph problem [Shearer 00], based on the previous work developed by Messmer and Bunke [Messmer 99]. Authors use the largest common subgraph between the graphs representing images as a measure of image similarity. Although this algorithm presents a significant performance improvement over previous approaches, it inherits the major drawbacks from Messmer work, namely the exponential complexity for the preprocessing step and the exponential space required to store the decision tree.

Recently, Bunke et al presented a performance comparison of two exact algorithms for maximum common subgraph detection [Bunke 02]. One algorithm computes all possible MCSs between two graphs and them choose the larger. The other algorithm, first computes the association graph between the two graphs and then the maximum clique of the latter graph, that will be the MCS. Although both algorithms have a time complexity that grows exponentially with the number of nodes, the first is better for low density graphs while the second is better for high edge density graphs.

Giugno and Shasha described GraphGrep [Giugno 02], an application-independent method for querying graphs, finding all the occurrences of a subgraph in a database of graphs. Their algorithm uses hash-based fingerprinting to represent the graphs in an abstract form and to filter the database. Their approach is made of three steps, first they build the database to represent graphs as sets of paths and index it using an hash table; second, they filter the database based on the submitted query to reduce the search space; finally, they look for all matching subgraphs in the remaining graphs. Querying times are linear in the size of the database and exponential in the size of the query graph.

Raymond et al introduced a new graph similarity calculation procedure, RASCAL, for comparing labeled graphs [Raymond 02]. The method consists of an initial screening process to roughly determine a measure of similarity between two graphs, followed by a rigorous maximum common edge subgraph (MCES) detection algorithm to compute the exact degree and composition of similarity. The screening procedure is intended to determine rapidly whether the graphs being compared exceed some specified minimum similarity threshold (without resulting in any false dismissals) in order to avoid unnecessary calls to the more computationally demanding, graph matching procedure. The main disadvantage of this approach is that we have to compute the rough similarity measure between the query graph and each of the model graphs in the database and additionally, for each different query we have to compute new similarity values.

Another technique to overcome the complexity of graph isomorphism is based on graph spectrum [Cvetković 97]. Graphs can be characterized by some properties called invariants, their spectrum is one of them. Graphs are represented as a vector of eigenvalues (spectrum) computed from the adjacency matrix of the graph. Then, the similarity between graphs is measured by computing the Euclidean distance (for example) between the two descriptors. Although this technique assures that similar graphs have similar spectra, there are some collisions and different graphs can have the same spectrum.

Shokoufandeh et al developed an approach using this technique [Shokoufandeh 99]. However, to reduce the size of the resulting feature vector, the authors decided to use

the sum of eigenvalues, instead of using all eigenvalues directly. Then, to compute the distance between descriptors of graphs the authors use a Voronoi diagram.

From all the surveyed techniques to measure graph similarity, we decided to use an approach based on graph spectrum. Contrary to Shokoufandeh's approach, we use all eigenvalues to describe graphs and an efficient multidimensional indexing structure to determine the similarity between graphs.

2.4 Multidimensional Indexing Structures

From the previous analysis we can conclude that shape information is converted to feature vectors by using a number of methods surveyed to describe shape. Furthermore, spatial information coded through graphs can also be mapped into multidimensional descriptors by computing the spectrum of the graph. In summary, all extracted information from figures can be mapped into multidimensional feature vectors. Thus, a content-based retrieval system using these type of information must include in its architecture an efficient multidimensional indexing structure.

To support processing large amounts of high–dimensional data, a variety of indexing approaches have been proposed in the past few years. Some of them are structures for low–dimensional data that were adapted to high–dimensional data spaces. However, such methods while providing good results on low–dimensional data, do not scale up well to high–dimensional spaces. Recent studies [Weber 98] show that many indexing techniques become less efficient than sequential search, for dimensions higher than ten. Other indexing mechanisms are incremental evolutions from existing approaches, where sometimes, the increased complexity does not yield comparable enhancements in performance. Other indexing techniques based on dimension reduction return only approximate results. Finally, structures combining several of the previous approaches have emerged. Often, the corresponding algorithms are very complex and unwieldy.

We describe existing indexing techniques by dividing these in four main categories.

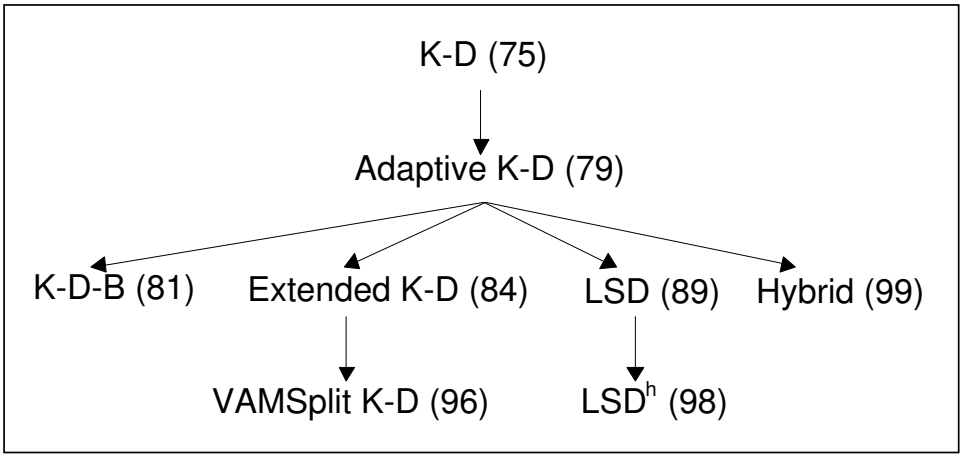


Figure 2.1: Indexing structures derived from the K-D-Tree.

The first two include all structures derived from the K-D-Tree (see Figure 2.1) and derivatives of the R-Tree (see Figure 2.2). The main difference between these two categories lies in the approach to dividing the data space. Structures in the first category use space-partitioning methods that divide the data space along predefined hyper-planes regardless of data distribution. The resulting regions are mutually disjoint, with their union being the complete space. Structures from the second class use data-partitioning methods, which divide the data space according to their distribution. This can yield possible overlapping regions. The other two categories of indexing structures include dimension reduction approaches and a class of indexing schemes combining several methodologies to improve performance.

2.4.1 Indexing Structures Derived from the K-D-Tree

White and Jain presented the VAM-Split K-D-Tree [White 96a], which is an extension to the K-D-Tree [Ooi 87] to improve the efficiency and to reduce the storage space. The main difference between these two trees is in the way they split the data space. While the K-D-Tree uses the 50% quantile, the VAM-Split K-D-Tree uses the maximum variance and the median. This tree outperforms the K-D-Tree and the SS-Tree, but while these two are dynamic index structures, the VAM-Split K-D-Tree is a static structure, *e.g.* all data items must be available at creation time.

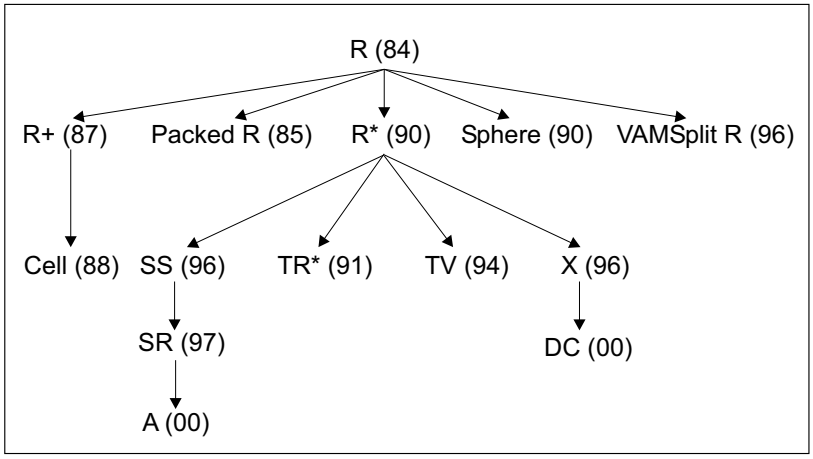


Figure 2.2: Indexing structures derived from the R-Tree.

In 1998 Henrich proposed the LSD^h-Tree [Henrich 98] as an improvement to the LSD-Tree [Henrich 89]. The paging routine of the LSD^h-Tree keeps part of the tree in main memory and stores some sub-trees in disk, when the structure becomes to large, keeping the tree balanced. The LSD^h-Tree combines the low fanout from K-D-Trees with the advantage that R-Trees have in not covering empty space. This structure is slightly better than the X-Tree.

The Hybrid-Tree [Chakrabarti 99] was introduced by Chakrabarti in 1999 to combine the advantages from space-partitioning structures and data-partitioning structures. The Hybrid-Tree always splits a node using a single dimension to guarantee that the fanout is independent of data dimension. However, this method allows overlapping regions as data-partitioning structures do. This structure outperforms both the SR-Tree and the hB-Tree [Lomet 90].

2.4.2 Indexing Structures Derived from the R-Tree

White and Jain presented the SS-Tree [White 96b], an R-Tree-like index structure that uses minimum bounding spheres (MBSs) instead of minimum bounding rectangles (MBRs). Even though the use of spheres reduce the overlapping of regions and consequently the SS-Tree outperforms the R*-Tree, spheres tend to overlap for high-dimensional

spaces.

The VAMSplit R-Tree [White 96a] was introduced in 1996 by White et al, which is a static index structure that splits the data space depending on the maximum data variance, like the VAM-Split K-D-Tree. While this structure seems to outperform the R-tree, it has the shortcoming that all data need to be known *a priori*. Thus it is not suited for dynamically varying data sets and environments.

Berchtold, Keim and Kriegel proposed the X-Tree [Berchtold 96], an index structure adapting the algorithms of R*-Tree to high-dimensional data. The X-Tree uses two techniques to deal with high-dimensional data: First, it introduces an overlap-free split algorithm, which is based on the split history of the tree. Second, where the overlap-free split algorithm would lead to an unbalanced directory, the X-Tree omits the split and the according directory node is enlarged becoming a super-node. The X-Tree outperforms both the R*-Tree and the TV-Tree [Lin 94].

In 1997 Katayama proposed the SR-Tree [Katayama 97], an improvement to the SS-Tree combining concepts from both the R*-Tree and SS-Tree approaches. This structure uses both MBRs and MBSs as an approximation in the directory. This way, the region spanned by a node is the intersection of the MBR and MBS associated to that node, thus reducing region overlap between sibling nodes. This structure has been shown to outperform both the R*-Tree and the SS-Tree. Furthermore, Sakurai presented an experimental evaluation [Sakurai 00], using non-uniform data, where the SR-Tree outperforms the VA-File [Weber 98].

Recently, Sakurai proposed the A-Tree [Sakurai 00], which is an index structure for high-dimensional data that introduced the notion of relative approximation. The basic idea of the A-Tree is the use of Virtual Bounding Rectangles to approximate minimum bounding rectangles or objects. Thus, on one node of the tree we have information about the exact position of the region MBR and an approximation of the relative position of its sons. Authors claim that the A-Tree outperforms the VA-File and the SR-Tree.

2.4.3 Dimension Reduction

Berchtold et al' Pyramid Technique [Berchtold 98] uses a special partitioning strategy optimized for high-dimensional data. Its basic idea is to perform a dimension reduction allowing the use of efficient unidimensional index structures to store and search data. The Pyramid Technique divides the data space into 2D pyramids whose apexes lie at the center point. In a second step, each pyramid is cut into several slices parallel to the basis of the pyramid forming the data pages. The Pyramid Technique associates to each high-dimensional point a single value, which is the distance from the point to the top of the pyramid, according to a specific dimension (j_{max}). This value is later used as a key in the B^+-Tree. Points in the same slice will have the same identification value. As a result, for skewed distributions, many points in the B^+-Tree will be indexed by the same key. The Pyramid Technique outperforms the X-Tree and the sequential scan using uniformly distributed data. Although the Pyramid Technique was developed mainly for uniformly distributed data, authors suggested an extended version to handle real (skewed) data. This version works by shifting the center point to the barycenter of the data. However, in a dynamic scenario, this implies the recalculation of all index values, *i.e.* redistribution of points among the pyramids and reconstruction of the B^+-Tree, each time the center of the cluster changes.

Shepherd et al presented an overview of an efficient file-access method for similarity searching in high-dimensional spaces, named CurveIx [Shepherd 99]. The basic idea is to order the d-dimensional space in many ways, with a set of (one-dimensional) space-filling curves. Each curve constitutes a mapping from $R^d \rightarrow R^1$, yielding a linear ordering of all points in the data set. During retrieval only points whose location is close to the curve-location of the query are considered. Their approach uses standard one-dimensional indexing schemes such as the B-Tree to perform searching. Close points along the space-filling curve tend to correspond to close points in the d-dimensional feature space. However, some near neighbors may be mapped far apart along a single space-filling curve. As a result, this method does not have an accuracy of 100%, *i.e.* some near neighbors may be

ignored.

Yu et al proposed a new index scheme, called iMinMax(θ) [Yu 04], that maps high-dimensional points to single dimension values determined by their maximum or minimum coordinate values. By varying the θ value they can optimize the iMinMax structure to different distributions of data sets. As other dimension reduction methods, this scheme also uses a B^+-Tree to index the resulting single dimension points. The iMinMax structure was mainly developed to address window search (range queries) while still supporting approximate K-NN search (with accuracy lower than 100%) at the expense of increased runtimes for higher accuracy. This approach outperforms both the VA-File and the Pyramid Technique for range queries.

Another new technique for KNN search, called iDistance [Yu 01], was presented by Yu et al in 2001. Their approach relies on partitioning the data and defining a reference point for each partition. Points are then indexed through the distance to the reference point of its partition, using a B^+-Tree. The effectiveness of iDistance depends on how the data are partitioned and how reference points are selected. To do that their approach first compute a set of statistics on points distribution from the database and then based on this it defines partitions and predicts clusters. Although the iDistance outperforms both the A-Tree and the iMinMax(θ), it has the shortcoming that all data must be available at creation time to extract statistics. This way, it is not suitable for dynamic contexts where there are a lot of insertions and deletions into the database. Moreover, when the dimension increases, the radius of each cluster becomes very large, resulting in the intersection of almost every cluster during a KNN query.

More recently, Zhang et al presented a new structure called P^+-Tree [Zhang 04], that supports both window and KNN queries efficiently. This approach divides the data space into subspaces based on clustering, then points in each subspace are mapped onto a single dimensional space using the Pyramid Technique and finally data are stored in a B^+-Tree. According to authors' performance study, the P^+-Tree outperforms the Pyramid Technique, the iMinMax(θ) and the iDistance approaches, both for KNN and window

queries. Although authors claim that their structure performs well for high dimensional data points, they just present results for points with dimensions up to 32. Like the iDistance structure, one of the P^+-Tree drawbacks is the need to know the entire data set *a priori*, to compute clusters. Another disadvantage of this structure is the need to rebuild the entire P^+-Tree after a number of insertions and deletions of data points, since clusters' centers change, affecting the overall performance of the indexing structure.

2.4.4 Other Indexing Techniques

In 1998 Weber et al proposed the VA-File [Weber 98], a method to search points of high dimension. The authors show that existing structures (R*-Tree and X-Tree) are outperformed on average by a simple sequential scan if the dimension exceeds around ten. So, instead of developing another indexing structure they proposed to speed-up the sequential scan. The basic idea of the VA-File is to keep two files; one with an approximate version of data points and other with the exact representation. When searching points, the approximation file is sequentially scanned with some look-ups to the exact file whenever it is necessary. The VA-File outperforms both the R*-Tree and the X-Tree when the dimension is higher than six, but its performance is very sensitive to data distribution.

Berchtold et al described a new contribution to the near neighbor search through precomputation and indexing the solution space [Berchtold 00b]. First, they precompute the result of any nearest neighbor search, which corresponds to computing the Voronoi cell of each data point. In a second step, they store approximations of each Voronoi cells in an indexing structure, the X-Tree. Subsequently, searching for the nearest neighbor maps to a simple point query on the index structure. Although the complexity of these approximations increases quickly with the data dimension, this new technique outperforms the X-Tree.

More recently, Berchtold et al proposed the IQ-Tree [Berchtold 00a] which combines a compression method with an index structure trying to join the best of both worlds. The IQ-Tree is a three-level structure, with a regular directory level, a compression level and

a third level that contains the actual data. While the VA-File uses a fixed number of bits to compress all partitions, the IQ-Tree uses a different compression scheme for each partition depending on the density of data points. This structure outperforms both the X-Tree and the VA-File.

2.5 Discussion

Content-Based Image Retrieval approaches herein surveyed use mainly primitive features, such as color, texture and shape to describe the content of digital images. Although vectorial drawings require different features and approaches to describe their content, some techniques from CBIR systems could be applied to vector drawing retrieval (*e.g.* indexing structures).

Existent methods for vector drawing retrieval rely mainly on shape information to describe drawing content. Few of them [Park 99, Leung 03a] combine shape and topological information. Furthermore, only the S3 system includes an indexing structure in its architecture, allowing the manipulation of large sets of drawings.

Shape representation techniques described in this chapter are mainly applied to objects extracted from digital images after a segmentation process. Although they can be applied to vector drawings, the algorithms are complex and computationally costly.

For structural comparison, many techniques resort to graph-based descriptions. However, graph matching and subgraph isomorphism are NP-Complete problems for which we do not have efficient solutions. Researchers have tried to find approximate results or to explore particularities of graphs to achieve algorithms of polynomial-time complexity. However, such solutions are only valid for specific types of graphs or impose too many restrictions both on the number and size of graphs.

From all surveyed indexing structures, techniques using dimension reduction present some of the more promising results. However, even the more recent ones [Zhang 04], that claim good performance are not dynamic, requiring all data points to be known at

creation time. Furthermore, all analyzed indexing mechanisms do not support data points of variable dimension, an important requirement in the context of our work, where the dimension of feature vectors can vary from object to object and the maximum dimension can not be predicted in advance. In such scenarios, current indexing structures will define a (maximum) fixed dimension for the data space and feature vectors of smaller dimensions will be padded with zeros. However, if we are to insert new feature vectors of larger than a specified maximum dimension, the indexing structure must be rebuilt to accommodate the new data, at the sacrifice of interactive performance.

2.6 Summary

This chapter reviewed the large set of matters related to content-based retrieval of vector drawings from large databases. This body of work was focused in areas relevant to our approach, such as content-based retrieval (of digital images, vector drawings and 3D objects), shape representation, graph matching and multidimensional indexing. Our research is unique in that it combines these technologies allowing users to efficiently and effectively retrieve vector drawings by content from large databases, using sketches as queries, as we will show in the next chapter.

3

Sketch-Based Retrieval of Drawings

As we have seen in the previous chapter, the majority of existent content-based retrieval approaches deal with digital images (bitmaps), requiring methods based mainly on color and texture to describe their content. Vector drawings (CAD, clip-arts, etc.), on the other hand, require different features to describe their content.

In this chapter we describe our approach that improves on systems developed by Berchtold [Berchtold 97b], Park [Park 99] and Leung [Leung 03a], since we aim to retrieve vector drawings privileging the use of spatial relationships and geometric information. Indeed, our method is more ambitious in the sense that we do automatic simplification, classification and indexation of existing drawings, to make the retrieval process both more effective and accurate. These activities imply specifying a description mechanism to describe drawings and sketched queries. Additionally, fast and efficient algorithms to perform similarity matching between sketched queries and a large database of drawings are required.

We start this chapter by presenting an overview of the main components from our approach: Classification, Interface & Query, Indexing and Matching. After, we deeply describe and evaluate two of our main contributions: a data structure for topology matching and a mechanism to describe geometric information. Finally, we describe our matching process.

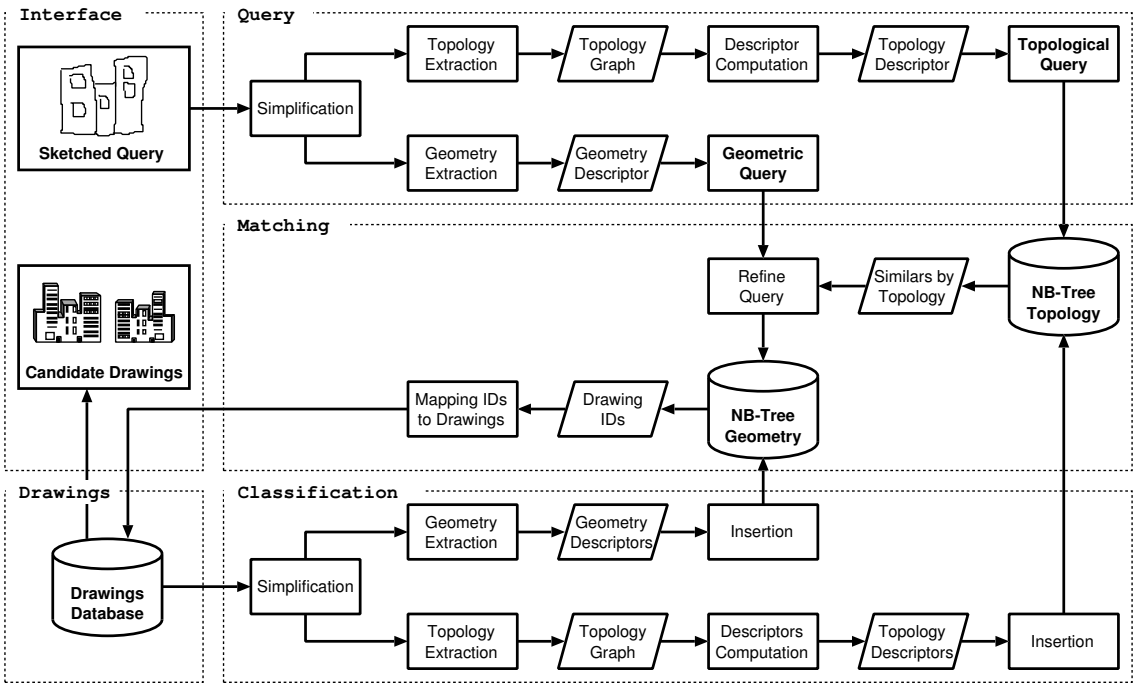


Figure 3.1: Detailed architecture for our approach.

3.1 Overview of Our Approach

Our approach solves both scalability, complexity and matching problems, identified in previous surveyed work, by developing a mechanism for retrieving vector drawings, in electronic format through hand-sketched queries, taking advantage of user's natural ability at sketching and drawing. Furthermore, unlike the majority of systems cited in the previous chapter, our method was developed to support large sets of drawings. To that end, we devised a multidimensional indexing structure (NB-Tree) that scales well with growing data set size.

Figure 3.1 presents a very detailed diagram of our system architecture, identifying its main components, which we describe in the remainder of this chapter.

3.1.1 Classification

Content-based retrieval of pictorial data, such as digital images, drawings or graphics, uses features extracted from the corresponding picture. Typically, two kinds of features

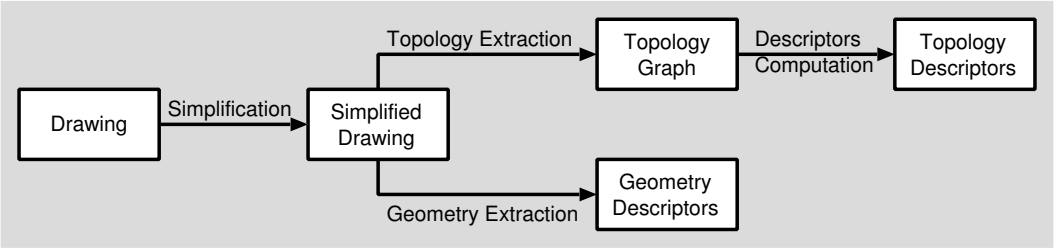


Figure 3.2: Overview of the Classification component.

are used. Visual features encode information, such as color, texture and shape. Relationship features describe topological and spatial relationships among objects in a picture. While digital images rely mainly on color and texture to describe their content, for vector drawings these features are irrelevant. Thus, we focus on topology, a global feature of drawings, and geometry, a local feature.

Our classification process starts by applying a simplification step, to eliminate most useless elements (see Figure 3.2). The majority of drawings contains many details, which are not necessary for a visual query and increase the cost of searching. We try to remove visual details (i.e. small-scale features) while retaining the perceptually dominant elements and shapes in a drawing. The main goal of this step is to reduce the number of entities to analyze in subsequent steps of the classification process, in order to speed up queries.

After simplification we identify visual elements, namely polygons and lines, and extract shape and topological information from drawings. We only use two relationships, **Inclusion** and **Adjacency**, which are a simplified subset of the topological relationships defined by Egenhofer [Egenhofer 92]. Relationships thus extracted are then compiled in a Topology Graph, where ”parent” edges mean `Inclusion` and ”sibling” connections mean `Adjacency`, as illustrated in Figure 3.3. Even though our relationships are less discriminating than Egenhofer’s original set, they hold regardless of rotation and translation. Evidently, the limitations of this scheme lie in that only two very simple spatial relations are considered. While this may not seem very effective for simple, trivial graphics, it becomes more and more efficient as the structure of drawings increases.

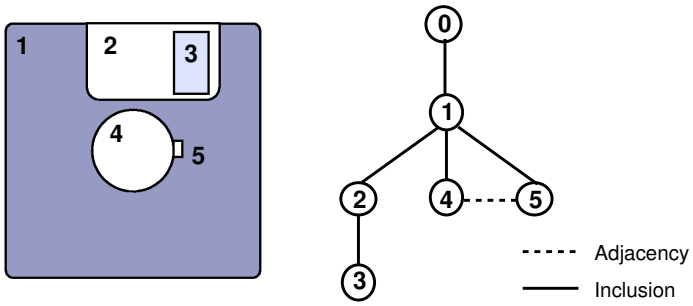


Figure 3.3: Drawing (left) and correspondent topology graph (right).

However, topology graphs are not directly used for searching similar drawings, since graph matching is a NP-complete problem. We use the corresponding graph spectra instead. For each topology graph to be indexed in a database we compute descriptors based on its spectrum [Cvetković 97]. In this way, we reduce the problem of isomorphism between topology graphs to computing distances between descriptors. To support partial drawing matches, we also compute descriptors for subgraphs of the main graph. Moreover, we use a new multilevel description scheme to describe drawings, by dividing them in different levels of detail and then computing descriptors at each level. This combination of subgraph descriptors and levels of detail, provides a powerful way to describe and search both for drawings or subparts of drawings, which is a novel feature of our work.

To compute the graph spectrum we start by determining the eigenvalues of its adjacency matrix. The resulting descriptors are multidimensional vectors, whose size depends on graph (and corresponding drawing) complexity. Very complex drawings will yield descriptors with higher dimensions, while simple drawings will result in descriptors with lower size.

To acquire geometric information about drawings we use a general shape recognition library called CALI [Fonseca 00d, Fonseca 01, Fonseca 02b]. This enables us to use either drawing data or sketches as input, which is a desirable feature of our system, as we shall see later on. After submitting each polygon of the drawing to this recognition library, we end up with a set of multidimensional feature vectors that describe their geometry.

The geometry and topology descriptors thus computed are inserted in two different in-

dexing structures, one for topological information and another for geometric information, respectively.

3.1.2 Query and Interface

Our system includes a Calligraphic Interface [Jorge 94] to support the specification of hand-sketched queries, to supplement and overcoming limitations of conventional textual methods. The query component performs the same steps as the classification process, namely simplification, topological and geometric feature extraction, topology graph creation and descriptor computation. This symmetrical approach is unique to our method. In an elegant fashion two types of different information (vector drawings + sketches) are processed by the same pipeline.

3.1.3 Indexing

Since we need to index most subgraphs of a given graph to allow for subgraph matching, indexing hundreds to thousands of drawings yields a large database comprising tens of thousands or potentially millions of descriptors. Thus, at the core of our approach, we need to develop efficient indexing structures for storing these descriptors. Such indexing mechanisms should minimize the number of false positives that have to be tested by a similarity search. However, indexing should not discard any relevant drawings. Good indexing methods should also be dynamic, allowing on-line insertion and removal of descriptors and should also scale well with growing data set sizes. Furthermore, the indexing structure should support data points of variable dimension, since descriptors have different sizes of which we do not know in advance the maximum. Additionally, to support matching similar drawings the indexing structure needs a fast and reliable K nearest-neighbors scheme, since most interesting candidates will probably yield near matches to the query. However, nearest neighbor search in high-dimensional data spaces is a difficult problem [Gaede 98].

We developed a new multidimensional indexing structure, the NB-Tree [Fonseca 03a, Fonseca 03b], that satisfies the requirements enumerated before, providing us with an efficient indexing mechanism for high-dimensional data points of variable dimension. In Chapter 4 we present a detailed description of our indexing structure and the correspondent experimental evaluation.

3.1.4 Matching

We perform the matching between a sketched query and drawings into the database in two steps. First, we select a set of drawings topologically similar to the query. This step works as a filter, reducing the number of potential candidates to compare in the next step. Second, we use geometric information to further refine the set of candidates. At the end, we have a measure of similarity between the sketched query and drawings retrieved from the database.

3.2 Data Structure for Topology Matching

Content-Based Retrieval systems use information extracted from objects and spatial relationships between them. Thus spatial information presented in drawings should be preserved during the classification process so that users can easily retrieve those from the database. Our approach not only preserve topological information, but use it to index drawings in a database. We choose to index by topology because it is a global feature of drawings, providing us with a good characteristic to distinguish figures from each other. This way, our searching process starts by selecting drawings with similar topology and then computes the geometric similarity between drawings. Topological information extracted is combined in a Topology Graph that describes the global topology among all elements in a drawing.

Since graph isomorphism is a well-known NP-complete problem, we try to avoid its computation, by reducing the problem to the calculation of distances between descriptors.

To that end, we map topology graphs into a multidimensional vector and perform nearest neighbor queries, in this D-space, to find associated similar graphs. In this manner, and since drawings have a reach topological structure, we can use topology as a discriminating index to reduce the number of candidate results.

In the remainder of this section, we identify the more relevant topological relationships and show how they are combined to create a topology graph. After, we describe how topology graphs are mapped into multidimensional vectors and present our novel multi-level description scheme to describe drawings hierarchically by level of detail. Finally, we present experimental results to validate our data structure for topology description and matching.

3.2.1 Topological Relationships

Spatial relationships may be classified into *directional* and *topological* relations. The most frequently used directional relationships are `North`, `South`, `East`, `West`, `North-East`, `NorthWest`, `SouthEast` and `SouthWest`, as depicted in Figure 3.4.

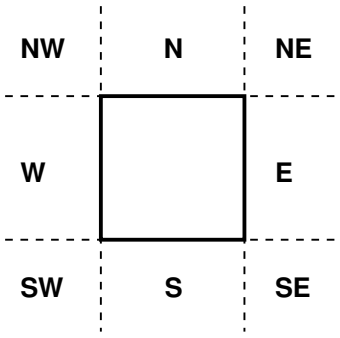


Figure 3.4: Directional relationships.

For topological relationships Egenhofer [Egenhofer 89, Egenhofer 92] presented a set of eight relations between two planar regions, namely `Disjoint`, `Meet`, `Overlap`, `Contain`, `Inside`, `Cover`, `Covered-By` and `Equal` as illustrated in Figure 3.5.

We decided to restrict spatial relationships to those that are independent of translation and rotation of drawings. Since directional relationships do not guarantee that, we just

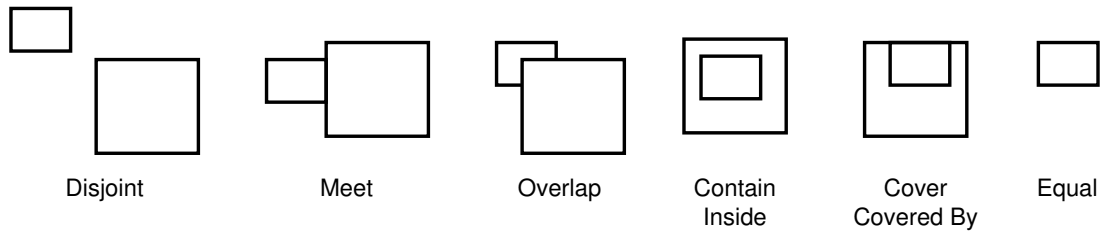


Figure 3.5: Topological relationships.

consider topological relationships. Moreover, to both make our approach less restrictive and the topology graph simpler, we simplified the topological relationships defined by Egenhofer, starting from his neighborhood graph for topological relationships (depicted in Figure 3.6 (left) and described in [Egenhofer 92]). Our set of topological relationships groups neighbor relations, yielding three topological relationships between two polygons: `Disjoint` , `Include` and `Adjacent` (see Figure 3.6 (right) and definitions below).

Definition 3.1 *If all intersections among all faces are empty then the two polygons are **Disjoint**.*

Definition 3.2 *If polygon P1 contains completely polygon P2 or P1 equals P2 then P1 **Includes** P2.*

Definition 3.3 *If two polygons meet or if they intersect then they are **Adjacent**.*

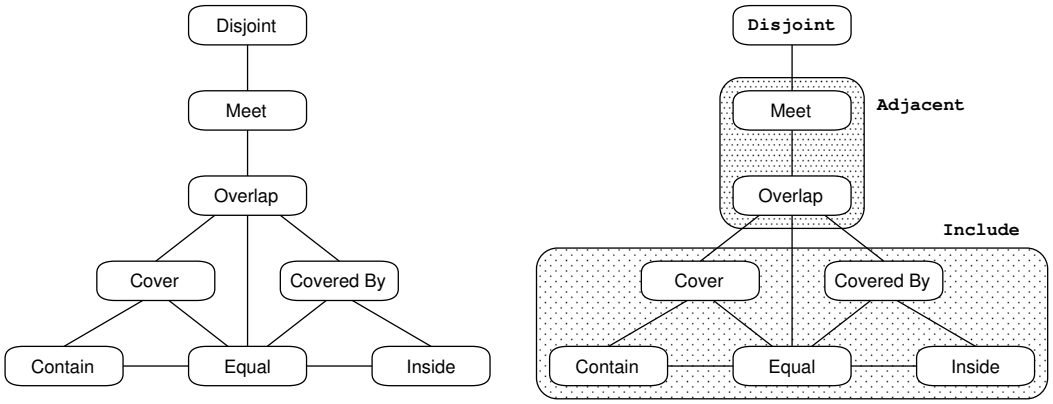


Figure 3.6: Topological relationships originally defined by Egenhofer (left) and our simplified version (right).

By reducing the number of relationships, we made our approach less restrictive and, more important, we got a simpler topology graph. Furthermore, by using only these two relationships, we guarantee the stability of graphs when drawings are subject to changes. Thus, similar drawings will be described by similar topology graphs.

Leung's approach [Leung 03a] reduced even more the number of relations, using just inclusion. In our opinion, this simplification is excessive, because inclusion alone is not enough to correctly describe topological relationships between objects and increases the number of collisions among graphs.

3.2.2 Topology Graph

Topological relationships extracted from drawings are compiled in a Topology Graph, where "vertical" edges mean `Include` and "horizontal" connections mean `Adjacent`, as illustrated in Figure 3.7.

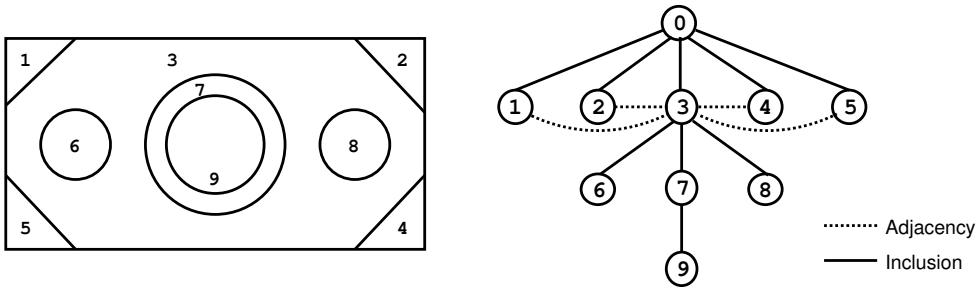


Figure 3.7: Technical Drawing (left) and correspondent topology graph (right).

Our topology graph has a well defined structure, being very similar to "a rooted tree with side connections". It has always a root node, representing the whole drawing. Sons from the root represent the dominant blocks (polygons) from the drawing, *i.e.* blocks that are not contained in any other block. The next level of the graph describes polygons contained by the blocks identified before. This process is applied recursively until we get the complete hierarchy of blocks. As a conclusion, we can say that each graph level adds more drawing details. So, by going down in the depth of the graph, we are "zooming in" in drawing details. Based on this, we can present the following definitions.

Definition 3.4 *A **Topology Graph** is a "tree-like" graph G(V,I,A), where V is a finite nonempty set of nodes representing objects (polygons) in the drawing, I is a set of edges representing inclusion and A is a set of edges representing adjacency.*

Definition 3.5 ***Inclusion edges** connect nodes from two consecutive levels of the graph, while **Adjacency edges** connect sibling nodes in the same level.*

3.2.3 Topological Descriptors

As we said before, we reduce the graph matching problem to the computation of distances between descriptors. To achieve this we use graph spectra [Cvetković 97] to map graphs into vector descriptors that can be manipulated through a multidimensional indexing structure. The spectrum of a graph G (which consist of n eigenvalues) is computed from the eigenvalues of its adjacency matrix A. The eigenvalues of G (that correspond to the eigenvalues of A) are usually denoted by $\lambda_1, \cdots, \lambda_n$, and we assume that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Finally, the largest eigenvalue, $\mu_1(G) = \lambda_1$, is called the index of the graph.

Since the spectrum of a graph is a graph invariant it is natural to think that it would provide a polynomial algorithm to decide whether two graphs are isomorphic and thereby solve the graph isomorphism problem. However, graph spectrum is not a *complete invariant*. A graph invariant ϕ is said to be complete if, for any graphs G, H, the equality $\phi(G) = \phi(H)$ implies that G is isomorphic to H. Although, isomorphic graphs have the same spectrum[1], two graphs with the same spectrum need not be isomorphic.

According to Cvetković [Cvetković 97] and Shokoufandeh [Shokoufandeh 99] the use of graph spectrum as an indexing method is valid since: (1) it captures local topology, (2) is invariant to subgraph re-order and (3) is stable, since small changes in the graph produce little changes in its spectrum. However, resulting descriptors are not unique.

[1] Graphs with the same spectrum are called cospectral or isospectral graphs.

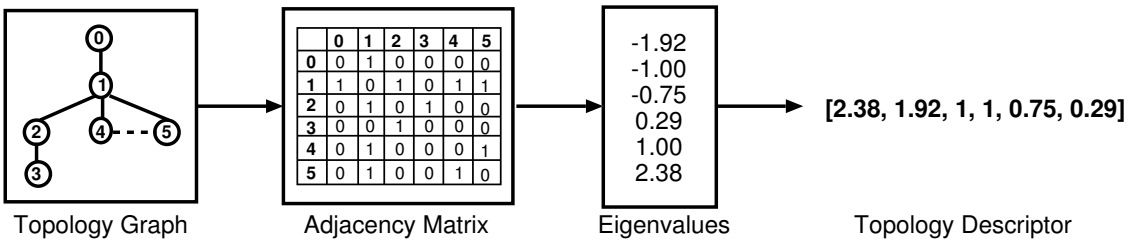


Figure 3.8: Block diagram for topology descriptor computation.

More than one graph can have the same spectrum, which gives rise to collisions similar to these in hashing schemes. In [Shokoufandeh 99] authors argue that these collisions occur rather infrequently, a claim seemingly verified by our experiments, described in Section 3.2.4.1.

3.2.3.1 Descriptor Computation

Figure 3.8 presents the block diagram for computing the topology descriptor. First we compute the adjacency matrix of the graph, second we compute its eigenvalues and finally we sort the absolute values to obtain the topology descriptor. Resulting descriptors are multidimensional points, whose dimension depends on graph (and drawing) complexity. Very complex drawings will produce descriptors with high dimensions, while simple drawings will produce descriptors with low dimensions.

We assume that our topology graphs are undirected graphs, yielding symmetric adjacency matrices and assuring that eigenvalues are always real. Furthermore, by computing the absolute value and sorting it decreasingly, we exploit the fact that the largest eigenvalues are more informative about the graph structure. Additionally, the largest eigenvalues are stable under minor perturbation of the graph structure, making our topological descriptors also stable.

3.2.3.2 Multilevel Description

As we have seen previously, the topological organization of a drawing is described using a topology graph, which we map into a multidimensional descriptor. This way,

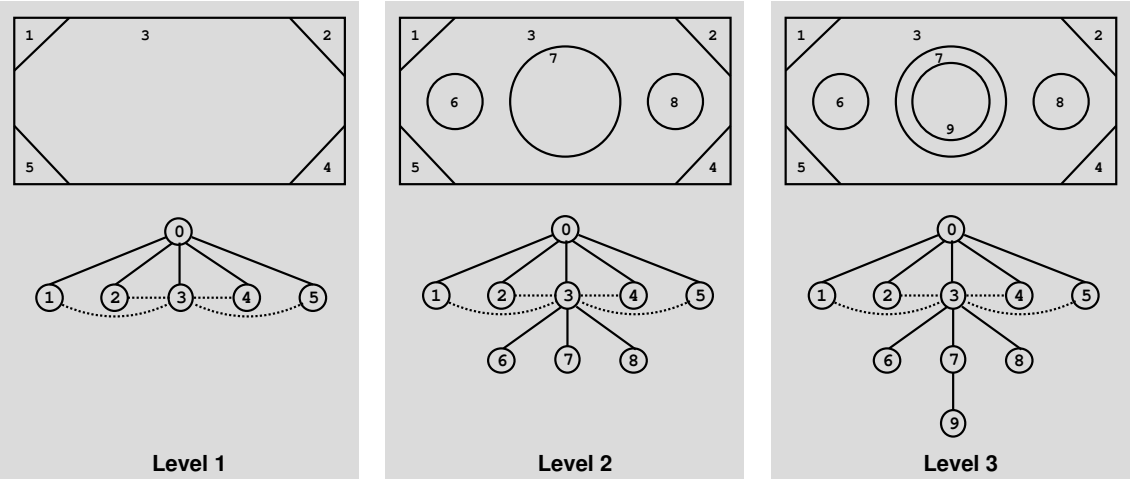


Figure 3.9: Different levels of detail and the correspondent graphs.

we get a descriptor for each graph (drawing) and we can compute the similarity between graphs by calculating a distance between the correspondent descriptors. However, when we have complex drawings in the database and users want to retrieve them, they might not remember or they might not want to sketch a complete query that includes every parts and details of the desired drawing. Thus, we need a mechanism to easy the task of sketching queries for complex drawings.

One solution is to divide the drawing in several parts and compute a descriptor for each. This way, we can search complex drawings by providing subparts of it as queries. Although this solves one problem, another still exist. Complex drawings have lots of details that sometimes users forget to draw when sketching a query. To overcome this we propose a novel multilevel description scheme to describe drawings hierarchically by levels of detail, allowing the use of rough approximations of drawings as queries. To that end we compute several descriptors for each drawing, one for each level of detail. At the end we have one descriptor per level, allowing the matching between a query and several representations of the same drawing.

To compute descriptors for subparts of drawings and for different levels of detail, we resort to the topology graph (see Figure 3.7). For subparts, we recursively divide the graph into various subgraphs and then we compute descriptors for each. To describe

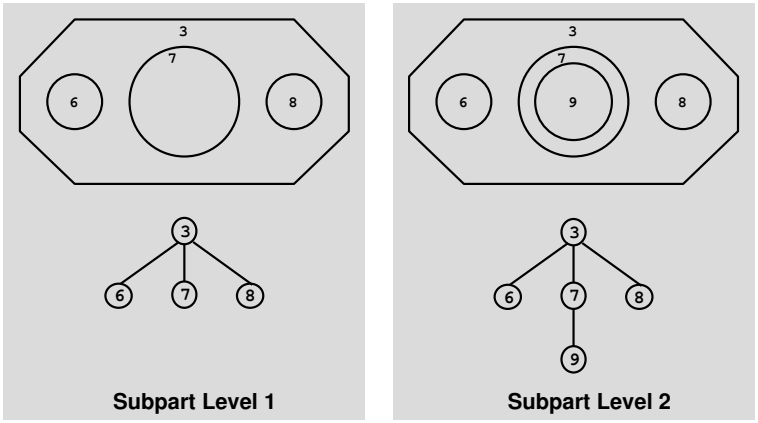


Figure 3.10: Subpart of the drawing with two levels of detail and the correspondent graphs.

different levels of detail, we exploit the ”tree-like” structure of our topology graph, and compute a descriptor for each level of the graph. Looking at Figure 3.9 we can see that by going down in the structure of our topology graph, we are adding more detail to the drawing. In this figure we can identify three different graphs, one for each degree of detail. So, if we now compute a descriptor for each of these levels, we end up with three ways to search for the current drawing, using more or less detail. This approach has also the merit to allow classifying subparts of drawings by computing descriptors for subgraphs of the main graph. Figure 3.10 illustrates the subgraphs thus extracted and their corresponding part of the drawing. We recursively apply the description by levels of detail to these subgraphs. The result of this process is a set of graphs and subgraphs that describe both the topology at different levels of detail and the different subparts of a drawing. In summary, after this multilevel description we have descriptors for the parts of the drawing shown in Figure 3.11.

3.2.4 Experimental Evaluation

To validate our selection of graph spectrum as the mechanism to describe and index topology graphs, we performed three experiments. The first study the stability of the graph spectrum under changes on its structure. The second, and the third measure and compare the accuracy of our approach based on all eigenvalues of a graph with other

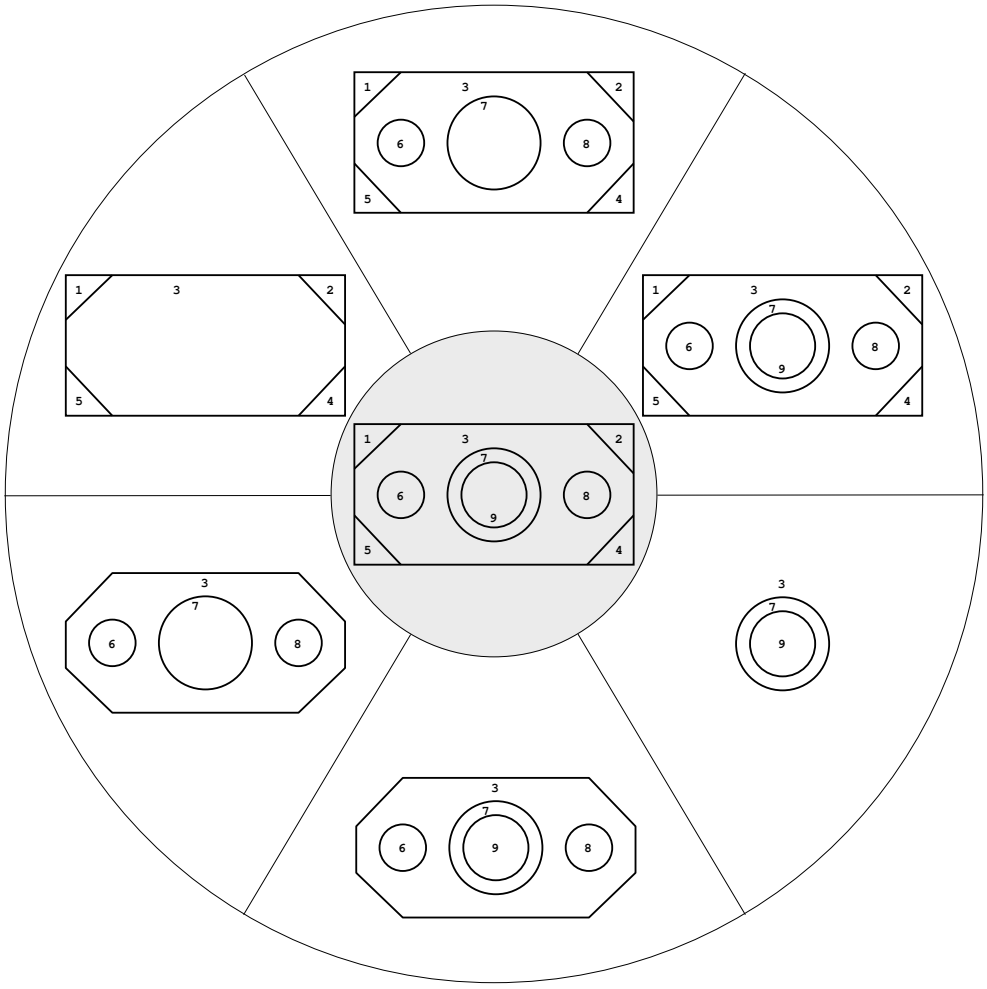


Figure 3.11: Several representations for a given drawing, using our multilevel description technique.

existing methods.

To that end we first create a small set of similar topology graphs with little differences from each other (see Figure 3.12). First we generated Graph 1. Then we created nine very similar graphs from this one, by removing or adding nodes and links (Graphs 2, 3, 4, 6, 7, 9, 10, 11 and 14). Finally, we created four graphs (Graphs 5, 8, 12 and 13) more different from Graph 1 than the others, by deleting a larger number of nodes and links. This set of graphs forms the basis for all tests described here and have the same structure as topology graphs built from topological relationships extracted from drawings.

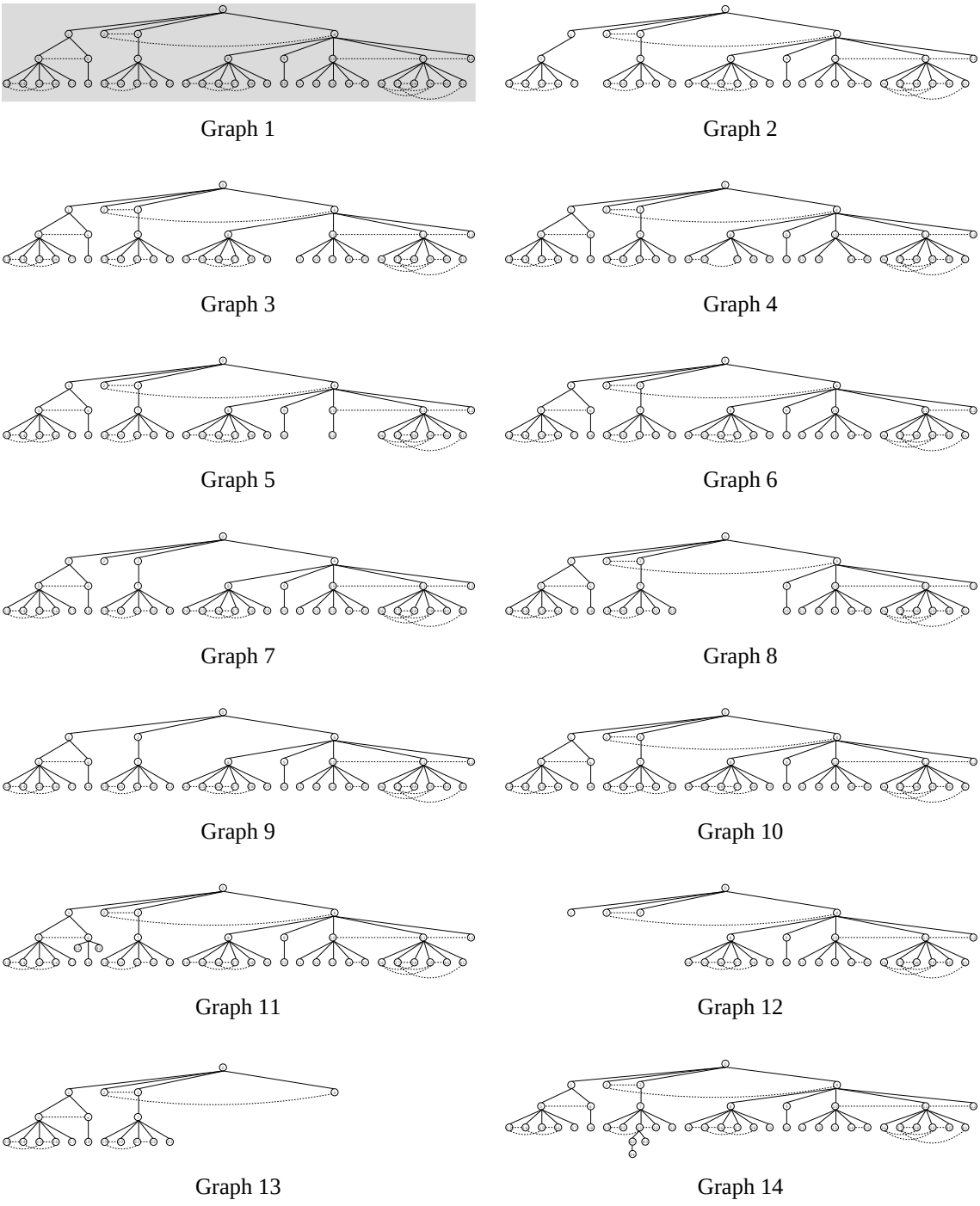


Figure 3.12: Topology graphs used in our experiments.

3.2.4.1 Stability

Some authors presented theoretical demonstrations [Cvetković 97, Shokoufandeh 99] on the stability of graph spectrum. Authors proved that small perturbations on the graph

structure produced small changes on its eigenvalues. Additionally, Sarkar and Boyer [Sarkar 96] presented an experimental study that ascertain the stability and robustness of eigenvalues when the number and weight of links change, while the number of nodes remains the same.

For the graph spectrum to form the basis for stable measures we need to study its sensitivity to the addition or deletion of nodes and links. To that end we performed three experiments. In the first we analyzed the behavior of eigenvalues within a set of ten similar graphs. For the second we used a set of 1,000 graphs generated with some constraints and finally we used a set of 100,000 graphs generated randomly. All graphs used in these studies respected the structure of topology graphs.

10 Similar Graphs

Our first study was made using the set of ten similar graphs mentioned before, namely Graphs 1, 2, 3, 4, 6, 7, 9, 10, 11 and 14, from Figure 3.12. We computed a topological descriptor for each graph using all eigenvalues. We must recall now that each position of the vector descriptor correspond to an eigenvalue and that they are ordered decreasingly. This way, the smallest index positions store the largest eigenvalues, while the largest index positions store the smallest eigenvalues. After, we computed the minimum, maximum and the average for each index position of the descriptors. Obtained results are depicted in Figure 3.13.

From Figure 3.13 we can see that the variation of the eigenvalues is low and tightly bounded. The horizontal axis denote the index of the eigenvalue in the graph descriptor, while the vertical axis presents the normalized values of eigenvalues.

During this study we also observed that the addition (removal) of nodes increases (decreases) the number of eigenvalues and consequently the dimension of the resulting descriptor. On the other hand, the addition (removal) of links only affects the values of the eigenvalues. This observation is also supported by Cvetković et al [Cvetković 97], where they state that any proper subgraph of G has smaller index than G and so $\mu_1(G') < \mu_1(G)$

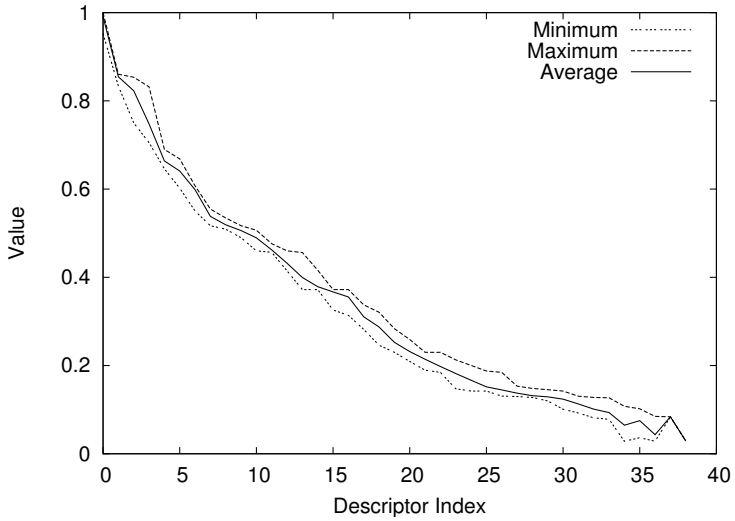


Figure 3.13: Stability analysis for 10 similar graphs.

whenever G' is obtained from G by deleting an edge or a vertex. By the same token, $\mu_1(G'') > \mu_1(G)$ whenever G'' is obtained from G by adding an edge or a vertex.

1,000 Graphs with Constraints

In the second study we generated 1,000 graphs randomly, but imposing some constraints, such as, the number of levels of the graph was always four and the number of nodes in the first level was always five. This way, we got a set of 1,000 different graphs, but with a very similar structure. Resulting graphs had 40 to 200 nodes and correspondent descriptors had dimensions from 40 to 190. Figure 3.14(left) shows the maximum, minimum and average values for each component of the descriptor, considering just positions from 0 to 40. In Figure 3.14(right) we present the same values, but now for all index positions of the descriptor.

From Figure 3.14 we can see that for positions larger than 40, the minimum value is always near zero. This happens because descriptors have different dimensions, and consequently, the last index position has always the smallest eigenvalue. We also observed that this set of eigenvalues have a larger variation than in the previous study, but they still bounded.

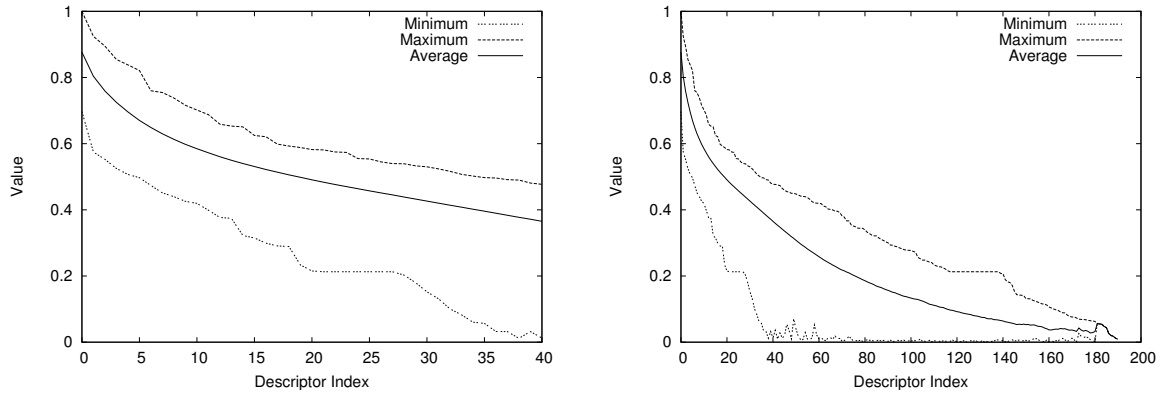


Figure 3.14: Stability analysis for 1,000 graphs with constraints.

100,000 Random Graphs

Our last stability study involved a set of 100,000 graphs randomly generated without any restriction. Resulting graphs had from 2 to 380 nodes, and correspondent descriptors had dimensions from 2 to 330. The main goal of this study was to observe the behavior of the eigenvalues when we have a very large set of heterogeneous graphs. From Figure 3.15 we can see that the variation of the eigenvalues increases, relatively to the previous study, but they still bounded. Moreover, and because we have descriptors with very low dimensions (2) the minimum value of eigenvalues approximates zero for very low indexes.

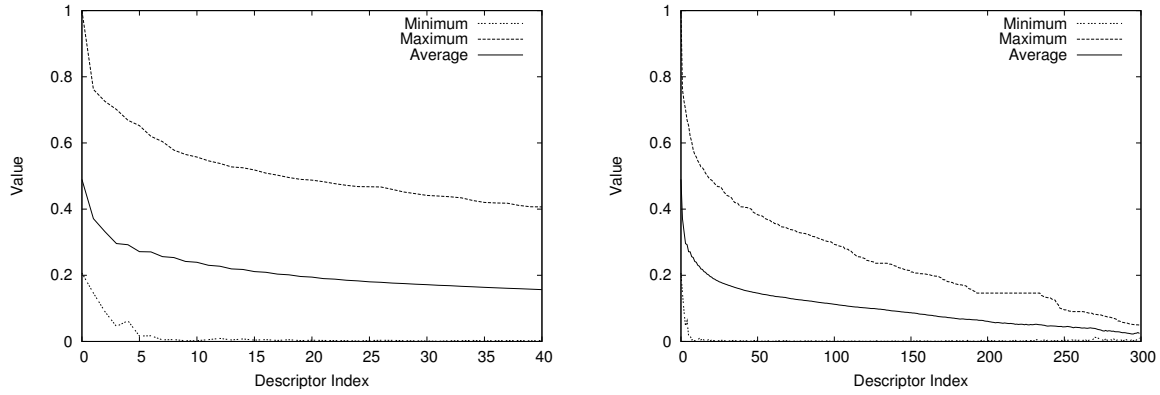


Figure 3.15: Stability analysis for 100,000 random graphs.

As a conclusion, we can say that the use of all eigenvalues as a mechanism to describe topology graphs is stable, since resulting eigenvalues are always within a range of values, even for large sets of very different graphs.

3.2.4.2 Similarity

This second experiment intent to evaluate the accuracy of each method on measuring the similarity between graphs. To that end, we compared our approach, using all eigenvalues, to a set of similarity metrics. We used the metric defined by Bunke and Shearer [Bunke 98], Johnson's metric [Johnson 85], the RASCAL metric [Raymond 02] and an approach based on eigenvalues defined by Shokoufandeh et al [Shokoufandeh 99].

The metric proposed by Bunke and Shearer compute the similarity between two graphs using the maximal common subgraph (MCS) of the two graphs. In their approach they use the relation between the number of nodes in the MCS and the number of nodes in the largest graph, as the similarity measure.

Johnson's metric uses the number of nodes and the number of edges of the maximum common edge subgraph (MCES) and the nodes and edges of the two graphs to be compared. The main disadvantage of this metric and of Bunke metric is the need to compute the MCES and MCS respectively.

The RASCAL metric, developed by Raymond et al, is a novel method to compute an upper-bound on the similarity between a pair of graphs. This approach is lighter computationally than Johnson's metric, because it does not require the calculation of the MCES.

Shokoufandeh's approach is also based on eigenvalues, but they sum these to reduce the dimension of descriptors rather than using eigenvalues by themselves (as we do). This is because efficient indexing structures for high dimensional data points were not used.

In this experiment we used the 14 graphs presented in Figure 3.12. We started our test by computing the difference in the number of nodes and links between Graph 1 and the other graphs from the set. Results are displayed in Table 3.1.

We then computed a measure of similarity between Graph 1 and the other graphs from the set, using each of the different approaches. Table 3.2 presents the achieved results, ordered by similarity, with the most similar on top.

Graph	Δ Nodes	Δ Links	Σ Differences
1	0	0	0
6	0	-1	1
7	0	-2	2
10	0	2	2
3	-2	-2	4
9	-1	-3	4
11	2	2	4
2	-2	-3	5
14	3	3	6
4	-2	-5	7
5	-4	-5	9
8	-7	-12	19
12	-14	-15	29
13	-23	-37	60

Table 3.1: Number of differences between Graph 1 and each graph of the set from Figure 3.12.

From these results, we can see that all approaches, except the sum of eigenvalues, isolate the four least similar graphs. In what concerns the five most similar graphs, all methods identify the same five graphs, except Shokoufandeh's approach (sum of eigenvalues) and Bunke's metric. Finally, we can observe that the RASCAL metric and Johnson's metric present the same result, except for graphs 3 and 9. Furthermore, our approach only have 3 graphs out of order when comparing to the RASCAL metric.

In summary, we can say that the use of all eigenvalues to describe graphs provides a good measure of similarity between graphs.

3.2.4.3 Effectiveness

This last experiment compared the retrieval effectiveness of our approach to the RASCAL metric and Shokoufandeh's method. We chose these two approaches because they are lighter computationally than the other two metrics. Bunke and Johnson's metrics require the computation of the maximum common subgraph, which has a high computational complexity.

Graph Position	Metrics			Eigenvalues	
	Bunke	**Johnson**	**RASCAL**	**All**	**Sum**
1	1	1	1	1	1
2	6	6	6	11	14
3	7	10	10	6	11
4	10	7	7	7	6
5	9	11	11	10	10
6	11	3	9	9	5
7	2	9	3	3	4
8	3	2	2	2	7
9	4	14	14	14	9
10	14	4	4	4	2
11	5	5	5	5	3
12	8	8	8	8	8
13	12	12	12	12	12
14	13	13	13	13	13

Table 3.2: Similarity provided by each approach.

For this test we consider the ten most similar topology graphs from Figure 3.12, namely Graphs 1, 2, 3, 4, 6, 7, 9, 10, 11 and 14. Then, we randomly generated 100,000 topology graphs and added them to the previous ten graphs. After, we computed descriptors for each of the 100,000+ graphs using both eigenvalues methods (this time we did not compute descriptors either by levels of detail or for each subgraph). We inserted the resulting descriptors into two different indexing structures (one for each method). From the set of original graphs we selected five, namely Graph 1 (Q1), 2 (Q2), 3 (Q3), 4 (Q4) and 6 (Q5), to be used as queries. We computed the corresponding descriptors for each query and used these to perform KNN queries ($K = 100$) to both indexing structures, getting the 100 more similar graphs to the given query. For the RASCAL metric we computed the similarity distance between query graphs and each graph from the set. Results were then ordered decreasingly by similarity distance. Table 3.3 summarizes the results for each approach and for each query. Values in the table represent the position in the list of results that the n similar graph appeared. For instance, for the RASCAL approach, the sixth similar graph for query Q1 was in position ten of the results list. Since we only analyzed the first 100 results, the value 150 in the table, means that the expected similar graph was not in the 100 first results.

Similar	All Eigenvalues					Sum of Eigenvalues					Rascal				
Graph	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	40	5	2	2	2	2	2	2
3	3	3	3	3	3	3	3	150	6	3	3	3	3	3	3
4	4	4	4	4	4	4	83	150	7	5	4	4	4	4	4
5	5	6	6	6	5	5	85	150	8	7	5	5	5	21	21
6	6	7	7	23	6	9	88	150	11	8	10	22	7	150	22
7	7	8	8	60	7	34	91	150	22	35	11	26	8	150	25
8	8	16	10	81	8	89	150	150	76	81	32	150	32	150	42
9	9	25	11	150	11	95	150	150	150	150	35	150	150	150	100
10	12	150	150	150	17	150	150	150	150	150	150	150	150	150	150

Table 3.3: Positions in the results list.

To compute the retrieval effectiveness, we used the two standard metrics for measuring the accuracy of individual methods, *recall* and *precision*, defined as follows:

$$Recall = \frac{number\ of\ relevant\ graphs\ retrieved}{total\ number\ of\ all\ relevant\ graphs\ in\ the\ database}$$

$$Precision = \frac{number\ of\ relevant\ graphs\ retrieved}{total\ number\ of\ graphs\ retrieved}$$

We determined the precision value at each point by varying the recall value. To calculate the effectiveness for each approach, we computed the average percentage of precision from the five queries. Results for each method are shown in Table 3.4 and depicted in Figure 3.16. From this figure we can observe that our method has the highest precision performance, while Shokoufandeh's approach has the lowest. Moreover, our approach assures a precision of 100% for recall values up to 40%, *i.e.* for all queries, the

Recall	All Eigenvalues	Sum of Eigenvalues	Rascal
10	100.0	100.0	100.0
20	100.0	69.0	100.0
30	100.0	70.4	100.0
40	100.0	48.9	100.0
50	90.0	48.6	69.5
60	79.5	41.4	40.9
70	77.3	28.4	42.1
80	68.0	10.9	15.9
90	61.1	6.8	10.5
100	32.4	7.4	6.7

Table 3.4: Precision vs Recall.

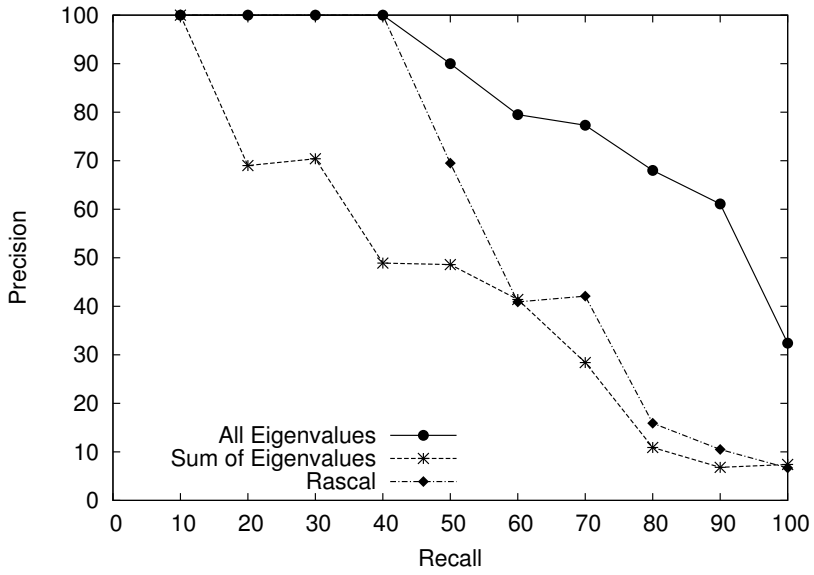


Figure 3.16: Precision and Recall chart.

first four results were always from the set of relevant graphs. Finally, our method offers a precision greater than 60% for recall values up to 90%.

On the other hand, the RASCAL metric outperforms the Shoukfandeh approach,assuring also a precision of 100% for recall values up to 40%. For recall values between 60% and 100%, both methods have similar precisions.

Experimental evidence reveals that using the sum of eigenvalues (Shokoufandeh's approach) yields higher collision frequency than when we use the eigenvalues themselves. Still, it is important to note that the use of all eigenvalues does not assure the unity of descriptors, *i.e.* we can have different graphs with the same descriptor. However there seem to be less collisions than using Shokoufandeh's approach.

3.3 Geometry Representation

While topology convey global information about the drawing, the shape of an object represents local characteristics which can be used to narrow down the search.

To describe the geometry of entities from vector drawings, we developed a general

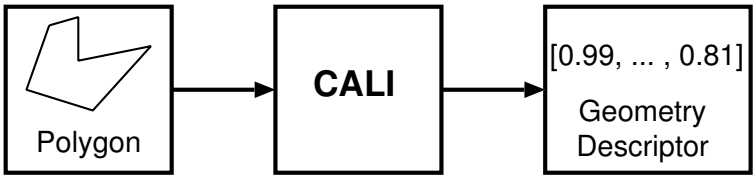


Figure 3.17: Block diagram for computing the geometric descriptor.

shape recognition library which is able to identify a set of geometric figures and gestural commands called CALI [Fonseca 00d, Fonseca 01, Fonseca 02b, Fonseca 00a]. In our approach instead of using CALI to recognize a shape or a gestural command from polygons, we compute a set of geometric features such as area and perimeter ratios from special polygons and store them in a multidimensional vector (see Figure 3.17). Using geometric features instead of polygon classification, allows us to index and store potentially unlimited families of shapes. We obtain a complete description of geometry in a drawing, by applying this method to each geometric entity from the drawing.

3.3.1 Geometric Features and Descriptor Computation

Our geometric description method uses a set of global geometric properties extracted from drawing entities. We start the calculation of geometric features by computing the *Convex Hull* of the provided element, using Graham's scan [O'Rourke 98]. Then, we compute three special polygons from the convex hull: the *Largest Area Triangle* and the *Largest Area Quadrilateral* inscribed in the convex hull [Boyce 85], and finally, the *Smallest Area Enclosing Rectangle* [Freeman 75]. Figure 3.18 depicts the special poly-

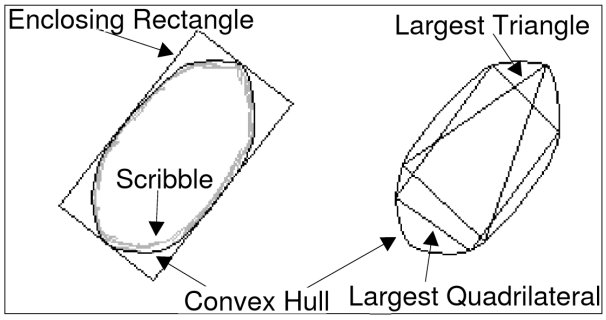


Figure 3.18: Special polygons of a geometric entity.

Feature	Observation
$\frac{Perimeter^2_{ConvexHull}}{Area_{ConvexHull}}$	This feature measures the similarity to a circular shape.
$\frac{Height_{EnclosingRectangle}}{Width_{EnclosingRectangle}}$	The aspect ratio measures the thinness of shapes.
$\frac{Area_{ConvexHull}}{Area_{EnclosingRectangle}}$	This set of features measures the similarity to a rectangle.
$\frac{Perimeter_{ConvexHull}}{Perimeter_{EnclosingRectangle}}$	
$\frac{Area_{LargestQuadrilateral}}{Area_{EnclosingRectangle}}$	
$\frac{Area_{LargestQuadrilateral}}{Area_{ConvexHull}}$	
$\frac{Perimeter_{LargestQuadrilateral}}{Perimeter_{ConvexHull}}$	
$\frac{Area_{LargestTriangle}}{Area_{ConvexHull}}$	This set of features measures the similarity to a triangle.
$\frac{Perimeter_{LargestTriangle}}{Perimeter_{ConvexHull}}$	
$\frac{Area_{LargestTriangle}}{Area_{LargestQuadrilateral}}$	
$\frac{TotalLength_{Shape}}{Perimeter_{ConvexHull}}$	This feature measures the complexity of shapes, *i.e.* it measures the length of strokes within a limited area (convex hull).

Table 3.5: Features used to describe the geometry of objects.

gons computed from a geometric entity.

Finally, we compute the ratios between area and perimeter from each special polygon. We experimentally evaluated several ratios, as described in detail in Appendix C, before we reach the set of features listed in Table 3.5. This set of features allow the description

of shapes independently of their size, rotation, translation or line type. This way, such features can either be used to classify drawings or hand-sketched queries.

3.3.2 Experimental Evaluation

In order to evaluate the retrieval capability (*i.e.* accuracy) of our method, we measured recall and precision performance figures using calibrated test data. Recall is the percentage of similar drawings retrieved with respect to the total number of similar drawings in the database. Conversely, precision is the percentage of similar drawings retrieved with respect to the total number of retrieved drawings.

We compared our method to describe shapes (CALI) with five other approaches, namely Zernike Moments (ZMD), Fourier descriptors (FD), grid-based (GB), Delaunay triangulation (DT) and Touch-point-vertex-angle-sequence (TPVAS). To that end we used results of an experiment previously performed by Safar [Safar 00a], where he compared his method (TPVAS) with the FD, GB and DT methods. In that experiment, authors used a database containing 100 contours of fish shapes[2], as the ones presented in Figure 3.19. From the set of 100 shapes in the database, five were selected randomly as queries. Before measuring the effectiveness of all methods, Safar performed a perception experiment where users had to select (from the database) the ten most similar to each query. This yielded the ten most perceptually similar results that each query should produce.

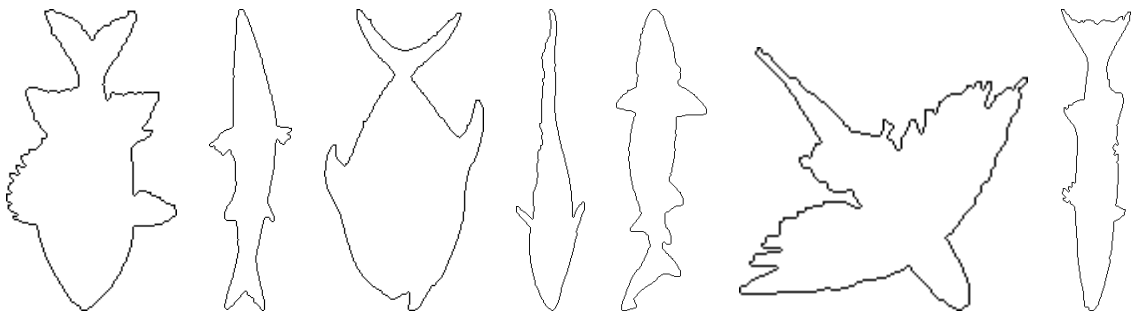


Figure 3.19: Example of objects stored into the test database.

[2]This database is available from ftp://ftp.ee.surrey.ac.uk/pub/vision/misc/fish_contours.tar.Z

We repeated this experiment, using the same database and the same queries, using our method and an implementation of Zernike moments. First we computed descriptors for each of the 100 shapes in the data set and inserted them in our indexing structure (NB-Tree). Then for each query, we computed the correspondent descriptor and used it to perform a nearest-neighbor search in the NB-Tree. Returned results are in decreasing order of similarity to the query. For each of the five queries, we determined the positions for the 10 similar shapes in the order response set. Using results from our method and the values presented in Table 2 from [Safar 00a] we produced the precision-recall plot shown in Figure 3.20.

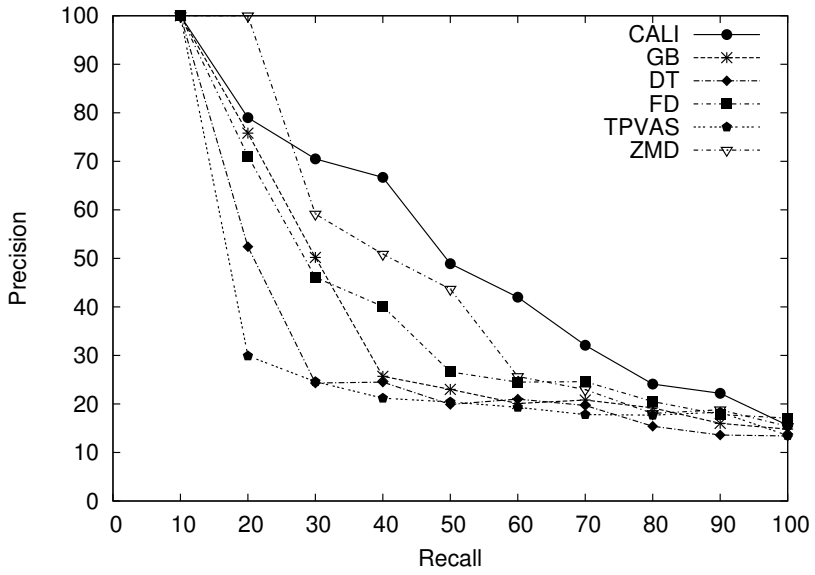


Figure 3.20: Recall-Precision comparison.

Looking at Figure 3.20 we can see that CALI outperforms all the other algorithms, including the Zernike moments, which according to an experimental evaluation [Zhang 03], was considered the best method to describe shape. Furthermore, we can see that it presents good precision for recall values up to 50%. Although, the set of features used by CALI is more suited to classify and describe geometric shapes, we can conclude, from this experimental evaluation, that it can also be used to describe more general shapes, as the contours from this database.

3.4 Matching

Computing the similarity between a hand-sketched query and all drawings in a database can entail prohibitive costs especially when we consider large sets of drawings. However, since drawings and queries are represented symbolically by topology graphs, we can decide whether or not these are similar by determining the similarity of graphs.

Thus, to speed up searching, we divide our matching process in a two-step procedure as shown in Figure 3.21. The first step relies on the global feature extracted from drawings, topology. It searches for topologically similar drawings, working as a first filter to avoid unnecessary geometric matches between false candidates. In the second step we compare the local geometric information from the query to that of drawings that passed the topology step. At the end of the matching process we get a measure of similarity, that combines topology and geometry, between the sketched query and drawings retrieved from the database.

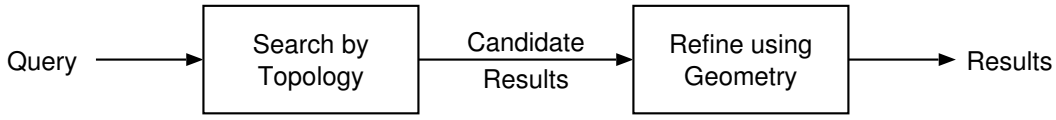


Figure 3.21: Block diagram for the matching process.

3.4.1 Filtering by Topology

Our matching procedure first ranks drawings in the database according to their topological similarity to the query. This is accomplished by performing a KNN query to the topology indexing structure, using the topological descriptor computed from the sketched query. Results returned by the indexing structure represent a set of descriptors similar (near in the space) to the query descriptor. Each returned descriptor correspond to a specific topology graph or subgraph stored in the topology database, which will be used in the geometric matching.

Filtering based on topology drastically reduces the number of drawings to compare

in the geometric matching, by selecting only drawings with a high likelihood of being similar to the sketched query. Each returned result has a degree of similarity that will contribute to the final similarity measure.

Topology plays an important role on the description and filtering of drawings for two reasons. One, topology is a global feature, providing a good mechanism to index drawings. Second, users always explore the spatial arrangement of query elements to convey more information.

3.4.2 Refining by Geometry

The reduced set of drawings resultant from topology filtering are then compared to the sketched query, yielding a measure of geometric similarity for each drawing. The similarity measure is obtained through the computation of distances between query descriptors and geometric data stored in an indexing structure. In our approach each drawing has its own indexing structure with the geometric information, to simplify the matching process. Thus, during polygon comparison we only compute similarity distances between relevant descriptors.

We start the matching procedure by computing a geometric descriptor for each entity specified in the query, using the CALI library. Then, each of these descriptors are used to perform a KNN search (being K the number of polygons in the sketched query) to each geometric indexing structure, one for each topologically similar drawing. Returned results have a distance associated that convey similarity. Smaller distances mean more similar while larger distances mean less similar. Finally, we iteratively select the pair of polygons (one from the query and other from the drawing) that has the smallest distance. After each selection the pair of polygons is eliminated from the list of results.

We will now explain in detail how we compute the geometric similarity measure resorting to an example. Consider that we have a query with four polygons and a geometric indexing structure with five polygons, belonging to a drawing. After performing four 4-

NN queries to the indexing structure, we build a matrix representing distances from each polygon in the query (qP1–qP4) to all polygons in the drawing (dP1–dP5), as the one depicted in Figure 3.22 (top-left).

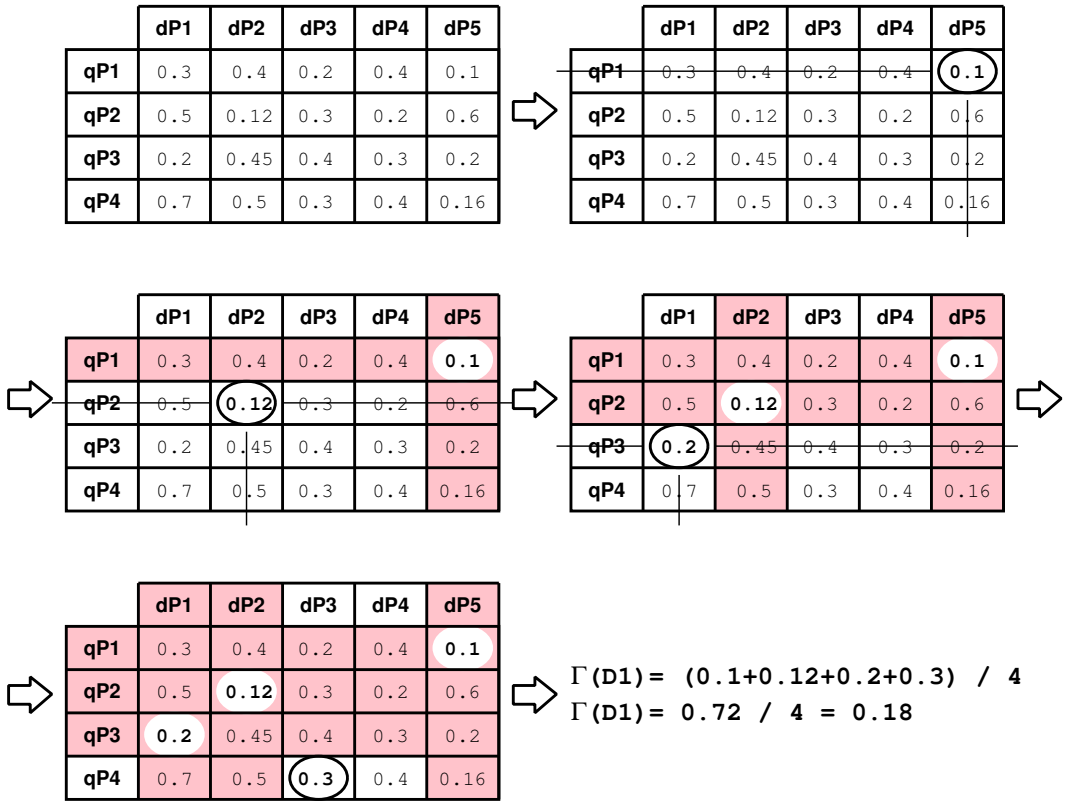


Figure 3.22: Steps to compute the geometric similarity between the sketched query and a drawing.

To compute the similarity value, we first find the row of the matrix with the smallest value (0.1). Then, we store this value and delete the corresponding row (qP1) and column (dP5). From the remaining matrix we perform the same steps, selecting the new smallest value (0.12) and deleting row qP2 and column dP2. We continue this iterative process until we have selected values for all polygons of the query. Finally, we sum all selected values and divide by the number of polygons in the query. When the drawing has less polygons than the query, we compute values for the query polygons most similar to those in the drawing and divide the sum by the number of polygons in the drawing. This process of computing the geometric similarity value is then applied to the remaining drawings that passed through the topology filtering, yielding a list of drawings with a geometric measure

of similarity to the sketched query.

This method does not return the ”optimum” result, since it does not minimize the similarity value. The final result depends on the order by which values are selected. For instance if we select 0.2 in (qP1,dP3) then 0.12 in (qP2, dP2) then 0.16 in (qP4,dP5) and finally 0.2 in (qP3,dP1) we get $\Gamma = (0.2 + 0.12 + 0.16 + 0.2)/4 = 0.17$, which is smaller than the value computed before (0.18). However, the computation of this minimum value is very expensive computationally.

Although, our similarity metric for geometry returns approximate results, its algorithm is simple and has a low computational complexity. Furthermore, we think we achieve a good trade-off between complexity and quality of results, since experimental tests revealed good similarity identification.

To finish the matching process, we combine the similarity measure from topology with the similarity measure from geometry. To that end, we first normalize the similarity values from topology and geometry (separately), dividing all values from each set by the biggest value of the set. The sum of both similarity values represents the measure of similarity between queries and drawings. If for any reason we want to give different weights to topology and geometry, we just need to apply these during the last sum.

3.5 Summary

We have described an integrated solution for indexing and retrieving vector drawings based on their content within large data sets. Our approach describes drawings using topological and geometric information. Furthermore, it avoids graph matching by combining a set of heuristics based on graph spectra and simple algorithms.

First, we presented an overview of our approach, introducing its main components. Classification, which extracts symbolic descriptions from drawings, using topological and geometric information. Query & Interface that allows the specification of queries using sketches. And finally, Indexing that incorporates multidimensional indexing structures to

support efficient searching in large sets of drawings.

After, we described and evaluated our main contributions, namely, the novel multi-level description of drawings that allows searching using different levels of detail. The use of graph spectra to describe and compare graphs, avoiding traditional graph matching algorithms that are NP-Complete. Finally, we presented our method to describe shape using global geometric features extracted from drawings.

Due to the definition of a new approach and new methods, we introduced new concepts, such as the concept of **Topology Graph**, which describes the topological relationships between the elements of a drawing and two topological relationships, **Inclusion** and **Adjacency**, which are a simplified subset of the relationships defined by Max Egenhofer.

To validate the method that we use to compute topological descriptors (the use of all eigenvalues from a graph), we performed a set of experimental tests. First, we checked its stability, showing that small perturbations in the graph structure produced small changes in the descriptor. Second, we compared our method to a set of graph similarity metrics to check its accuracy. Finally, we computed recall and precision measures, comparing our method to a similar one developed by Shokoufandeh (that uses the sum of eigenvalues) and to a new graph similarity metric, RASCAL, which do not require the computation of maximum common subgraphs. Experimental results shown that the use of all eigenvalues to measure graph similarity, presents the highest precision values.

Afterwards, we explained our method to calculate geometric descriptors from drawings using our CALI library. Comparative studies with other shape description techniques revealed that our method outperforms all of them.

Finally, we described our matching mechanism, performed in two steps. One, where we select drawings topologically similar to the sketched query and another where the geometry of these drawings is compared to the query geometry.

4

Multidimensional Indexing with the NB-Tree

A general approach to efficiently support drawing retrieval from large databases has not been developed so far. The majority of current approaches are designed to only handle dozens or at most a few hundreds of images. Therefore, a sequential search of all images into a database will not affect the system's performance significantly. Due to this reason, only a few existing systems feel the need to use an indexing scheme. Our approach takes the size of databases in consideration. Thus, to achieve a fast retrieval speed and make the retrieval system truly scalable to large size drawing collections, an effective multidimensional indexing structure is an indispensable part of the whole system.

Many indexing approaches for high–dimensional data points have evolved into very complex and hard to code algorithms. Sometimes this complexity is not matched by increase in performance. Motivated by these ideas, we take a step back and look at simpler approaches to indexing multimedia data. In this chapter we present a simple, (not simplistic) yet efficient indexing structure for high–dimensional data points of variable dimension, using dimension reduction. Multidimensional points are mapped to a 1D line by computing their Euclidean Norm. In a second step we sort these mapped points using a B^+-Tree on which we perform all subsequent operations. We exploit B^+-Tree efficient sequential search to develop simple, yet performant methods to implement point, range and nearest-neighbor queries.

To evaluate our technique we conducted a set of experiments, using both synthetic and real data. We analyzed creation, insertion and query times as a function of data set size and

dimension. Results so far show that our simple scheme outperforms current approaches, such as the Pyramid Technique, the A-Tree and the SR-Tree, for many data distributions. Moreover, our approach seems to scale better both with growing dimensionality and data set size, while exhibiting low insertion and search times.

We start this chapter by explaining the basic idea of the NB-Tree, its algorithms for insertion and search and the complexity analysis. Section 4.2 describes our experimental evaluation and shows performance results comparing our structure to the best recently published approaches. We conclude the chapter by presenting experimental studies to characterize the different data sets used for evaluation.

4.1 The NB-Tree Structure

Unlike other existing approaches, we use a very simple dimension reduction function to map high-dimensional points into a one-dimensional value. Our approach is motivated by three observations. First, real data sets tend to have a lot of clusters distributed along the data space. Thus, the Euclidean norm of points tend to be ”evenly” distributed (see section 4.3). Second, the set of resulting nearest neighbors will lie inside a narrow range of norm values, thus reducing the number of data points to examine during search. Third, some application domains need to manipulate large amounts of variable dimension data points, which calls for an approach that can handle variable data in a natural manner. Based on these requirements, we developed the NB-Tree[1], an indexing technique based on a simple, yet efficient algorithm to search points in high–dimensional spaces with variable dimension, using dimension reduction. Our approach supports all typical kinds of queries, such as point, range and nearest neighbor queries (K-NN).

The NB-Tree provides a simple and compact means to indexing high-dimensional data points of variable dimension, using a light mapping function that is computationally inexpensive. The basic idea of the NB-Tree is to use the Euclidean norm value as the index

[1] **N**orm + **B**$^+$-Tree = **NB**-Tree

key for high-dimensional points. Thus, values resulting from the dimension reduction can be ordered and later searched in the resulting one-dimensional structure. To index data points sorted by their Euclidean norm we use the B^+-Tree, since it is the most efficient indexing structure for one dimension and also, because it is supported by all commercial Database Management Systems (DBMSs).

The use of the Euclidean norm as a mapping function from $R^D \rightarrow R^1$, assures that near high-dimensional points will have near Euclidean norms. Consequently, when performing a query (of any type) the system only examine points whose norm is in the neighborhood of the query point norm. Moreover, the B^+-Tree has the particularity that its leaves are linked as an ordered list. Thus, walking sequentially through all the elements of the B^+-Tree is a costless operation.

4.1.1 Creation

In the following discussion, we consider that the data space is normalized to the unit hypercube $[0..1]^D$ and that an arbitrary data point in the hyperspace is defined as $P = (p_0, p_1, \cdots, p_{D-1})$.

To create an NB-Tree we start by computing the Euclidean norm of each D-dimensional point from the data set, using the Formula: $\|P\| = \sqrt{p_0^2 + p_1^2 + \cdots + p_{D-1}^2}$. D-dimensional points are then inserted into a B^+-Tree, using the norm as key, as described in the pseudo-code of INSERT-POINT.

```
INSERT-POINT(point)
1  pointNorm ← EUCLIDEAN-NORM(point)
2  INSERT-POINT-BTREE(point, pointNorm)
```

After inserting all points we get a set of D-dimensional data ordered by norm value. Figure 4.1 shows an example of dimension reduction for 2D data points. As we can see, with this mapping function, different points can have the same key value, but on the other hand near data points have similar keys (norms).

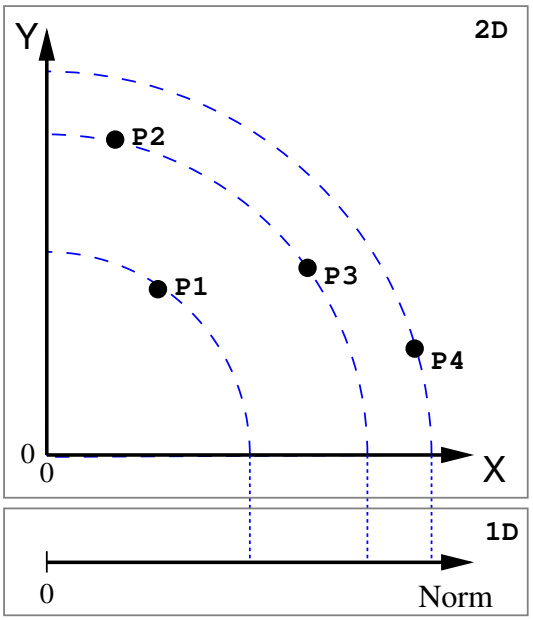


Figure 4.1: Dimension Reduction in 2D.

4.1.2 Searching

The searching process in the NB-Tree starts by computing the norm of the query point. Then we perform a search in the 1–dimensional B^+-Tree, whose steps will depend of the query type. Current indexing structures usually support three types of queries. The first is **Point Query**, which checks if a specific point belongs or not to the database. The second type, **Range Query**, returns the points inside a specific range of values. In our case that range will be specified by an hyper-ball. Finally, the **K-NN Query** (K Nearest Neighbors) returns the K nearest neighbors of the query point. This is the most often-used query in content–based retrieval.

Point Query After computing the Euclidean norm of the query point, we use this value as key to search the B^+-Tree. For the point(s) returned by the B^+-Tree (if any) we compute the distance to the query point. If we get a distance of zero, it means the query point exists in the NB-Tree. The pseudo-code for POINT-QUERY is listed below.

```
POINT-QUERY(query)
 1  queryNorm ← EUCLIDEAN-NORM(query)
 2  point ← SEARCH-POINT-IN-BTREE(queryNorm)
 3  while point ≠ NIL
 4  do
 5     distance ← DISTANCE-TO-QUERY(point, query)
 6     if distance = 0
 7       then
 8            return TRUE
 9     point ← NEXT-POINT-FROM-BTREE
10  return FALSE
```

Range Query The next step for a range query, after computing the query norm, is to compute the lower and the higher bounds on the norm for the corresponding query. $lowerLimit = \|Q\| - bRadius$ and $higherLimit = \|Q\| + bRadius$, where $bRadius$ is the radius of the hyper–ball, which spans the search range.

After that, we search the B$^+$-Tree for the point with $lowerLimit$ as key. After finding the first point we sequentially scan the B$^+$-Tree until we reach the $higherLimit$. As we said before, scanning the B$^+$-Tree sequentially is an inexpensive operation. For each point

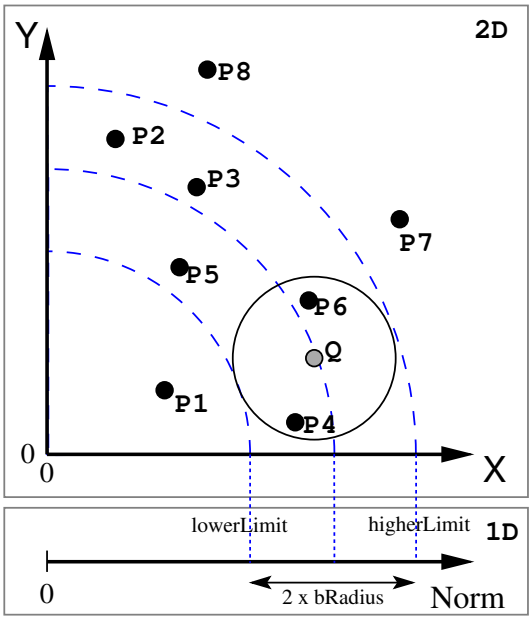


Figure 4.2: 2D Range query example.

with a norm value in the interval $[lowerLimit, higherLimit]$, we compute the distance to the query point. If this distance is smaller than or equal to the $bRadius$, the point is added to the list of points to be returned. This list is ordered by distance to the query point. Figure 4.2 illustrates a ball query in 2D, where we can see that an interval in the 1D space corresponds to part of a D–dimensional hypersphere. The next pseudo-code implements the BALL-QUERY algorithm.

```
BALL-QUERY(query, bRadius)
 1  queryNorm ← EUCLIDEAN-NORM(query)
 2  lowerLimit ← queryNorm − bRadius
 3  higherLimit ← queryNorm + bRadius
 4  point ← SEARCH-POINT-IN-BTREE(lowerLimit)
 5  while pointNorm ≤ higherLimit
 6  do
 7     distance ← DISTANCE-TO-QUERY(point, query)
 8     if distance ≤ bRadius
 9       then
10            ADD-POINT(pointList, point)
11     point ← NEXT-POINT-FROM-BTREE
12  return pointList
```

K-NN Query Before we start describing our K-NN algorithm we have to define some relevant concepts: **higherLimit** and **lowerLimit** are the norm values used to define the upper and lower bounds to search for nearest neighbors; **delta** - value by which the previous limits are incremented at each step; **priority list** - list with K nodes, where we store K points that temporarily store for intermediate results. The last element of this list is the farthest nearest neighbor so far. Thus, when we want to insert a new neighbor into the list, we just have to compare its distance against the last.

We start the nearest neighbor search by locating a point in the B^+-Tree with a key equal (or near) the norm of the query point, as described in the KNN-QUERY pseudo-code. After this positioning in the 1D line, we execute a set of iterative steps until we get all the desired results. Since the B^+-Tree has its leaves linked sequentially, we only navigate at the leaves level, as if we were using a double linked ordered list. Below we

describe all the steps to achieve the final K-NN points.

```
KNN-QUERY(query, knn)
 1  queryNorm ← EUCLIDEAN-NORM(query)
 2  lowerLimit ← queryNorm
 3  higherLimit ← queryNorm
 4  repeat
 5        point ← SEARCH-POINT-IN-BTREE(higherLimit)
 6        higherLimit ← higherLimit + delta
 7        while pointNorm ≤ higherLimit
 8        do
 9           distance ← DISTANCE-TO-QUERY(point, query)
10           if distance < farthest
11             then
12                  DELETE-LAST-POINT(pointList)
13                  ADD-POINT(pointList, point)
14           point ← NEXT-POINT-FROM-BTREE
15        point ← SEARCH-POINT-IN-BTREE(lowerLimit)
16        lowerLimit ← lowerLimit − delta
17        while pointNorm ≥ lowerLimit
18        do
19           distance ← DISTANCE-TO-QUERY(point, query)
20           if distance < farthest
21             then
22                  DELETE-LAST-POINT(pointList)
23                  ADD-POINT(pointList, point)
24           point ← PREVIOUS-POINT-FROM-BTREE
25    until ENOUGH-POINTS(pointList, knn)
26  return pointList
```

After positioning in the 1D line we define the $upperLimit$ and the $lowerLimit$ based on the $delta$ value. Next we go through all points until we reach the $upperLimit$ or the point whose norm follows the limit. In this case the $lowerLimit$ is changed to keep the symmetry to the query. After, we do the same for the $lowerLimit$. During this forward and backward advance we compute the distance from each point to the query point. As described before, if the distance to the query point is smaller than the distance of the farthest current near neighbor, we store that point in the list. After going up and down, we check whether there are enough points inside the hypersphere of diameter equal to the difference between limits. This iterative process happens until all K neighbors from the

list are contained in the previously defined hypersphere.

In summary, we compute the K-NN through an iterative process, where the size of the searching ball increases gradually until we get all the points specified by the query.

4.1.3 Computational Complexity

In this subsection, we present the computational complexity of the NB-Tree insertion, point, range and K-NN query algorithms. We describe the running time of the algorithms as a function of data point dimension and data set size. It is important for our complexity study to recall that the B$^+$-Tree has a computational complexity of $O(lgN)$ for insertion and searching algorithms and requires linear space for storage.

Insertion From the pseudo-code for INSERT-POINT we conclude that inserting points into a NB-Tree requires two operations: the computation of the Euclidean Norm and an insertion into the B$^+$-Tree. Table 4.1 presents the running time for each operation as a function of data set size (N) and data point dimension (D).

Operation	N	D
Computation of the Norm	$O(1)$	$O(D)$
Insertion into the B$^+$-Tree	$O(lgN)$	$O(1)$
Total Running Time	$O(lgN)$	$O(D)$

Table 4.1: Analysis of the NB-Tree Insertion algorithm.

Point Query As described previously the main operations of a point query are the computation of the Euclidean Norm, a search in the B$^+$-Tree and the calculation of a distance. Table 4.2 shows running times for the point query algorithm.

Range Query From the pseudo-code for BALL-QUERY, we can identify the computation of the Euclidean Norm, a search in the B$^+$-Tree, the calculation of N distances (in the worst case), the insertion of M points into a B$^+$-Tree (the ones inside the ball, $M << N$)

Operation	N	D
Computation of the Norm	$O(1)$	$O(D)$
Search in the B$^+$-Tree	$O(lgN)$	$O(1)$
Calculation of a distance	$O(1)$	$O(D)$
Total Running Time	$O(lgN)$	$O(D)$

Table 4.2: Analysis of the NB-Tree Point Query algorithm.

and N sequential steps in the B$^+$-Tree (in the worst case), as main operations. Therefore, the total running time for the range query algorithm is presented in Table 4.3.

Operation	N	D
Computation of the Norm	$O(1)$	$O(D)$
Search in the B$^+$-Tree	$O(lgN)$	$O(1)$
N x Calculation of a distance	$O(1)$	$O(D)$
M x insertion into a B$^+$-Tree	$O(lgM)$	$O(1)$
N x sequential step	$O(N)$	$O(1)$
Total Running Time	$O(N)$	$O(D)$

Table 4.3: Analysis of the NB-Tree Range Query algorithm.

K-NN Query From the pseudo-code presented KNN-QUERY we can extract the following operations: computation of the Euclidean Norm, a search in the B$^+$-Tree, the calculation of N distances (in the worst case), the insertion of M points into a list ($M << N$) and N sequential steps in the B$^+$-Tree (in the worst case). Table 4.4 shows the running times for the K-NN algorithm.

Operation	N	D
Computation of the Norm	$O(1)$	$O(D)$
Search in the B$^+$-Tree	$O(lgN)$	$O(1)$
N x Calculation of a distance	$O(1)$	$O(D)$
M x insertion into a list	$O(lgM)$	$O(1)$
N x sequential step	$O(N)$	$O(1)$
Total Running Time	$O(N)$	$O(D)$

Table 4.4: Analysis of the NB-Tree K-NN Query algorithm.

In Table 4.5 we present a summary of the complexity analysis. As we can see, the

NB-Tree presents a logarithmic running time for insertion and point query and a linear running time for range query and K-NN query, when we consider the size of the data set. If we now consider the dimension, the NB-Tree has a worst-case running time linear with the dimension.

NB-Tree Algorithm	N	D
Insertion	$O(lgN)$	$O(D)$
Point Query	$O(lgN)$	$O(D)$
Range Query	$O(N)$	$O(D)$
K-NN Query	$O(N)$	$O(D)$

Table 4.5: Summary of the NB-Tree complexity analysis.

4.2 Experimental Evaluation

While our method seems to yield commensurable times to other structures tested, we had difficulties comparing it to other approaches, since some of them crashed on data sets of significant size, preventing comparison. We chose the SR-Tree, the A-Tree and the Pyramid Technique as benchmarks because they are the more recent indexing structures and because there are reliable and stable implementations, which provide correct results and scale up to our intended test data sizes.

In this section we describe the experimental evaluation performed to compare our NB-Tree with the SR-Tree, the A-Tree and the Pyramid Technique. We conducted a set of experiments to analyze final tree size (on disk), creation times and query times as a function of data set dimension and size. All experiments were performed on a PC Pentium II @ 233 MHz running Linux 2.4.8, with 384 MB of RAM and 15GB of disk. Our NB-Tree algorithms were developed using the *dbm* implementation of the B^+-Tree.

First, we present the results of the evaluation for several synthetic data sets of uniformly distributed data. Then, we introduce our observations for some real data sets and for data sets of variable dimension.

We did not measure I/O accesses for four reasons: First, nowadays computers can

handle databases of millions of high–dimensional points in main memory. Second, after a dozen queries all the indexing structure is loaded into main memory. Third, users of content-based retrieval systems do not care about the number of I/O operations, but they do care about response time. Fourth, in many indexing schemes, data structure overhead dominates I/O, especially after most of the database is memory–resident. Finally, we assume that locality of reference holds for well–behaved features.

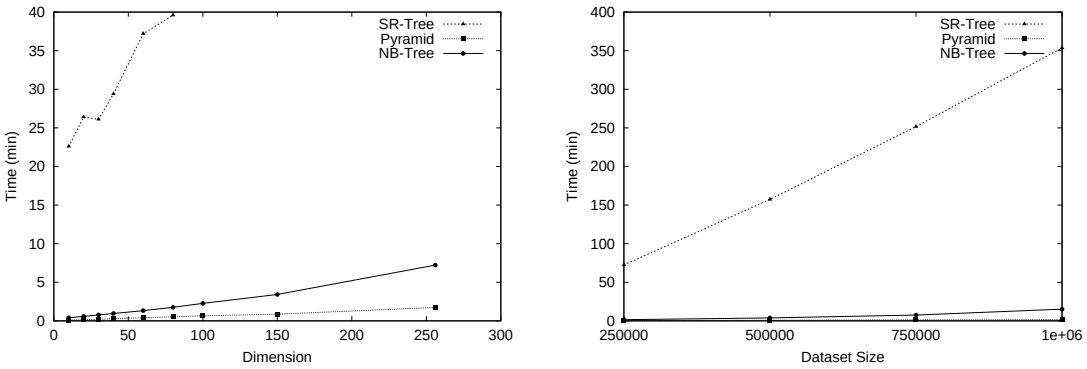
Thus, during our experimental evaluation we start by loading all the structures into main memory and after that we measure search times. We think that the time taken to load the NB-Tree into main memory is not significant, taking into account that it is done just one time. The NB-Tree takes less than one minute to load, for dimensions up to 100 and less than 4 minutes for dimensions up to 256.

4.2.1 Performance Measures with Uniformly Distributed Data

While data points generated from a uniformly distributed set of coordinates arguably represent any ”real” distribution, most published approaches provide performance measurements and comparisons based on such data. Thus it becomes logical to evaluate our indexing structure using these data sets. Moreover, as we will show in the present section, many algorithms (including ours) seem to perform poorly with these data, which makes better suited for the task at hand than most empirical data sets.

We evaluated the structures using data sets of randomly generated uniformly distributed data points of fixed size (100,000) and various dimensions (10, 20, 30, 40, 60, 80, 100, 150 and 256). We also created data sets with fixed dimension (20) and various sizes (250,000, 500,000, 750,000 and 1,000,000). Additionally, we randomly generated a set of 100 queries for each dimension, which we later used to evaluate the searching performance of each approach.

We selected the number of nearest neighbors to search for to be always ten, while the radius (width) of the ball (rectangle) was different for each data set depending on the



a) Data set size of 100,000 points b) Data points of dimension 20

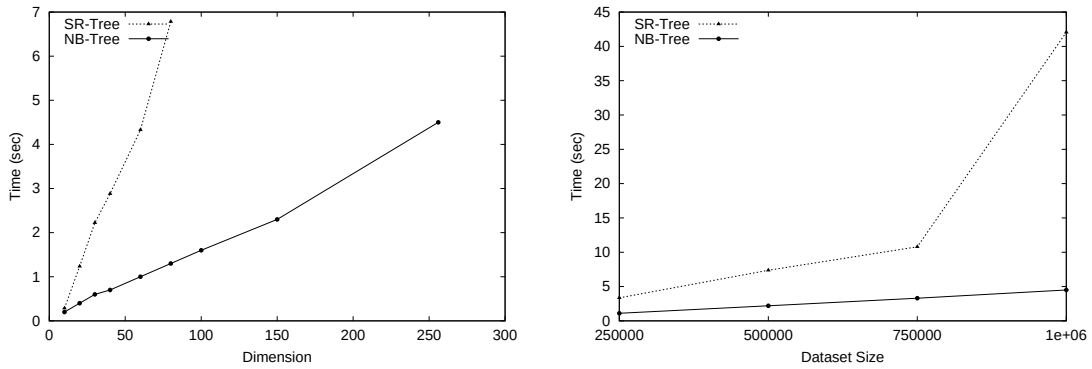
Figure 4.3: Creation times as a function of a) dimension. b) data set size.

dimension of the data points. This variation in radius is due to increasing average distance between points when the dimension increases.

In the following paragraphs we present the results of our experimental evaluation organized by creation, searching and final tree size.

Creation Times Most published work tends to ignore tree insertion times. This is because conventional scenarios focus on large static databases which are far more often queried upon than updated. However, there are many applications requiring frequent updating of data sets. For these applications, low insertion times are an important usability factor.

We have compared the creation times to these of the SR-Tree, A-Tree and Pyramid Technique. Figure 4.3.a shows the time spent to create each structure when the dimension of the data changes. We do not present the creation times for the A-Tree because they are very high, even for low dimensions (around 300 minutes for dimensions 10 and 20). As we can see, the NB-Tree and the Pyramid Technique largely outperform the SR-Tree. While the NB-Tree takes 24 seconds to insert 100,000 points of dimension 10, the SR-Tree takes 23 minutes. If we now consider higher dimensions, such as 80, the difference increases even more with the NB-Tree taking 2 minutes and the SR-Tree taking 40 minutes. The Pyramid Technique reveals creation times below the two minutes, for



a) Data set size of 100,000 points b) Data points of dimension 20

Figure 4.4: Range Query searching times as a function of a) dimension. b) data set size.

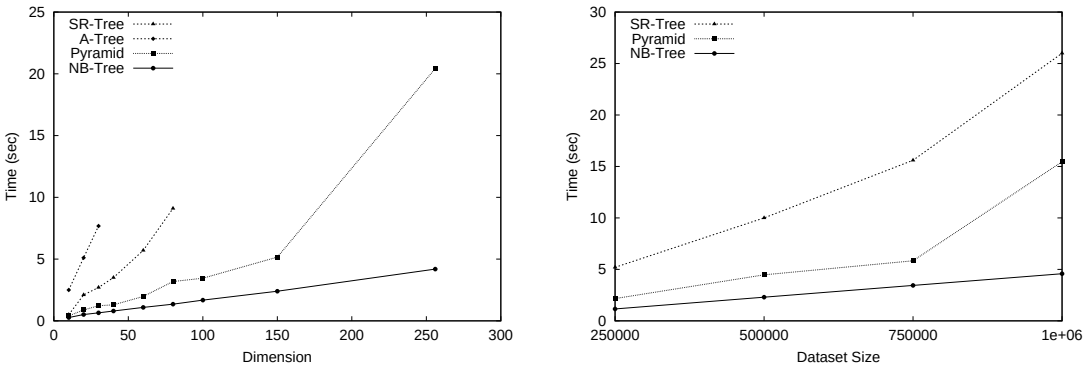
dimensions up to 256.

Figure 4.3.b shows insertion times for data sets of varying size. Although, all structures seem to exhibit linear growth with dimension, SR-Tree creation times grow faster than those of the NB-Tree and of the Pyramid Technique. While the NB-Tree requires no more than 15 minutes to create a tree with one million data points, the SR-Tree takes around six hours and the Pyramid Technique takes only two minutes. From this observation it is clear that the Pyramid Technique and the NB-Tree are more suited to large data sets than the SR-Tree.

We were not able to create the SR-Tree for data sets of dimension bigger than 100, in our system. Thus, in the following subsections we do not display values for dimensions higher than 100 corresponding to the SR-Tree.

Range Query Times Another useful primitive in multidimensional data is the range query, where we want to look for data points within a specified distance of the target.

Figure 4.4.a shows performance for both trees as a function of the dimension. Once again the NB-Tree largely outperforms the SR-Tree for all dimensions. While our approach exhibits searching times smaller than one second for dimensions up to 60, and smaller than 2 seconds for dimensions up to 100, the SR-Tree needs more than six sec-



a) Data set size of 100,000 points b) Data points of dimension 20

Figure 4.5: Search times for K-NN as a function of a) dimension. b) data set size.

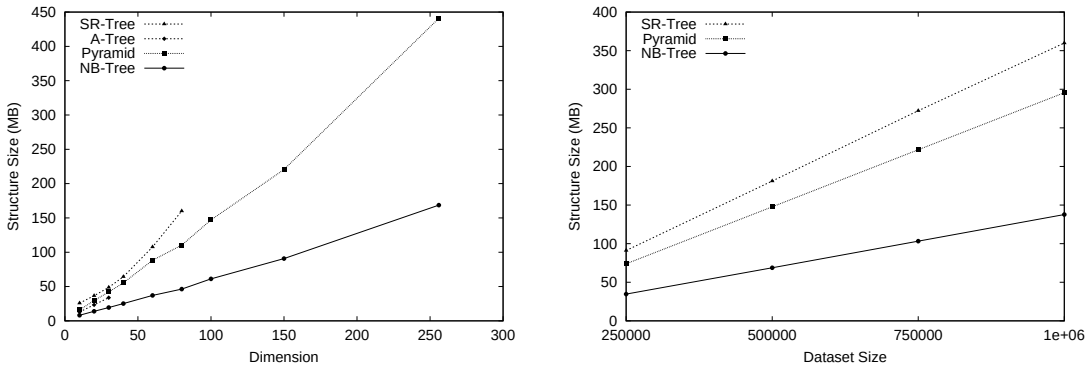
onds. Additionally we can see that the NB-Tree has a linear growth with the dimension while the SR-Tree seems to grow quadratically.

Figure 4.4.b plots times required for range queries with varying data set sizes. The NB-Tree outperforms the SR-Tree for all data set sizes. The NB-Tree takes only five seconds to perform a range query in a 1 million point data set, while the SR-Tree spends around 40 minutes to yield the same results.

We do not present results for the A-Tree, because the available implementation of the A-Tree does not support range queries. While we have tried hard to evaluate the range query algorithm of the Pyramid Technique we have been unable to reproduce results using the currently available implementation.

K-NN Query Times Nearest-neighbor queries are useful when we want to look at the point in the data set which most closely matches the query.

Figure 4.5.a depicts the performance of nearest neighbor searches when data dimension increases. We can see that the NB-Tree outperforms all the structures evaluated, for any characteristic dimension of the data set. Our approach computes the ten nearest neighbors in less than one second for dimensions up to 40, less than two seconds for dimensions up to 100 and less than five seconds for dimensions up to 256. Moreover, we can



a) Data set size of 100,000 points b) Data points of dimension 20

Figure 4.6: Final tree size as a function of a) dimension. b) data set size.

notice that the NB-Tree shows linear behavior with the dimension while the SR-tree and the A-Tree seems to exhibit at least a quadratic growth or worse. The Pyramid Technique performs better than the A-Tree and the SR-Tree, but it shows worst search times than the NB-Tree, for all dimensions. Figure 4.5.b also shows that the NB-Tree outperforms all structures for K-NN queries when the size of the data set increases.

Storage Requirements Looking at Figure 4.6 we can realize that the NB-Tree requires less storage space than any other structure. The SR-Tree uses at least three times more storage space than the NB-Tree, while the Pyramid Technique requires more than twice the storage space used by our structure. The A-Tree seems to use a storage space of the same size of the Pyramid Technique. Furthermore, the SR-Tree size seems to increase quadratically with the data dimension while the NB-Tree and the Pyramid Technique grow linearly. This linear growth is inherited from the B^+-Tree. We can also see that all structures present a linear growth with data set size (c.f. Figure 4.6.b). However, the storage requirements from our approach grow slower than those from the SR-Tree and the Pyramid Technique. Even though, the NB-Tree and the Pyramid Technique share a B^+-Tree to store information and even though they store the same information (high–dimensional point + 1D value) the files produced by the Pyramid Technique are around twice the size of the ones from the NB-Tree. The only explanation that we encounter is the

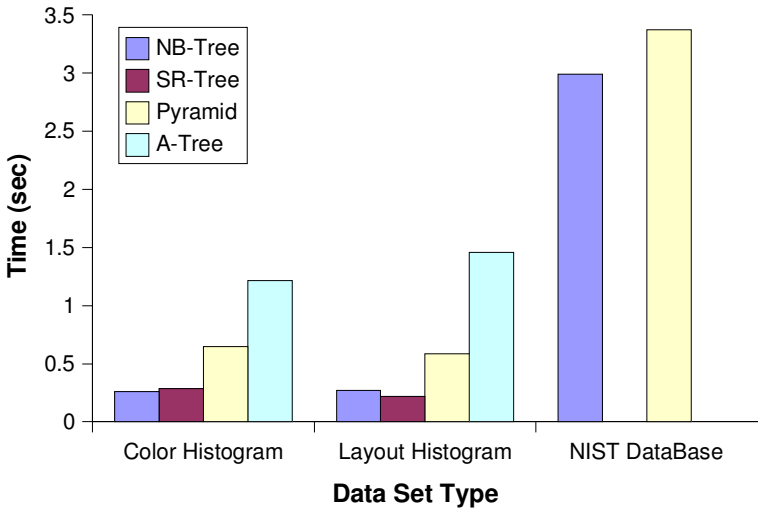


Figure 4.7: K-NN searching times for real data sets.

different implementations used. We used the `dbm` version, while the Pyramid Technique uses their own.

4.2.2 Performance Measures with Empirical Data Sets

After our experimental evaluation with synthetic uniformly distributed data sets, we evaluate the indexing structures using three data sets from real data. Two of them had dimension 32 and contain image features extracted from Corel image collection. One contains 68,000 color histograms, while the other contains 66,000 color histogram layouts. The last real data set, from the NIST database, has 100,000 points of dimension 256 and each one represents an instance of a hand-drawn digit.

Since the K-NN search is the most often-used query in content-based retrieval, we just present experimental values for this kind of query.

Figure 4.7 presents times for real data sets. Looking at the two data sets from the left (Histograms) we can see that the NB-Tree and the SR-Tree have the best performance, with query times around 270 milliseconds. We can also notice from Figure 4.7 that the performance of the SR-Tree is very influenced by the size of the data set. For a data set of 66,000 points (Layout Histogram), the SR-Tree outperforms the NB-Tree, but when we

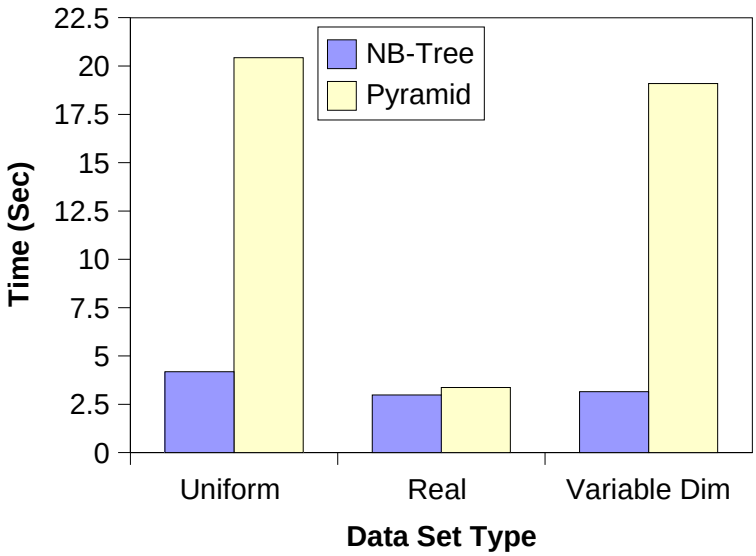


Figure 4.8: K-NN searching times for various data sets of dimension 256.

move to a data set with more 2,000 points (Color Histogram), the SR-Tree decreases its performance, while the NB-Tree takes about the same time.

The A-Tree once again presents the worst results, taking five times longer than our approach. Considering now the data set from the NIST database, with dimension 256, we can see that the NB-Tree and the Pyramid Technique have very similar times, but with some advantage to the NB-Tree. We do not present values for the A-Tree and the SR-Tree, because these structures can not handle data sets of this dimension (256).

Contrary to published experimental values [Sakurai 00], we have found better query times for the SR-Tree over the A-Tree. Maybe a fine tuning would yield a different outcome. This did not succeed for the many combinations we have tried.

4.2.3 Performance Measures with Variable Dimension Data Sets

Finally, we evaluated the indexing structures with other synthetic data set that try to simulate data points of variable dimensions. To that end, we filled the first N coordinates of points with values and the rest ($D-N$) with zeros. We generate points with dimensions between 4 and 256 (but all points have 256 coordinates). We had to do this, because the

other structures do not support points of multiple dimensions at the same time.

From Figure 4.8 (right) we can see that the NB-Tree is six times faster than the Pyramid Technique finding the ten nearest neighbors.

Figure 4.8 presents results for the NB-Tree and the Pyramid Technique using three data sets, uniformly distributed data points (left), a real data set from the NIST database (center) and a synthetic data set to simulate data points of variable dimension (right). From this figure we can see that the NB-Tree is always faster than the Pyramid Technique for any kind of data set with dimension 256. Moreover, Figure 4.8 shows that the NB-Tree presents the best results when we consider real data sets or data sets that simulate the context where we are using our indexing structure (data points with variable dimension).

Finally, we evaluated our indexing structure in two more experiments where we tried to simulate our domain of application. To that end, instead of generating descriptors, we generated topology graphs and then we computed the corresponding descriptors. We used the 100,000 topology graphs described in Section 3.2.4.1. Then, we computed descriptors for those graphs using our multilevel scheme (*i.e.* we compute descriptors for subgraphs and for different levels of detail) and not using it (*i.e.* one descriptor for each topology graph). Table 4.6 summarizes the results for KNN queries, with $K = 10$, for the NB-Tree and the Pyramid Technique (only for 100,000 descriptors). Times presented in the table are average figures obtained from performing 100 KNN queries.

Descriptor Type	Nº of Descriptors	Max. Dimension	NB-Tree	Pyramid Tech.
Without levels	100,000	330	0.04 Sec	0.25 sec
With levels	1,524,000	330	1.15 Sec	—

Table 4.6: Search times for descriptors generated from 100,000 topology graphs.

As a conclusion, and looking at Table 4.6, we can say that our indexing structure presents low searching times for very large data sets containing information from our domain of application, *i.e.* descriptors computed from topology graphs.

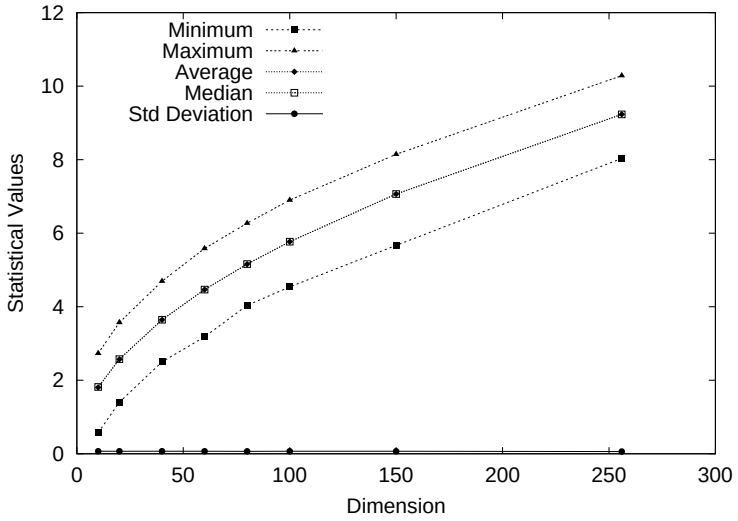


Figure 4.9: Statistical values for the Euclidean norm.

4.3 Data Set Characterization

To validate the approach used in our NB-Tree, we performed an experimental study to characterize the different data distributions that we used during the experimental evaluation of the different indexing structures. The first data sets analyzed were of uniformly distributed data, with independent attributes. After, we analyzed the real data and finally the synthetic data set simulating a data space of variable dimension. We assume that the data space, for all kinds of data sets, is normalized to the unit hypercube $[0..1]^D$.

We start by analyzing the distribution of Euclidean norms of data points thus generated to illustrate relevant artifacts characteristic of high-dimensional data spaces to show how the growing dimensionality affects performance of our data structure.

4.3.1 Uniform Data Distribution

We start our data analysis by computing statistical moments for the Euclidean norm of data points. We calculate the minimum, the maximum, the average, the median and the standard deviation for data sets of dimension 10, 20, 40, 60, 80, 100, 150 and 256. Figure 4.9 shows the results we got. As we can see, the average and the median have

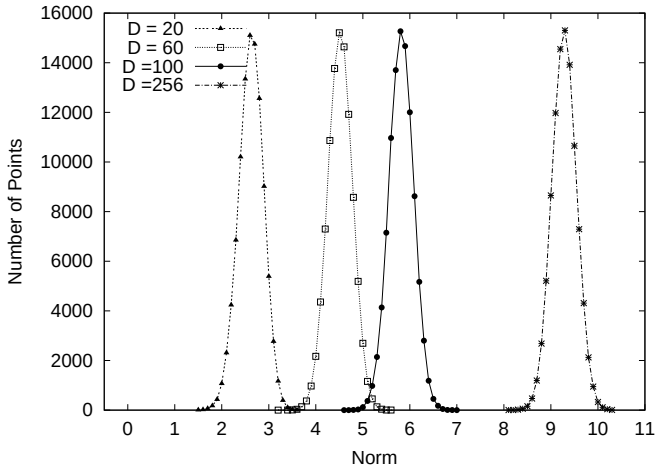


Figure 4.10: Distribution of norms for several dimensions.

the same value for all dimensions, meaning that we have a normal distribution for the values of the Euclidean norm. We can also observe that the standard deviation as a value near zero and that it remains the same for all dimensions. The amplitude ($maximum - minimum$) of the norm values is always constant for all dimensions. Finally, we can see that the minimum, the maximum, the average and the median present the same evolution along the dimension.

Our next step was to study the distribution of points according to the Euclidean norm. To that end we compute the mode of the norm for each dimension, using intervals of 0.1. Figure 4.10 presents the point distribution for dimensions 20, 60, 100 and 256. As we can see points have the same distribution (normal) for all dimensions.

4.3.2 Other Distributions

Besides the uniformly distributed data, we also analyzed a data set of real data from the NIST database. Data points have a dimension of 256 and coordinate values can each take the value zero or one. Finally, to simulate an application domain where the data points have different dimensions, we generated synthetic data with variable dimensions and analyzed it.

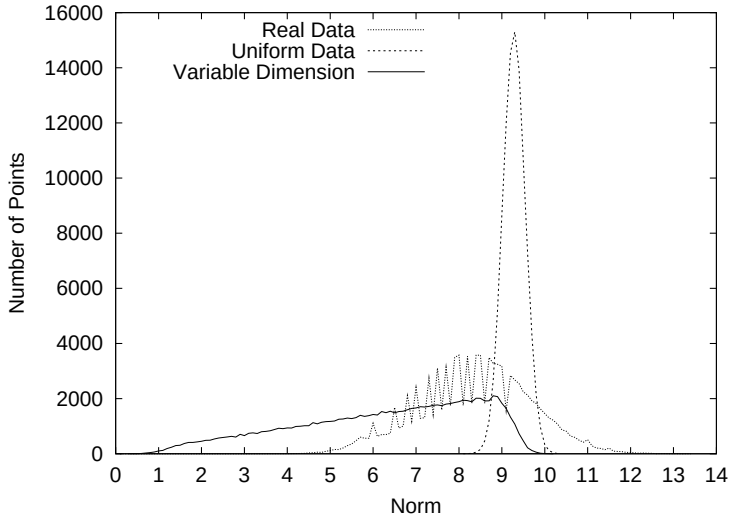


Figure 4.11: Distribution of norms for different types of data sets.

The main objective of this study was to achieve some conclusions about the behavior of the Euclidean norm when we move from uniformly distributed data to real data and see what kind of data sets yield worst performance for the NB-Tree approach. To that end, we computed the distribution of points per norm interval and present it in Figure 4.11.

As we can see, for uniformly distributed data the values of the norm form a ”tight” cluster around a single value. In the case of the real data and of points with variable dimensions, the values of the norm are more evenly distributed. This wide distribution of the norm values seems to validate the NB-Tree approach of using the Euclidean norm as a method to index high-dimensional real data sets of fixed or variable dimensions.

4.4 Summary

In this chapter we presented a simple, yet efficient structure for data of highly–variable dimensionality, the NB-Tree, using the Euclidean norm to reduce dimension and a B^+-Tree to store the 1D values. We described simple algorithms for insertion, point, range and nearest neighbor queries.

We have conducted experimental evaluations using either uniformly distributed data

sets, variable dimension and real data. Results show our method to outperform the Pyramid Technique[2], the SR-Tree and the A-Tree. Difference in performance is expected to increase as data dimensionality and data set size increase, and for real (skewed) data points.

Our experimental evaluation indicates that the NB-Tree can efficiently support a wide type of queries, including point, range and the most used (at least in multimedia databases) nearest neighbor queries. This is significant because most indexing structures proposed for range queries are not designed to efficiently support similarity search queries (K-NN) and metric-based approaches proposed for similarity queries are considerably more difficult to apply to range queries. In contrast to other approaches which use complex algorithms (combination of MBRs and MBSs to describe partitions, division of the space using hyper-pyramids, etc.) ours relies on very simple (and therefore practical) methods.

Tree-based techniques do not seem to scale up well with growing data set sizes typical of multimedia databases, where it is relatively easy to assemble large collections of documents. The overhead introduced by data-structure traversal and book keeping in these approaches far outweighs the potential advantages of sophisticated spatial indexing structures.

For a seemingly ”bad” point set, generated from uniformly distributed data, both the range and K-NN search algorithms can degenerate into a sequential search, where we have to compute the distance from the query point to all the points in the data set. However, as we have seen in section 4.3 real data and data of variable dimension both exhibit Euclidean norm distributions that better match our selection and indexing methods. We conjecture that our approach tends to excel for sparsely–distributed data where clusters tend to be evenly distributed throughout the hyperspace. Moreover, since our algorithms are very simple, response times remain reasonable for data sets of considerable size and dimension.

[2]K-NN queries only. We were not able to reproduce results in range queries.

5

Applications and Experimental Results

This chapter describes two prototypes developed using our approach. One is for retrieving technical drawings of moulds. This application has been developed within the European project SmartSketches[Consortium 00]. The other helps users in retrieving clip-art drawings. We selected these two types of media objects to search for, because both are well structured figures and are complex enough to represent a good testbed for our approach. On the other hand, technical drawings are characterized by very rich topological information. Furthermore, their visual elements are mostly basic geometric shapes. In contrast, clip-art drawings present more generic visual entities and a poorer spatial organization. In the remainder of this chapter we define the goals behind each prototype and shortly describe their functionalities. Finally, we present results from preliminary usability tests.

5.1 Sketch-Based Retrieval of Moulds

Draftspeople from the mould industry while drawing a mould for a new object usually consult previous projects, searching for something similar. This way, they can use solutions already achieved in the past, reducing the time needed to produce a new mould.

We developed a Sketch-Based Retrieval (SBR) system [Fonseca 04d] able to find a small set of drawings in a large technical drawing database through a hand-sketched query. To evaluate the current status of our SBR prototype and the performance of its algorithms,

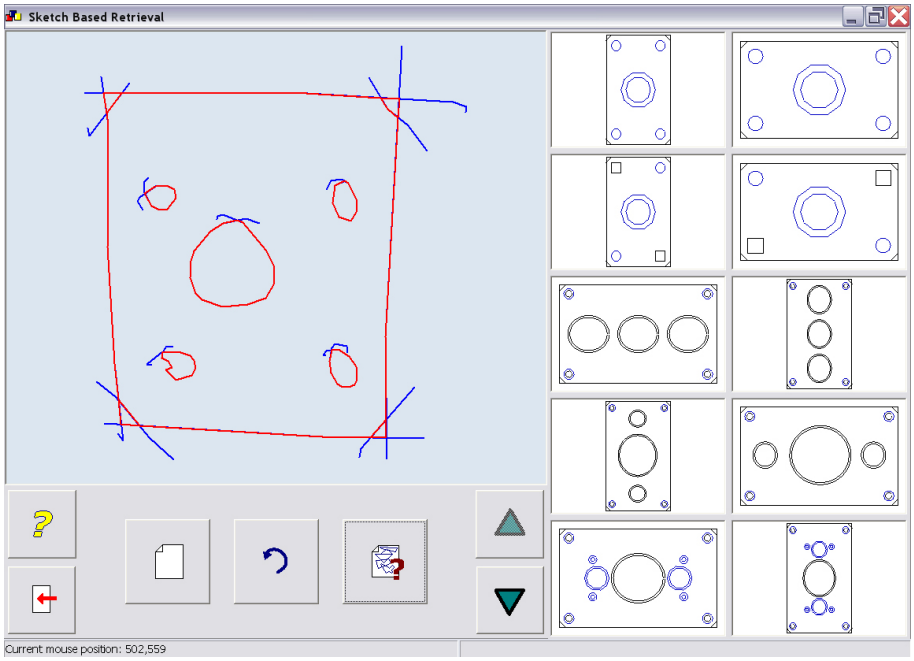


Figure 5.1: Sketch-Based Retrieval prototype.

we made a preliminary usability test with users at CENTIMFE[1]. We asked users to sketch a set of queries, to comment the returned results and to answer a questionnaire, while we collected the produced data and videotaped the tests. This experiment involved three users a database with 78 drawings and 12 queries.

5.1.1 Application Description

Using the techniques described before, we developed a Sketch-Based Retrieval prototype to retrieve technical drawings[2]. Our system allows retrieving sets of technical drawings, in CAD format, similar to a hand-sketched query. Figure 5.1 depicts a screenshot of our application, where we can see on the left the sketch of a plate and on the right the results returned by the implied query. These results are ordered from top to bottom and from left to right, with the most similar on top.

We also developed the simplification and classification component, to extract topological and geometric information from drawings. Information thus collected was used to

[1]CENTIMFE is a technological training center for the Portuguese Mould Industry.
[2]More information can be found in `http://immi.inesc-id.pt/projects/sbr/`.

compute geometric descriptors and to create the topology graph, and correspondent topological descriptors. Resulting descriptors were then inserted into two indexing structures (NB-Trees), one for topology and another for geometry.

5.1.2 User Evaluation Tests

The main goal of this experiment was to test classification and retrieval algorithms and collect users' opinions about the answers to their queries. Additionally, we also asked them to evaluated the user interface, suggesting changes to improve the next version of the prototype. The complete user evaluation tests, including the setup of the material for the experiment, took around five hours, and was performed at CENTIMFE headquarters.

5.1.2.1 Description of the Experiment

The evaluation tests involved three users from CENTIMFE and was divided in three parts. First, we explained the experiment and introduced the SBR prototype to users. During the second part users executed two sets of queries using the prototype, first sketching basic drawings and after querying for simple technical drawings. Finally, users answered a questionnaire, where we try to figure out their profile, their opinions about the prototype and their evaluation of the user interface. We also asked them, in an informal manner, about suggestions and ideas to improve the current version of the SBR prototype.

We conceived the experiment to take around 90 minutes per user and to be videotaped and photographed (see Figure 5.2). Our tests were made entirely of individual sessions, to minimize the impact of the experiment in users work routine. Each individual session included the following steps:

- Description of the experiment;
- Introduction to the SBR prototype;
- Accomplishment of tasks (querying) using two sets of queries;

Figure 5.2: Observer, user and video camera during a test session.

- Answering to the questionnaire;
- Informal conversation about the SBR system.

During the execution of tasks users were encouraged to make comments ("think loud") and even raise questions to observers. This form of interaction proved to be very fruitful, since we were able to collect useful information from users.

The query session comprised two parts, one using basic geometric shapes and another using simple technical drawings. After the querying session each user answered to a questionnaire about their previous experience in drawing, their opinion about the SBR system and about its user interface. At the end of each session we had an informal conversation with the user focusing the concepts beyond SBR, the functionality of the prototype and its application to real situations in the mould industry. From these conversations we gathered relevant information, since users were not constrained by previously prepared questions.

Drawing Database

The drawing database used in this experiment was constructed by joining two sets, one with basic drawings and other with simple technical drawings of plates, in a total of 78 drawings. Some instances are rotated or inverted versions of other drawings in the database, because we want to check that our retrieval algorithm is orientation indepen-

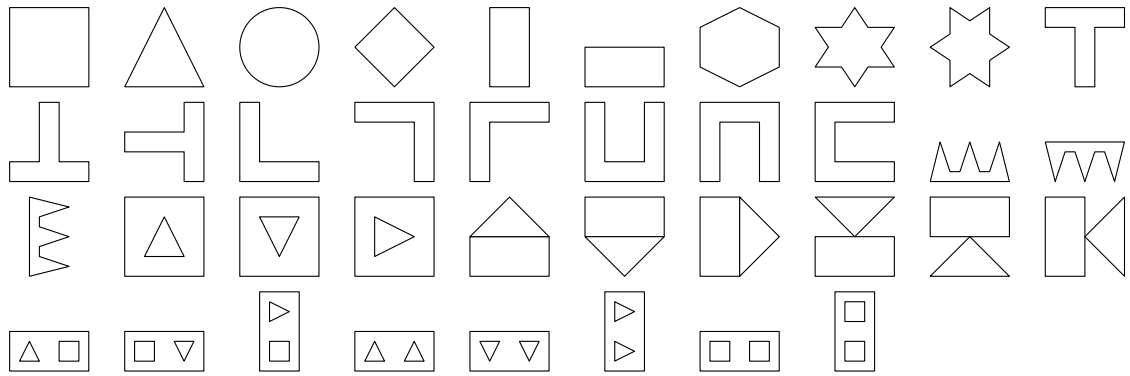


Figure 5.3: Basic Drawings.

dent. The set of basic drawings has 38 elements, composed mainly by simple shapes or combinations of two or three simple shapes, as depicted in Figure 5.3, while the set of simple technical drawings has 40 elements, which are simplifications of technical drawings of mould plates (see Figure 5.4).

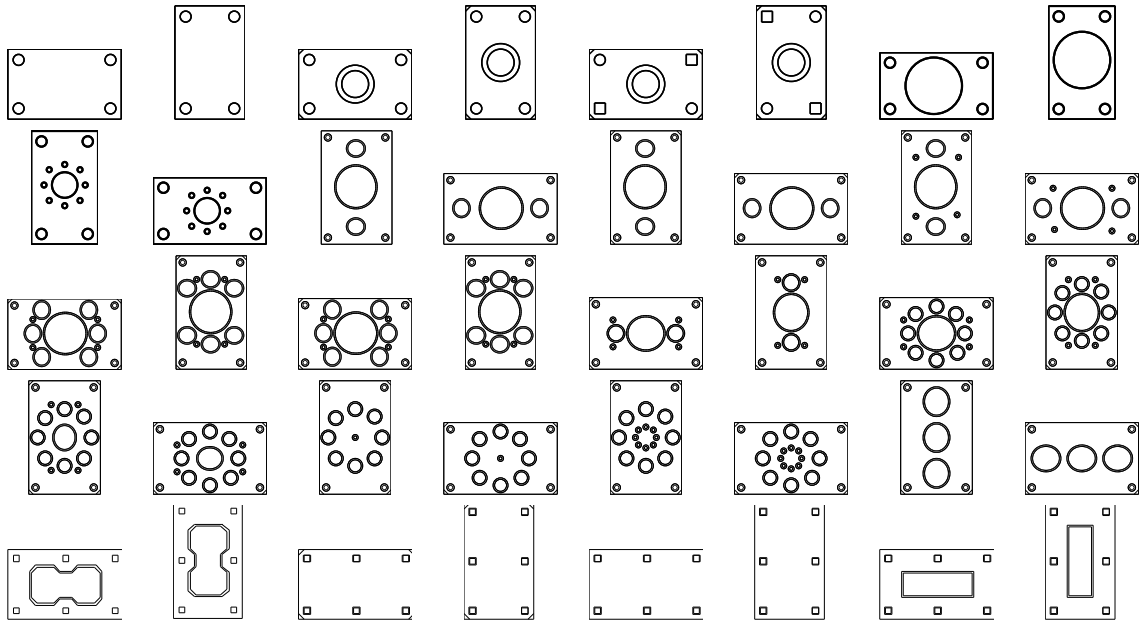


Figure 5.4: Simple technical drawings of mould plates.

Queries

In this section we present the twelve drawings that users were asked to search for using our prototype, and the reasons why we selected them. Users started by sketching six basic drawings and then they sketched six simple technical drawings. This order of actions

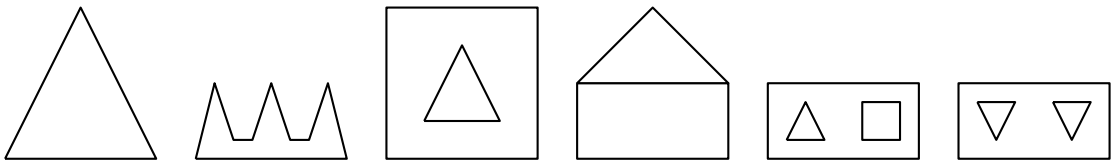


Figure 5.5: Basic drawings to search in the database.

assured that the first retrieving task (basic drawing) work as a training session. This way, when users perform the second task they already are used to the prototype.

We presented six basic drawings, Q1 to Q6, (see Figure 5.5) to users and ask them to search for these drawings in the database, by sketching something similar. The query session started by a simple shape, a triangle (Q1), passing to a more complex shape, a Chantal's comb (Q2). Drawings Q3 and Q4 are composed by two simple shapes arranged in different manners. The last couple of drawings have two simple shapes inside a rectangle, Q5 had a triangle and a square and Q6 had two triangles.

We selected queries Q1 and Q2 to allow users to practice shape sketching and to provide us with some input on how users sketch simple geometric shapes. Queries Q3 and Q4 were included to validate our matching algorithm, since both queries have the same geometric elements, but with different topological relationships (in Q3 we have inclusion, while in Q4 we have adjacency). Finally, we presented drawings Q5 and Q6 to see how users sketch similar drawings, with the same topology but different geometry in just one shape and to check the behavior of our algorithm in this situation.

Drawings Q7 to Q12 are simple technical drawings (see Figure 5.6) that users were

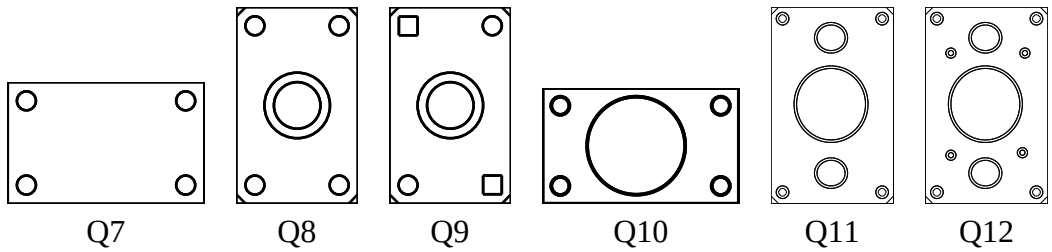


Figure 5.6: Simple technical drawings to search in the database.

asked to search in the database. The first technical drawing presented to users (Q7) is the most simple, consisting of a plate with four holes. Drawings Q8 and Q9 were selected mainly to see how users will sketch two similar drawings with just a small difference. To check if users respect relative sizes among shapes, we presented Q10 to them. This drawing has the particularity of having a very large circle in the middle and four small circles at each corner. Finally, we asked users to search for two very similar drawings, Q11 and Q12, where the last one had some more shapes.

Final Questionnaire

The questionnaire was divided in three parts and can be found in Appendix A. One was designed to collect general information about users experience on sketching, drawing tools usually used and input devices preferred and most used. In the second part we questioned users about the use of the SBR prototype to retrieve technical drawings. Users were asked about the time spent to get results and about the quality of results. Finally, we have a set of questions to evaluate the user interface in terms of window layout, size of the main areas and icons of buttons. The time to complete the questionnaire was approximately 10 minutes.

5.1.2.2 Experiment Execution

All tests took place in a meeting room, where we installed the video camera and setup the Tablet PC running the prototype. While users were performing the requested tasks (see Figure 5.7), one of the observers was taking some notes and the other was controlling the video camera and the digital still camera.

Users involved in the experiment have a good know-how on drafting with CAD tools and no prior experience on using an SBR system. This way, they are initially unbiased. However, after the first sketching session we give them some tips on how to sketch “good queries” for our retrieval system.

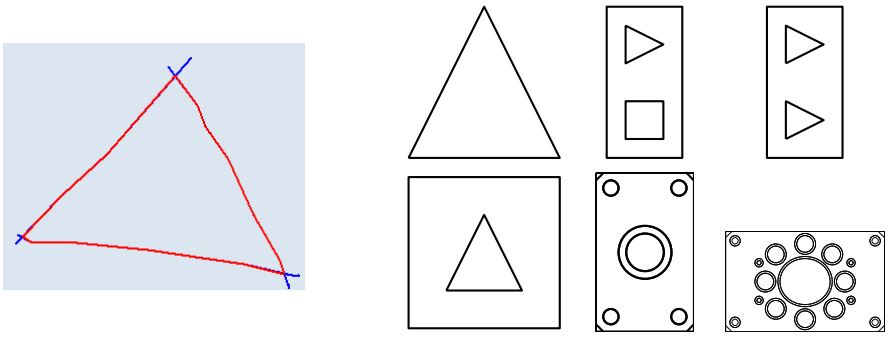
These tests only involved three users, because the prototype was in its first iteration

Figure 5.7: User performing the experiment.

and more than collect good usability results, we wanted to validate our approach (and algorithms) and get feedback from real users in order to improve the next version of the prototype.

Basic Drawings

Sketches drawn to search for a triangle have noticeable differences from the original shape. Among these differences we highlight the fact that users ignore the angles and the aspect ratio of the triangle. This situation occurs in sketch S1A (Sketch performed by user A for query Q1). However, our algorithm was always able to return the correct drawing within the first ten results. In this case the correct result was the first one, as



Sketched Query Returned Results

Figure 5.8: Sketch for Q1 made by user A and returned drawings.

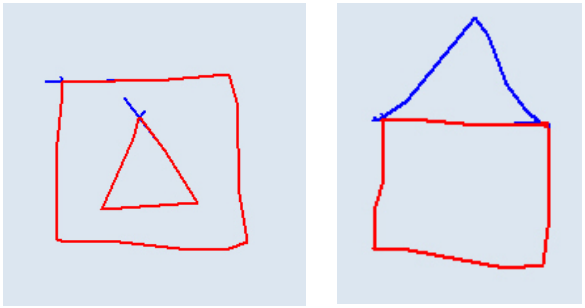


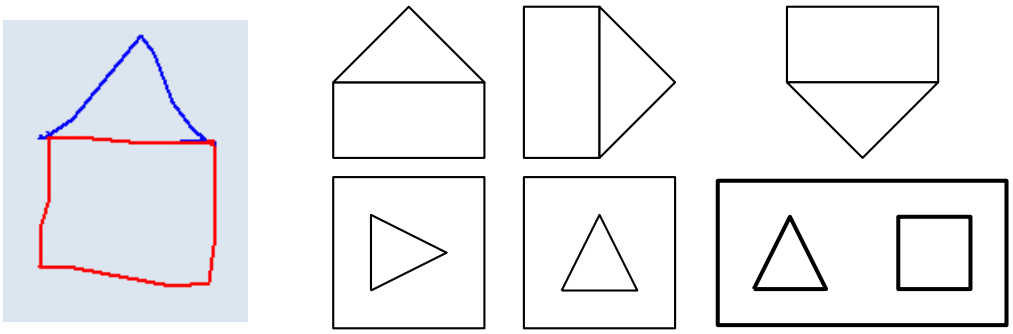
Figure 5.9: Sketches made by user C for Q3 (left) and Q4 (right).

depicted in Figure 5.8 (results are order from left to right and top to bottom).

When users had to sketch a more complex shape, for instance the Chantal's Comb (Q2), it was clear that they had some difficulty in representing it at the first try. While sketching this shape, users used the UNDO command very often, because they made a lot of mistakes. However, when they executed the query the results were very good, since the wanted drawing was always the first in the results list.

To execute queries for Q3 and Q4 users had to draw a triangle and a square or rectangle. As in Q1 the triangles sketched for Q3 and Q4 did not respect angles or aspect ratio. Moreover, users did not distinguish sketching a rectangle and a square, as illustrated in Figure 5.9, where user C sketches the rectangle and the square similarly.

The lack of accuracy in sketches had a major effect in results for Q3, since the desired drawing did not appeared in the firsts results. On the other hand the query for Q4 produced good results despite the inaccurate sketches, as depicted in Figure 5.10.



Sketched Query Returned Results

Figure 5.10: Sketch for Q4 made by user C and returned drawings.

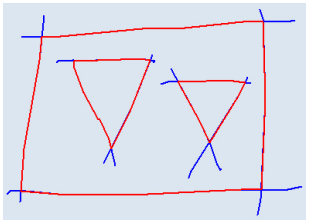


Figure 5.11: Sketch made by user A for Q6.

In queries for Q5 and Q6 users searched for similar drawings with a small geometric difference, one had two triangles inside a rectangle while the other had a square and a triangle. This difference was clearly represented in sketches produced by users. Additionally to the lack of accuracy referred previously, users also had no concerns about shape alignments, as depicted in Figure 5.11. However, the results for queries Q5 and Q6 fully satisfied users, because the desired drawing was always in first or second.

Simple Technical Drawings

The major problem encountered by users when sketching a query for Q7, and for the other drawings, was the accuracy used to draw circles. However, after a few tries users successfully sketched what they wanted. Despite the problems encountered during sketching, users were very satisfied with returned results.

Sketching a query for Q8 highlighted the problem of drawing circles, because users represented all six circles of the original drawing. The task of sketching these circles was error-prone and sketched circles were just rough approximations of a real circle, as depicted in Figure 5.12.

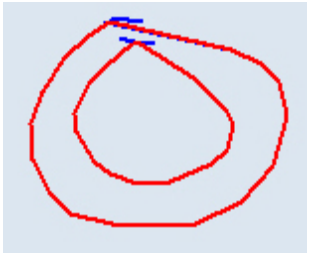
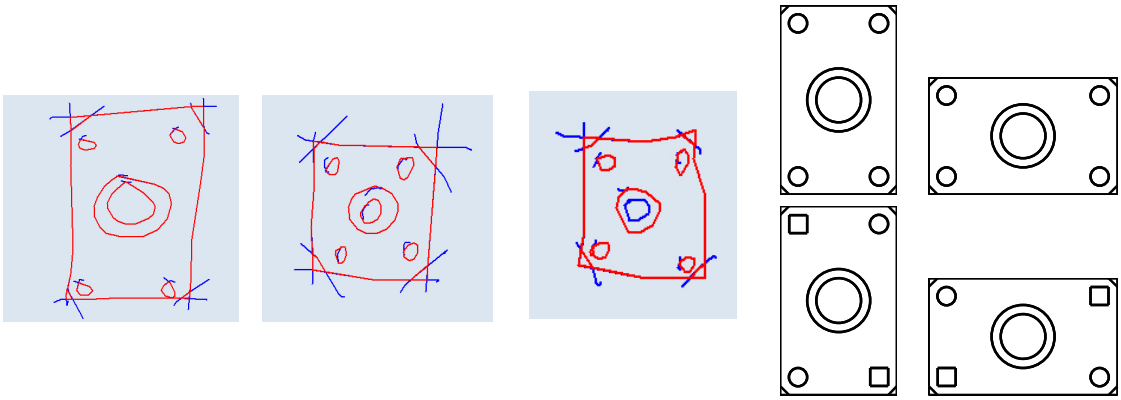


Figure 5.12: Sketch of two concentric circles.



Sketched Queries Returned Results

Figure 5.13: Sketches for Q8, one for each user, and returned drawings.

Despite the lack of accuracy in sketches for Q8, the SBR prototype always returned the desired drawing in first place. Moreover, the first four results were the same for all users, as illustrated in Figure 5.13.

The major difference between Q8 and Q9 was that users had to draw two squares instead of two circles in the corners of the part. In spite of roughly sketched circles, squares were easily identified, because users sketch them correctly, as shown in Figure 5.14. Indeed sketched squares were closer to a rectangle than to a square but the resemblance is greater than that between the sketched circle and the circle itself.

Users respected the topological relationships of shapes while drawing their sketches. Queries for Q10 were a good example of this, because regardless the accuracy of sketched shapes, users positioned these taking into account their relative position. In this case all users sketched the outer rectangle and the five circles with little geometric accuracy but respecting the placement of shapes in the original drawing, as depicted in Figure 5.15.

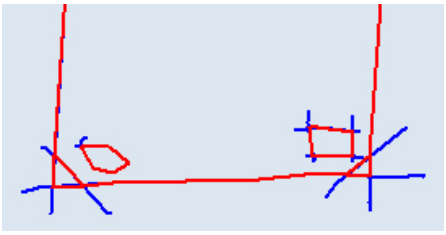
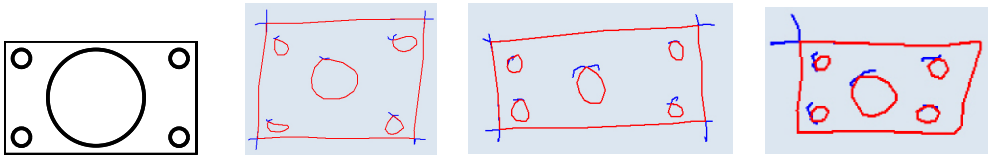


Figure 5.14: Detail of the sketch made by user A for Q9.



Original Drawing Sketched Queries

Figure 5.15: Original drawing (Q10) and sketches performed by each user.

Drawings for Q11 and Q12 were more complex to sketch than previous ones, because both had several circles with different sizes and some of them were concentric. Although, users did not represent all the circles, our system returned the wanted drawing within the first eight results. These were due mainly because of our multilevel description scheme that computes descriptors for several representations of the same drawing, using different levels of detail.

5.1.2.3 Analysis and Results

During test sessions each user performed a set of sketched-queries, with the objective of obtaining the complete filename of each searched drawing. To achieve this users had to locate the wanted drawing in the result list returned by the SBR prototype and then opened a pop-up window were he could find some details about the drawing. However, to have a well defined end, we consider the task accomplished as soon as the user selected the drawing from the results list, ignoring the time spent opening and resizing the pop-up window to obtain the filename.

Since sketches are rough approximations of original drawings, it is acceptable that the wanted drawing is not the first returned result. Therefore, the SBR prototype used in this evaluation is able to return twenty similar results for each query, displaying ten at a time. If the wanted drawing is not in the first ten results the user must proceed to the next ten. This operation takes time and causes some mistrust to the user. So, to make our prototype usable and trustful the desired drawing should appear in the first set of results.

We checked the position of the desired drawing within the returned results for each query and summarized it in Figure 5.16. From the analysis of the depicted chart it is clear

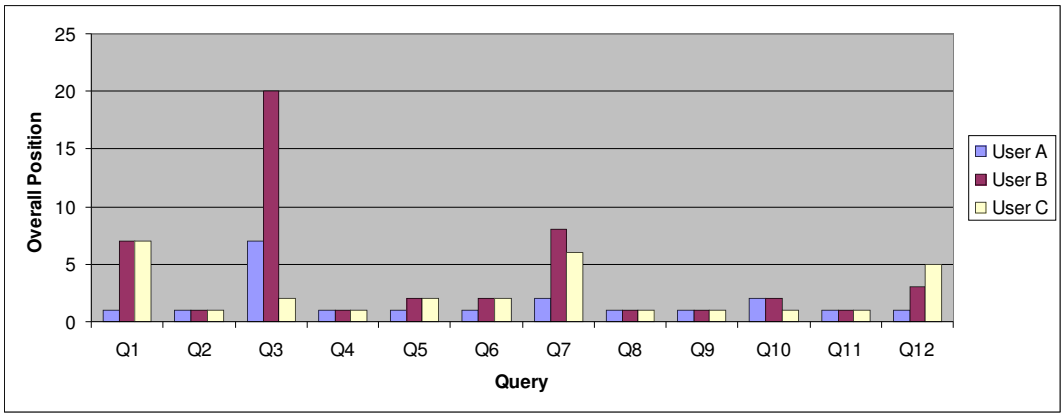


Figure 5.16: Overall position of the desired drawing in the results list.

that in the majority of the queries, wanted drawings were in the first five returned results. Moreover, we can see that in all queries, except one, the desired drawing was within the ten first results, giving some trust to the user.

These results report to successful queries - queries that produce a set of results that include the desired drawing. Unsuccessful queries occurred mainly because of incorrect polygon identification, due to the lack of accuracy in sketches.

Another measure used to evaluate our prototype was the number of sketched-queries necessary to achieve the desired drawing. The correspondent values are presented in Figure 5.17. Looking at this chart we can observe that in the majority of the cases we have a successful query with the first sketch. However, there are some situations where users had to repeat the sketch. Most of the times another iteration was enough to achieve a successful query. Even in the worst situation users re-sketch the query only four times.

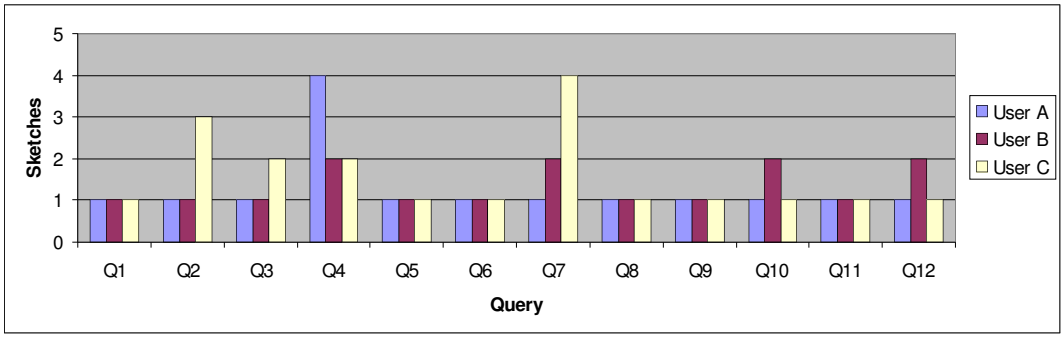


Figure 5.17: Number of sketches drawn before finding the correct result.

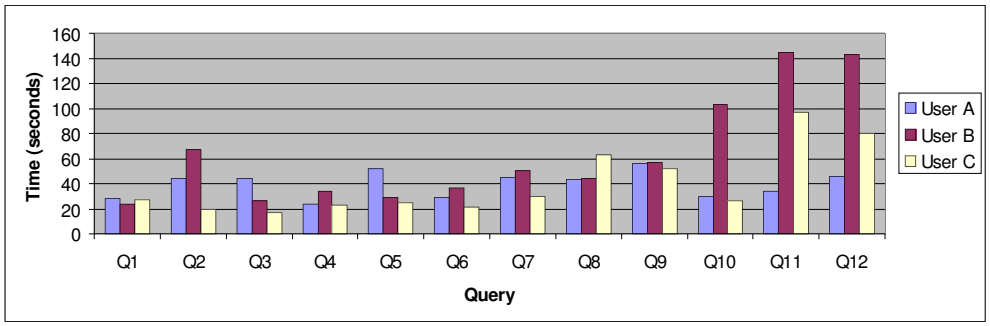


Figure 5.18: Time spent performing queries.

To be useful, a SBR system must provide good results on short notice. We measured the total time including sketching and query execution on a Tablet PC (Pentium III @ 800 MHz, running Windows XP with 256 MB of RAM). Query execution proper took from two to ten seconds, while the total time for users to draw the sketch and obtain results was less than one minute, in most cases. Values for each query are shown in Figure 5.18.

Defining what is considered a short time was one of the purposes of this usability test. From informal conversations with users and analysing their feedback during the sketching session we noticed that they were satisfied with the amount of time spent sketching and waiting for results.

Questionnaire

The first set of questions in the questionnaire was meant to know previous experience of users and what kind of input devices they usually use and prefer to use. From results, we conclude that users have a good experience with CAD tools, for 2D, 3D and modeling. Moreover, we can say that users usually use the mouse as input device.

From the second part of the questionnaire, we tried to extract some feedback from users about the performance of the SBR prototype to retrieve technical drawings using sketches. Results show that users like the idea of specifying a query using sketches, that the quality of results and the time needed to find the desired drawing are very satisfactory.

Finally, from the third part of the questionnaire, we wanted to know what users think

about the user interface of the SBR prototype. All users considered the drawing area too small and the buttons area too big, but they liked the general layout of the window. Users required a new design for the Query icon, but considered that the others are good. Another suggestion common to all users was the substitution of the Up and Down buttons by a Scrollbar to control the displayed results. Regarding to the presentation of results, users refer that it was difficult to locate at a glance a specific drawing among ten similar ones. They suggested displaying only five results at a time.

5.1.2.4 Conclusions

We have described our preliminary usability test for the Sketch-Based Retrieval prototype. We tried to evaluate the main retrieval algorithms and the user interface of the prototype. This experiment was also important to see how users sketch queries using a Tablet PC and a pen. To that end we involved three draftspeople working in the mould industry. Subjects performed different sketching tasks to search for a set of drawings using our prototype.

Notwithstanding the low number of users involved, preliminary results are very encouraging. In the majority of the queries, sought drawings were among the first five results and almost always within the first ten results, which gave some trust to users. One of the things that we observed during the execution of tasks was that users did not care about where in the order of retrieval the intended drawing appears, the important fact being that it was there. One of the users produced this comment ”It [the SBR system] found it [the drawing]! That is what counts!”

In summary, users liked the interaction paradigm very much (sketches as queries), were satisfied with returned results and pleased with the short time they had to spend to get what they wanted in contrast to more traditional approaches. From users’ comments and suggestions, and from our observations, we are improving our prototype and algorithms. We plan to test the new version of the prototype with a larger number of users and with a larger database of drawings, to get more supported results and conclusions.

5.2 Sketch-Based Retrieval of Clip-Art Drawings

These days there are a lot of vector drawings available for people to integrate into documents. Typically, such clip-art drawings tend to be achieved and accessed by categories (e.g. food, shapes, transportation, etc.). However, to find a drawing among hundreds of thousands is not an easy task.

In this section we present and evaluate an application to index and retrieve clip-art drawings by content, using topological and geometric information automatically extracted from graphics [Fonseca 04a, Fonseca 04c, Fonseca 04b]. This prototype can already handle databases with thousands of drawings using commodity hardware. To evaluate it we conducted preliminary usability tests with twelve users that performed two tasks and answered a questionnaire at the end. The experiment involved a database of 968 clip-art drawings and seven queries.

5.2.1 Application Description

We developed a prototype to retrieve clip-art drawings, called Bajavista[3], which allows retrieving sets of drawings similar to a hand-sketched query.

Figure 5.19 depicts a screen-shot of our application. On the top-left we can see the sketch of a cloud and on the bottom results returned by the implied query. These results are ordered from left to right, with the most similar on the left. It is also possible to perform Query-by-Example allowing the user to select a result and use it as the next query, since our classification scheme handles graphics and sketches in the same manner.

Additionally, this system also includes a set of heuristics to simplify drawings. These eliminate redundant information and useless elements from vector graphics, increasing the processing speed and reducing the complexity of extracted information. The classification component of this system extracts topological and geometric information from drawings, convert it in descriptors and insert these into the correspondent topological and

[3]More information can be found in `http://immi.inesc-id.pt/projects/bajavista/`

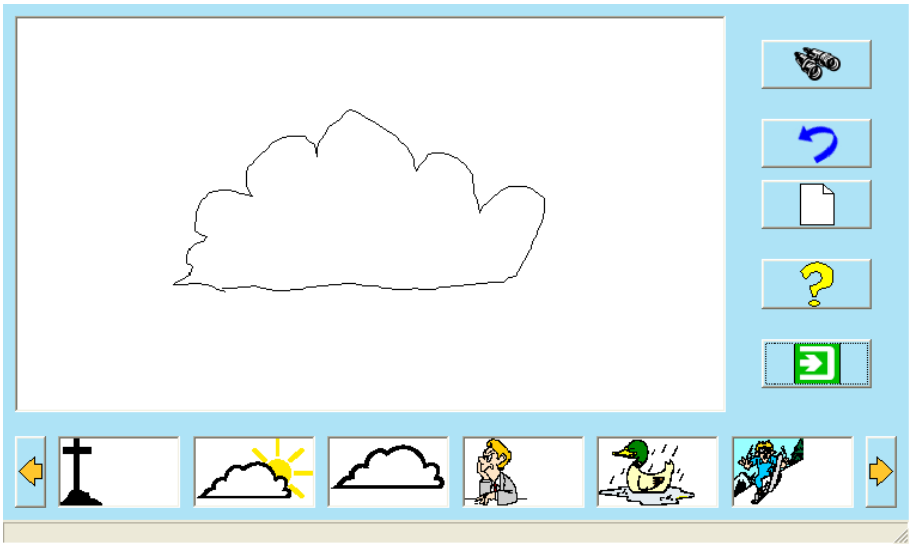


Figure 5.19: Clip-art finder prototype.

geometric indexing structures. In this application we created an indexing structure with the topological information from all drawings and an indexing structure for each drawing with its geometric information. Additionally, we had to create another indexing structure with geometric information from all drawings. This extra indexing structure is used when users draw just a polygon as query. In this situation, there is no topological information and consequently the fastest way is to only perform geometric matching using this extra structure.

5.2.2 User Evaluation Tests

These tests served mainly to evaluate the quality of returned results, and consequently of the classification and retrieval algorithms. Additionally, we collected feedback about the general utility of an application of this kind and about its user interface functionality, through a questionnaire.

5.2.2.1 Experiment Description

We conducted preliminary usability tests using a Wacom Cintiq LCD digitizing tablet and a cordless stylus to sketch queries. We involved twelve users, 8 males and 4 females,

from 23 to 31 years old. All users were used to computers, but none of them have ever used a digitizing tablet. For testing, we used a database of 968 clip-art drawings. These drawings were classified using our multilevel scheme to derive descriptors for each level of detail and for each subpart. This way, we not only describe the overall drawing, but we also describe subparts and different levels of representation of it. Resulting descriptors were then inserted into NB-Trees.

Our experiment was made of three parts. First, we gave users a brief description of our prototype, and allowed them to practice using the digitizing tablet. Second, users performed two tasks, one where they searched drawings from a verbal description and other where we provided sample drawings. Finally, users answered to a questionnaire.

Clip-art Database The set of 968 clip-art drawings used in our experimental evaluation was extracted from a sample clip-art CD. We selected drawings from different categories, such as, food, transportation, etc. Figure 5.20 shows some clip-art drawings from our database.

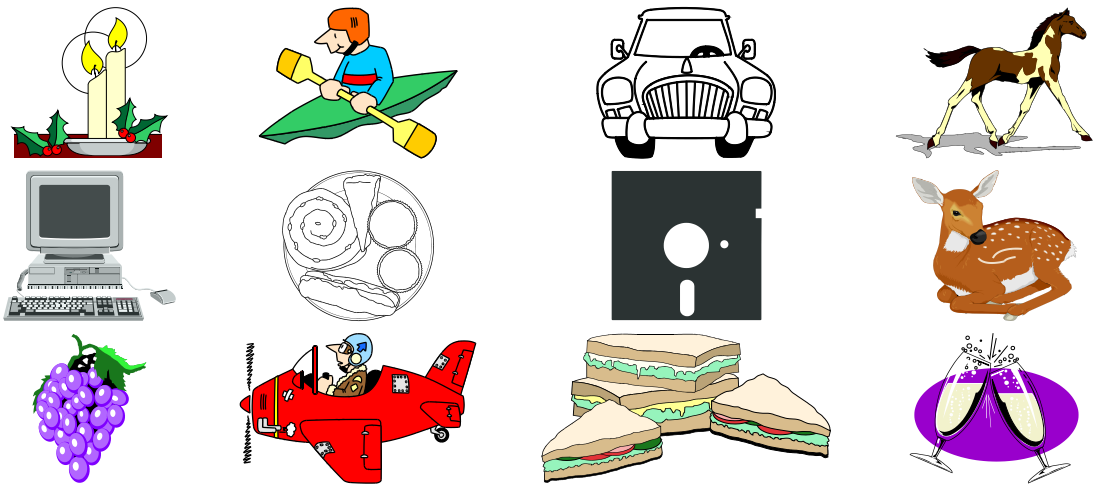


Figure 5.20: Example of clip-art drawings from our database.

Task 1 This task comprises the searching of drawings by providing users with verbal queries describing objects. We asked users to search for:

- Q1 - A car.
- Q2 - A cloud.
- Q3 - A bunch of grapes.
- Q4 - A beret.

The main goal of this task was to measure users satisfaction about returned results. We told users that they could also use Query-by-Example, *i.e.* they could use a result as query, to get at more results.

Task 2 In Task 2 we asked users to search for drawings provided by us. The goal of this experiment was to check in what position of the result list the corresponding drawing appeared.

Q5 Q6 Q7

Figure 5.21: Drawings used as query examples.

5.2.2.2 Analysis and Results

Task 1 The outcome from this task revealed that searching by sketch was in general less successful than using Query-by-Example (QbE), as shown in Figure 5.22. This is due mainly to the way people drew objects described verbally.

Task 2 The second task produced diverse outcomes, which must be analyzed one by one (see Figure 5.23). Drawing Q5 produced the best results, because it is made of simple

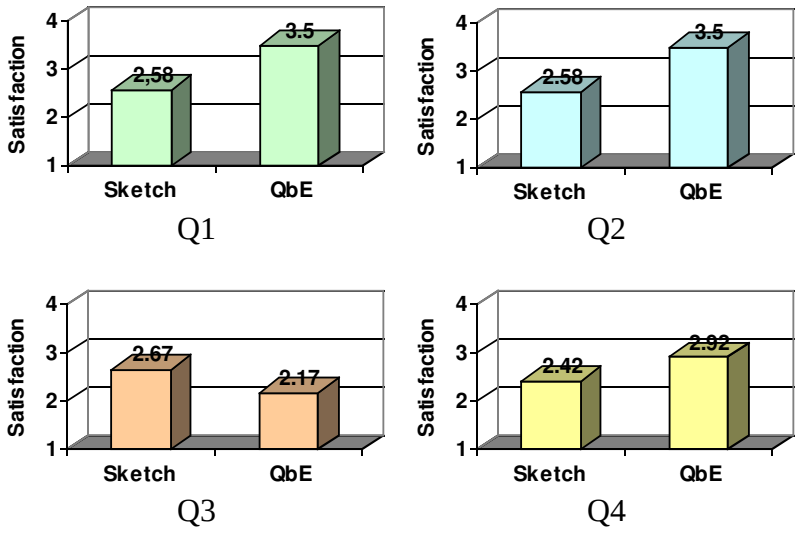


Figure 5.22: User satisfaction about returned results (1-Bad 4-Good).

geometric shapes (and consequently easy to draw) and has topological relationships between its elements. The wanted drawing for this query was almost always among the first six results. Drawing Q6 presented average to poor results, because it has a very strong geometric component, requiring the user to have some drawing skill. Thus, if the user does not draw it carefully and precisely, the final result is not the best. Finally, drawing Q7 achieved the poorest results. This drawing has some small details, which we knew *a priori* would be eliminated by our simplification heuristics. However, users were not aware of that and drew all details, producing a sketch different topologically (and geometrically) from the stored representation.

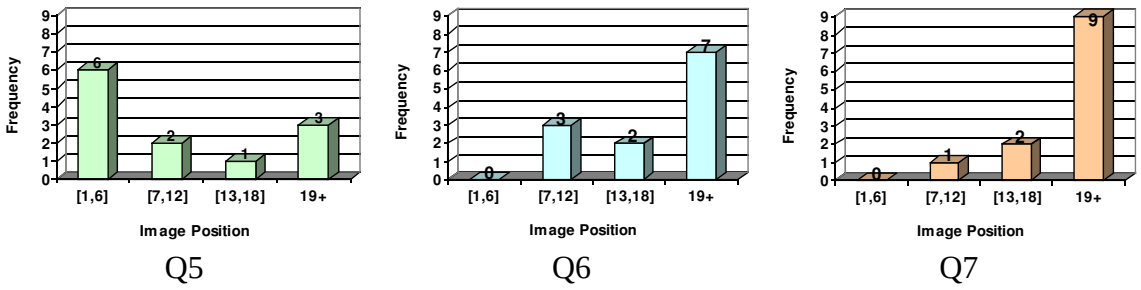


Figure 5.23: Number of times that the wanted result appeared at a given position.

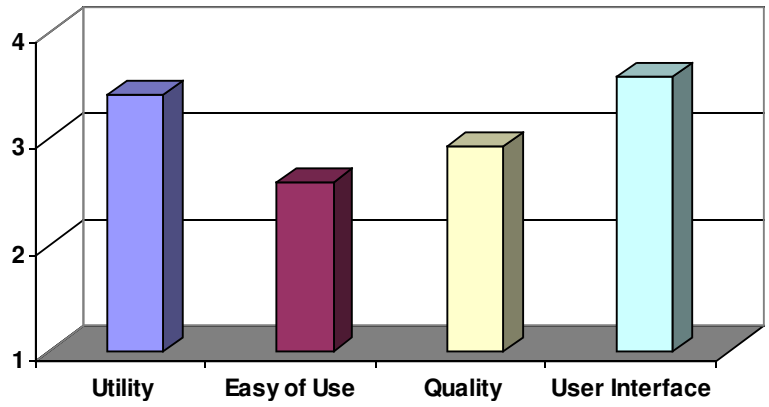


Figure 5.24: Users' opinion about the BajaVista prototype (1-Bad 4-Excellent).

Questionnaire In general, users enjoy the interaction paradigm very much (sketches as queries), even though some did not like the idea of sketching "too much detail" when looking for more complex drawings. They were satisfied with the returned results and pleased with the short time they had to spend to get what they wanted in contrast to more traditional approaches. Query times, using an AMD Duron @ 1.3GHz were between 1 and 2 seconds, which most users found satisfactory. Furthermore, users liked the user interface and its layout, but suggested redrawing the icons.

Finally, we collected some comments from users, which we transcribe:

- "After all this work sketching a query, I was expecting better results!" (retrieving drawing Q6)
- "This requires that I have some drawing skills!"
- "I was expecting something more similar to my sketch, but this one is even better!" (retrieving drawing Q2)
- "Yes, it was something like this that I was searching for."
- "Great! It found all of this!" (searching a car using Query-by-Example in Task 1)

5.2.2.3 Conclusions

We have described our prototype to retrieve clip-art drawings and the preliminary usability test to evaluate it. We conducted tests involving twelve users, that performed two types of queries.

Experimental results show that the BajaVista system performs better for drawings which contain collections of easy-to-draw shapes, with a strong topological component. In these cases, the topological filtering is effective and reduces the number of drawings to compare in the geometric matching. Furthermore, easy-to-draw shapes assure that users will sketch something very similar to what is stored in the database.

5.3 Summary

In this chapter we presented two prototypes developed using the principles and components from our approach. The Sketch-Based Retrieval prototype was developed to retrieve technical drawings, while the BajaVista system was developed to retrieve clip-art drawings. These systems attack two different domains, one where the topological structure of drawings is very rich (technical drawings) and other where the geometric information is more relevant (clip-art drawings). However, experimental evaluation with users revealed that our approach presents good results for both domains.

We also describe the two usability tests, one with real users and other with generic users. The main goals of these tests were to evaluate the accuracy of the classification and retrieval algorithms and to collect feedback from users. Results from the usability tests showed that these goals were met, since users were able to retrieve requested drawings in a short time. Moreover, users liked the use of sketches as queries, were satisfied with returned results and with the short time needed to achieve the desired results.

We consider our results very encouraging and comparable to other systems performance. For instance, a comparison study of three systems for Content-Based Retrieval

of Images [Heesch 03] revealed that the fastest will need at least 5 seconds to find results within a collection of 2,000 images, on a Pentium IV @ 1.8GHz. On the other hand, our systems take less than 2 seconds to return results from a data set of 1,000 drawings using an AMD Duron @ 1.3GHz.

6

Conclusions and Future Work

When men speak about the future, the gods laugh.
— Chinese proverb

This chapter summarizes the dissertation, presents the final conclusions, where we enumerate the benefits of the current approach and discuss its limitations and contributions. In addition we present new research topics opened by our work.

6.1 Summary of the Dissertation

We can divide our dissertation in three main parts: 1) a review of related work; 2) an approach and correspondent components to support retrieving drawings from large data sets; and 3) results and experimental evaluation. The main goal of this work was to develop a set of techniques to efficiently and effectively support the retrieval of drawings from large databases, using sketches. First, we reviewed existent research works to understand the current techniques and to explore new possibilities. Then, we combined pattern recognition techniques, shape representation methods, topology information, graph matching algorithms and multidimensional indexing structures into a new approach for content-based retrieval of drawings. Finally, we developed prototypes using our approach and evaluated these with users. In the next paragraphs we will summarize how each chapter addressed these issues.

Chapter 2 started by reviewing content-based retrieval approaches for three different domains of application, namely, images, vector drawings and 3D objects. Although, the core of these approaches could be shared, relevant features and feature extraction mechanisms are different. The second part of this chapter analyzed techniques to describe objects shape. The third part of Chapter 2 reviewed graph and subgraph isomorphism algorithms and similarity measures between graphs. Our approach describes the spatial organization of drawings using a topology graph. So, at the core of our approach we would need a mechanism to compute the similarity between graphs. Finally, we deeply analyzed current indexing structures for high-dimensional data points. Usually, retrieval approaches convert drawing features into multidimensional vectors and the matching process into a distance calculation between a query and a set of models stored into a database. Normally, indexing structures are used to compute these distances efficiently. A realistic approach to retrieve drawings from large data sets must include a very efficient indexing structure, or else a bottle neck in the retrieval process will be created.

In Chapter 3 we first presented an overview of our approach for sketch-based retrieval of drawings by content. The second part of this chapter describes a data structure for topology matching. We defined a reduced set of topological relationships (Inclusion and Adjacency) based on Egenhofer's set of relations and we explained how we create our topology graphs. After, we described the computation of topological descriptors from graphs, using graph spectrum. Our novel method to compute a multilevel description of drawings, using levels of detail, was also presented in Chapter 3. In order to assess the stability and accuracy of our method to describe graphs, we conducted a set of experimental evaluations. In part three, we described our technique for shape representation and compared its performance to other existing methods. Finally, we describe our two step matching process, that first filters results employing topology and then refines results using the geometric information.

Chapter 4 introduced another element from our approach, the multidimensional indexing structure (NB-Tree). We described its algorithms, determined its computational

complexity and presented a complete experimental evaluation, comparing its performance with other well known indexing structures. The outcome from these tests revealed that our NB-Tree outperforms all the other indexing structures. Moreover, it is important to notice that our structure is the only one that supports data points of variable dimension.

Finally, Chapter 5 presented two prototypes for content-based retrieval of drawings, employing our approach. One for retrieving technical drawings and the other for retrieving clip-art drawings. Additionally, we described preliminary usability tests, which revealed a good acceptance from users and demonstrated that our approach can be applied to different domains of application with success.

6.2 Final Conclusions and Discussion

While the majority of surveyed work focuses on particular aspects of content-based retrieval, in this thesis we attacked it in three flanks, namely, content description, indexing and user interface. Although we have obtained results in all of them, we consider the NB-Tree, our multidimensional indexing structure, the CALI library for shape characterization and the Multilevel description method, both for content description, the most important results. The prototype applications and correspondent usability tests at the user interface, have been done mostly as concept demonstrators, to show that it is possible to combine all parts of our approach in a functional system. We will now describe the benefits introduced by our techniques, their limitations and review our main contributions.

6.2.1 Benefits of the Current Approach

From task analysis and informal conversations with users, we found out that there was a need for searching and retrieving past drawings. Existent approaches based on textual classification were not suited for their needs and other systems based on drawing-content do not support large sets of drawings or complex drawings.

Our approach overcomes existent problems by developing mechanisms for content-

based retrieval of drawings, through hand-sketched queries. Using a pen as input device provides the user with a more natural way of interaction, compared to other input devices such as the keyboard. Moreover, it takes advantage of user's natural ability at sketching and drawing to specify queries. Unlike the majority of retrieving systems, our supports large sets of drawings by integrating a new multidimensional indexing structure. Furthermore, from usability tests we noticed that users take advantage of our multilevel description scheme, because during query sessions, users did not sketch all the details of the wanted drawing. Additionally, users liked the idea of using sketches to specify queries, considered the returned results satisfactory and were pleased with the short time needed to achieve the wanted drawing.

6.2.2 Limitations

Although our approach overcomes problems unsolved by other systems, it also has some limitations. In the following paragraphs we describe the limitations of each of the components from our approach.

One of the limitations of our indexing structure, the NB-Tree, is its degeneration in a sequential search, when we have data sets of points with very similar Euclidean norms. In this situation our range and KNN algorithms have to compute the distance from the query point to all the points in the data set. However, even in this situation, our NB-Tree outperforms other structures, such as the Pyramid Technique. On the other hand, real and variable dimension data sets present Euclidean norm distributions that favor our indexing technique, as illustrated in Section 4.3. Another limitation of the indexing structure is the main memory (RAM) available, since it was designed to be load to memory before performing any operation. Although current PCs have enough memory to accommodate an NB-Tree with dozens of millions of data points, older PCs (with less RAM), may not support data sets of that size.

The CALI library that we are using to describe the geometry of objects is based mainly on global geometric features, which are good to classify simple geometric shapes.

Although, experimental evaluation revealed that it also performed well describing more general shapes, such as fish contours, we think that it will have some difficulties in describing more complex shapes.

The generic approach for content-based retrieval of drawings, described in this dissertation, assumes that classified and retrieved drawings are strongly related topologically (*e.g.* mould drawings, architectural plants, mechanical drawings, etc.). In other application domains where geometry predominates over topology, the first filtering based on topology is not fruitful. For example, in the BajaVista system we notice that the best results were achieved when we provide reach topological information in the query.

Although results from our preliminary tests were very promising, we think that the number of users involved and the size of the drawing sets used did not come up to our expectations. We would like to had performed tests with thousands of drawings and dozens of users, to state with more confidence that our approach has good performance and accuracy for very large sets of drawings. Even though we used a small database in our tests, our approach has the potential to deal with large sets of drawings. The indexing structure, that could prove the main bottle neck during retrieval, has shown good performance for data sets around one million elements, as illustrated in Figure 4.5 (right) and Table 4.6. From these, we believe that our approach will present good performance for large sets of drawings.

6.2.3 Contributions

Our approach demonstrates that we can retrieve complex drawings by content from large databases, using sketches as queries. We grouped the contributions of this research into concepts and techniques, artifacts and experimental results.

Concepts and Techniques To reach its goals, this dissertation presented new concepts and techniques. First, we developed a multilevel description scheme that describes drawings using different levels of detail. This way, we compute several descriptors for the

same drawing, offering a mechanism to search using both coarse or very detailed representations. Second, we created a new indexing structure that supports large sets of high-dimensional points of variable dimension, something that is new, since all existent indexing structures only support data points of fixed dimension. In addition, we developed a recognition library to describe the geometry of elements from drawings, that outperforms well known techniques usually used to describe shapes. Finally, we resort to the graph spectra to overcome the graph isomorphism problem, converting it in a simple comparison of vector descriptors.

Artifacts Our research produced various artifacts under the form of software components[1]. Two prototypes, one to classify and retrieve technical drawings and other to classify and retrieve clip-art drawings. A gesture recognition library, CALI, able to identify a set of simple geometric shapes and some gestural commands (Appendix C presents a detailed description of this library). A library that implements our multidimensional indexing structure, NB-Tree, and an application to measure the performance of its algorithms. Finally, an application that integrates all our theoretical developments, namely, the multilevel description, the computation of topological descriptors using graph spectrum, the CALI library to compute the geometric information and the NB-Tree indexing structure to index topological and geometric descriptors. This application can be used (called) by any content-based retrieval system that uses our scheme (topology graphs and geometric information) to describe objects.

Experimental Results At the practical level, the efficiency of our methodology has been assessed. Usability tests showed that our approach is indeed usable to quickly find drawings using sketches.

Preliminary tests of our prototypes with users revealed very encouraging results. Users were pleased with both the returned results and the interaction mechanism (sketches).

[1] All these software packages are available under the GNU Public License, at http://immi.inesc-id.pt/∼mjf.

Indeed, for the majority of the queries, drawings sought were found among the topmost five results and could almost always be found within the top ten results. These results gave some confidence to users.

6.2.4 Final Remarks

This thesis proposed to investigate a way of enabling users to retrieve drawings from large databases, using sketches as queries. We believe our research has met this goal.

In this dissertation, we presented interesting solutions to many problems in original ways. For example, we described our novel multilevel description concept, that not only offers a hierarchical description of drawings, but also allows describing drawings using different levels of detail. This scheme computes several descriptors from the topology graph of a drawing, allowing the retrieval either by partial matching or by coarse specification of queries. This mechanism offers a transparent way of ignoring small details present from drawings.

To address the problem of indexing data points of variable dimension, we developed a simple and fast multidimensional indexing structure, that outperforms current approaches. Experimental evaluation showed that our NB-Tree can efficiently support point, range and nearest neighbor queries for large data sets of high dimensional points.

To overcome the NP-Problem of the graph isomorphism problem, we used the graph spectra theory to convert graphs into descriptors. Eigenvalues from the adjacency matrix of topology graphs are combined to create topological descriptors. Then, we measured the similarity between topology graphs by calculating the difference between descriptors (using the Euclidean distance). Although this mapping of graphs into descriptors is not unique, the number of collisions is small, providing us with a polynomial method to compute approximate and inexact graph and subgraph matching.

We implemented a method to describe the shape of objects in drawings, based on a generic gesture recognition library. This algorithm uses global geometric features

extracted from strokes to describe drawings regardless of size, rotation and number of strokes. Each geometric element from the drawing is mapped into a descriptor that describes its geometry. Geometric similarity is then measured by computing the Euclidean distance between descriptors.

Finally, we integrated all created components in a seamless architecture for the development of content-based retrieval systems. The resulting approach provides an effective representation, suited for automated processing, and efficient indexing and searching methods to retrieve drawings from large databases. For this, we proposed a combination of algorithms to generate the representation of drawings (shape and topology), a new indexing technique to speed-up search in large data sets of drawings and a new method based on eigenvalues of graphs to compute descriptors for topology information. We believe that our approach is general enough to handle different types of drawings (technical or clip-art drawings, for instance) and flexible enough to handle queries specifying the whole drawing or just subparts of it.

To test our prototypes we selected a set of potential users and not programmers writing the code. We evaluated our prototypes with real users and real drawings, performing tasks in a possible scenario. Results from these experiments showed that users liked the new mechanism to retrieve drawings and provided us with very important information to improve our prototypes and algorithms.

In summary, we have presented a generic approach suitable for content-based retrieval of structured graphics and drawings. Our method hinges on recasting the general picture matching problem as an instance of graph matching using vector descriptors. To this end we index drawings using a topology graph which describes adjacency and containment relations for parts and subparts. We then transform these graphs into descriptor vectors in a way similar to hashing to obviate the need to perform costly graph-isomorphism computations over large databases, using spectral information from graphs. Finally, a novel approach to multidimensional indexing provides the means to efficiently retrieve sub-drawings that match a given query in terms of its topology.

6.3 Directions for Further Research

The outcomes from this dissertation make the way for direct extensions and conceptual improvements to the current work. With the goal of offering an efficient and effective mechanism to retrieve complex drawings from large databases, we foresee four areas in which some research problems still need to be solved.

Topology Graph Currently, our topology graph only code two topological relationships, Inclusion and Adjacency. One possible extension is the use of weights in adjacency links to code different types of adjacency. This way, instead of having "adjacent" and "not adjacent" we will have Adjacent, Very Near, Near, Far and Very Far, for instance. Or else we could use Fuzzy Logic to convey different degrees of adjacency. There are studies [Sarkar 96] and theorems [Cvetković 97] that support and demonstrate the stability of eigenvalues with changes in the weights of links. Moreover, Sarkar and Boyer [Sarkar 96] verified that graphs with similar weight distribution tend to have similar spectra. Thus, our method of using the spectrum of graphs to compute graph similarity is still valid.

Another extension to topology graphs could be the integration of both the topological information and the geometric information into a single graph. Thus, at the end we will have a graph and correspondent descriptor that describes simultaneously topology and geometry. The codification of the geometric information could be done using the weights of the Inclusion links of the topology graph.

Although these two extensions to the topology graph will provide a richer description of drawings, some care must be taken while choosing weights for links to assure that drawings with similar topology and geometry still have similar descriptors.

Indexing Structure Even though our indexing method outperforms current indexing structures and supports large data sets of high-dimensional points of variable dimension, its algorithms are linear. One possible improvement to the NB-Tree could be the research

and implementation of sub-linear algorithms. This will imply new and deeper studies of the particularities of the Euclidean norm.

Another possibility is the development of a parallel version of the NB-Tree, allowing the creation of a powerful search engine for drawings. Since its algorithms are simple, we believe that this improvement to the indexing structure will not be very hard to understand and implement.

Geometric Description and Matching Our library to describe the shape of objects is based mainly on global geometric features. The addition and evaluation of new features to describe shape could be one of the extensions to this library. Furthermore, these features only allow the description of 2D objects, but if we want to extrapolate our approach to 3D objects or surfaces, more new features will be needed.

Our current geometric matching algorithm is very simple and as we said before it does not assure the best result, but it provides a good result in a short time. A possible improvement to our approach is the development of a new matching algorithm that provides a better comparison between drawings and queries. It could for instance compare only objects in the same level of graphs, distinguishing a circle inside a square from a square inside a circle.

Domain of Application We have applied our approach in the development of prototypes to retrieve vector drawings, namely technical and clip-art drawings. However, there is no limitation to these type of data. We believe that if some effort is placed in the description part, this approach will be suitable for retrieving 3D objects or even 3D surfaces. The important is to produce a topology graph, similar to ours, from extracted information. This way, all existent algorithms can be used.

Some of the extensions and improvements mentioned here are part of proposed projects or are already being tackled. The current status of this research makes us fill confident about future outcomes from these new research areas.

A

SBR Usability Tests

This chapter presents the protocol specified for the user tests performed at CENTIMFE, to evaluate our Sketch-Based Retrieval prototype.

Our experiment involved three users from CENTIMFE and was divided in three parts, as described in the protocol. First, we gave a brief description of the experiment, then users performed a set of tasks and finally they answered a questionnaire.

With this experimental evaluation we wanted to get feedback from users about our first Sketch-Based Retrieval prototype and of its user interface. We videotaped all experiments, measured times and took notes of users' comments or suggestions.

During the experiment we collected the following information:

- Number of Undos
- Number of News (new query) per query
- Number of queries per search until the final results
- Time spent before finding the desired result
- Number of strokes sketched per query

Sketch-Based Retrieval Prototype

Usability evaluation session

Thank you for having accepted to participate in this experiment. Its main objectives are the evaluation of the underlying ideas of our Sketch-Based Retrieval prototype and the validation of its algorithms. This prototype allows the retrieval of technical drawings using sketches to specify the desired drawing.

The schedule of this session is described in the following table, indicating the estimated time for each of the foreseen tasks, with an expected total time of about 1 hour and 30 minutes.

1	Experiment Description	10 m
2	Accomplishment of tasks using our prototype	50-60 m
3	Questionnaire about users, prototype and tasks	10 m

This experiment intends to evaluate the utility and the effectiveness of our prototype, so, all comments and suggestions are welcome. “Think loud” during task execution and do not feel inhibited to point out negative or positive aspects, of the prototype.

To finish, we would like to thank you the time and effort spent.

Consent form

Part of this evaluation session will be videotaped and we would like to include some excerpts in a small film about the system. Please indicate whether you authorize or not the diffusion of the excerpts where you appear:

Yes ☐

No ☐

Name: _____

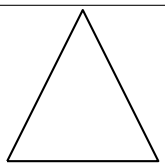
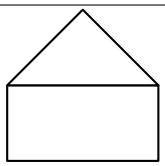
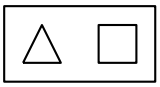
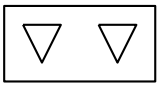
Signature: _____

First set of Tasks

(Simple Drawings)

Duration: 20 min

Please perform the following 6 (six) queries using the Sketch-Based Retrieval System and comment the returned results. Do not hesitate in making comments in loud voice, or in asking for help whenever necessary. Start the construction of each model only after you have been told to do so.

Query	Comments
 **Q1**	
 **Q2**	
 **Q3**	
 **Q4**	
 **Q5**	
 **Q6**	

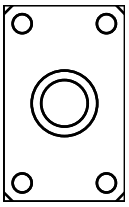
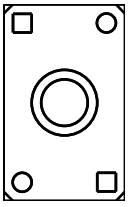
Second set of Tasks

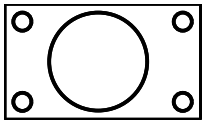
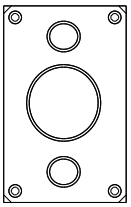
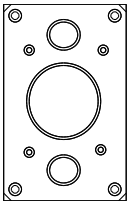
(Plate Drawings)

Duration: 30 min

Please perform the following 6 (six) queries using the Sketch-Based Retrieval System and comment the returned results. Do not hesitate in making comments in loud voice, or in asking for help whenever necessary.

Start the construction of each model only after you have been told to do so.

Query	Comments
 **Q7**	
 **Q8**	
 **Q9**	

Query	Comments
 **Q10**	
 **Q11**	
 **Q12**	

Questionnaire

General questions

1 - Do you have previous experience on free-hand sketching? Yes ☐ No ☐

2 - What kind of drawing tools do you usually use? (You can select more than one option)

a) CAD 2D ☐

b) CAD 3D ☐

c) Modeling ☐

d) Other: _____

3 - How long have you been using drawing tools? _____

4 - What input devices have you already used? (You can select more than one option)

a) Mouse ☐

b) Pen and tablet ☐

c) 3D Mouse ☐

d) Keyboard ☐

e) Other: _____

5 - What input device do you use most for drawing? (You can select more than one option)

a) Mouse ☐

b) Pen and tablet ☐

c) 3D Mouse ☐

d) Keyboard ☐

e) Other: _____

6 - What input device do you prefer for drawing? (You can select more than one option)

a) Mouse ☐

b) Pen and tablet ☐

c) 3D Mouse ☐

d) Keyboard ☐

e) Other: _____

Questions about the Prototype

Please characterize the adaptation of our Sketch-Based Retrieval prototype to retrieve technical drawings, according to:

1 - Use of sketches to specify queries. Bad | | | | Excellent

Comments:

2 - Number of iterations to achieve the wanted result. Bad | | | | Excellent

Comments:

3 - Total time to get the wanted result. Bad | | | | Excellent

Comments:

4 - Quality of results. Bad | | | | Excellent

Comments:

5 - Critics and suggestions to the Sketch-Based Retrieval prototype:

Questions about the User Interface

Please characterize the user interface of our Sketch-Based Retrieval system, according to:

1 - Size of the Sketching area. Too Small ☐ OK ☐ Too Big ☐

2 - Size of the Result area. Too Small ☐ OK ☐ Too Big ☐

3 - Size of the Buttons area. Too Small ☐ OK ☐ Too Big ☐

4 - Layout of the window. Are the three areas well distributed in the window?

Yes ☐ No ☐

5 - Quality of Button Icons

a) Quit button Bad ☐☐☐☐ Excellent

Comments:

b) Help button Bad ☐☐☐☐ Excellent

Comments:

c) Query button Bad ☐☐☐☐ Excellent

Comments:

d) Undo button Bad ☐☐☐☐ Excellent

Comments:

e) New button Bad | | | | Excellent

Comments:

f) Up button Bad | | | | Excellent

Comments:

g) Down button Bad | | | | Excellent

Comments:

6 - Functionality Bad | | | | Excellent

Comments:

7 - Critics, suggestions or new functions to integrate in the user interface of the prototype:

B

BajaVista Usability Tests

Task 1

Suppose you want to find a set of drawings related to the objects described below. Please draw a sketch for each of the objects described. If one of the results is similar to what you are looking for, you can use this result as a new query (Query-by-Example). For each of the following objects, please classify your degree of satisfaction.

Q1 - Car

Using Sketches Discontent ☐☐☐☐ Very Pleased

Using Query-by Example Discontent ☐☐☐☐ Very Pleased

Q2 - Cloud

Using Sketches Discontent ☐☐☐☐ Very Pleased

Using Query-by Example Discontent ☐☐☐☐ Very Pleased

Q3 - Bunch of grapes

Using Sketches Discontent ☐☐☐☐ Very Pleased

Using Query-by Example Discontent ☐☐☐☐ Very Pleased

Q4 - Beret

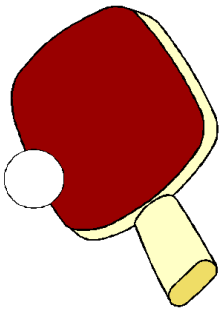
Using Sketches Discontent ☐☐☐☐ Very Pleased

Using Query-by Example Discontent ☐☐☐☐ Very Pleased

Task 2

Suppose that you are searching for the drawings depicted below. Please draw a sketch to retrieve these drawings and take note of the position where it appeared.

Q5



Position in the Results list

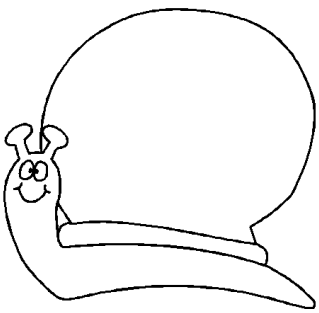
- ☐ - between 1 and 6
- ☐ - between 7 and 12
- ☐ - between 13 and 18
- ☐ - greater than 19

Q6

Position in the Results list

- ☐ - between 1 and 6
- ☐ - between 7 and 12
- ☐ - between 13 and 18
- ☐ - greater than 19

Q7



Position in the Results list

- ☐ - between 1 and 6
- ☐ - between 7 and 12
- ☐ - between 13 and 18
- ☐ - greater than 19

Questionnaire

Please classify the BajaVista prototype according to the following points:

1 - Utility of a program of this kind.

Bad ☐☐☐☐ Excellent

2 - Easyness in finding what you want.

Bad ☐☐☐☐ Excellent

3 - Quality of results.

Bad ☐☐☐☐ Excellent

4 - Quality of the user interface.

Bad ☐☐☐☐ Excellent

C

CALI: An Online Scribble Recognizer for Calligraphic Interfaces

User interfaces are evolving away from the classical WIMP (windows, icons, mouse and pointing) paradigm. Among the new generation of intelligent systems there is a broad class based on new modalities such as sketching, hand-drawing and stylus input gestures, which we call calligraphic interfaces [Jorge 94]. These interfaces rely on gesture recognition components to identify sets of visual elements and commands, and help people use computers in more engaging and natural modes. To this end, we have developed a simple recognizer capable of identifying the most common constructs and gestures used in drawings. The techniques described in this appendix, have been put to good use in shape characterization (see Section 3.3). We have found out that the features used for geometric classification of gestures can also be used for describing arbitrary shapes effectively. Here we describe our techniques applied to gesture and shape recognition.

We will now describe our shape and gesture recognizer, called CALI. It is a fast, simple and compact online recognizer that identifies Scribbles (multi-stroke geometric shapes) drawn with a stylus on a digitizing tablet. Our method is able to identify shapes of different sizes and rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines. We use temporal adjacency to allow users to input the most common shapes in drawing such as triangles, lines, rectangles, circles, diamonds and ellipses, using multiple strokes. We have further extended this approach to identify useful shapes such as arrows, crossing lines and unistroke gesture commands and have developed a li-

brary of software components to make this software generally available. The recognition algorithm uses Fuzzy Logic and geometric features, combined with an extensible set of heuristics to classify scribbles. Evaluation results show recognition rates over 97%.

The rest of the this chapter is organized as follows. First we describe related work about calligraphic interfaces and gesture recognizers. Then we present our gesture recognizer, CALI. In Section C.2 we identify the geometric features used by the recognizer, we explain how Fuzzy Logic is integrated in the recognizer and how it models ambiguity. The next section describes CALI architecture and its main classes. Finally, we present the results of our experimental evaluation and some conclusions.

C.1 Motivation and Related Work

The idea of calligraphic interfaces is not new. Sutherland presented Sketchpad the first interactive system that used one light pen to draw diagrams directly over the screen surface [Sutherland 63]. Nevertheless, due in part to ergonomic problems with the light pen and the invention of the mouse in 1964, current graphical interfaces relegated pen-based interactions to specific CAD applications. Things changed with the appearance of the first pen computers in 1991, even though pens were used mainly to input text through handwriting recognition.

The Newton system [Kounalakis 93], one of the first hand-held pen-based computers, incorporates handwriting, shape and gesture recognizers. While the shape recognizer only handles uni-stroke, axis-aligned shapes, the gesture recognizer is more suitable for text editing than for drawing schematics.

Our work is based on a first approach to recognize schematic drawings developed at the University of Washington [Apte 93], which recognized a small number of non-rotated shapes and did not distinguish open/closed, dashed or bold shapes. Zhao described an interactive editor [Zhao 93] based on the StateCharts [Harel 87] formalism. Although the editor uses a mouse and direct manipulation, many of the ideas described by Zhao express

an approach based on diagram recognition guided by syntax. Gross described a system fundamentally based on sketches that are partially interpreted for use in architecture drawings, using a recognizer substantially simpler and less robust than ours [Gross 96b]. Although this recognizer is trainable, the number of identified shapes is apparently smaller than ours. Landay uses a recognizer developed by Rubine [Rubine 91] to sketch graphical arrangements of bidimensional documents [Landay 96]. Rubine's recognizer is a trainable single-stroke gesture recognizer that uses a classic linear discriminator-training algorithm. One main advantage of our approach over Rubine's is that our method allows the user to draw scribbles without any restriction and as close as possible to the intended shapes. This way, we can recognize shapes without the limitation of being single-stroked (i.e., without having to rule out crosses or arrows).

Other authors [Ulgen 95] have proposed more complex methods, involving neural networks, to identify a small number of geometric shapes (rectangles, squares, circles, ellipses and triangles). These shapes are recognized independently of size and rotation, but they cannot be drawn using dashed or bold lines. Even though the authors claim good performance for their method, the paper does not present clear measurements to back their claims.

C.2 The Recognition Algorithm

The recognition method used in CALI is based on three main ideas. First, we use entirely global geometric properties extracted from input shapes, because we are interested in identifying geometric entities. Second, to enhance recognition performance, we use a set of filters either to identify shapes or to remove unwanted shapes using distinctive criteria. Third, to overcome uncertainty and imprecision in shape sketches, we use fuzzy logic [Bezdek 92] to associate degrees of certainty to recognized shapes, thereby handling ambiguities naturally.

This algorithm recognizes elementary geometric shapes, such as `Triangles`, `Rec-`

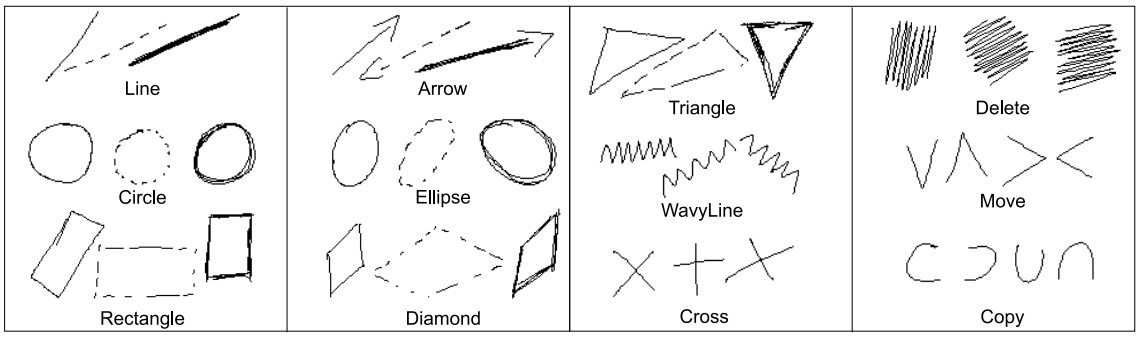


Figure C.1: Shapes identified by the recognizer.

`tangles`, `Diamonds`, `Circles`, `Ellipses`, `Lines` and `Arrows`, and five gesture commands, `Delete`, `Cross`, `WavyLine`, `Move` and `Copy`, as depicted in Figure C.1. Shapes are recognized independently of rotation, size or number of strokes.

The set of shapes selected and presented in Figure C.1 are the basic elements to allow the construction of technical diagrams such as electric or logic circuits, flowcharts or architectural sketches. These diagrams also require distinguishing between solid, dashed and bold depictions of shapes in the same family. Typically, architects will use multiple overlapping strokes to embolden lines in sketches, a mechanism commonly used in drawing packages. We are therefore interested in recognizing different renderings of a given shape as illustrated in Figure C.1. We are just interested in collecting qualitative differences, not in obtaining precise values for attributes, without regard to different line-styles and widths.

The algorithm works by collecting strokes from a digitizing tablet. A stroke is the set of points from pen-down to pen-up. We collect strokes into scribbles until a set timeout value is reached. Recognized scribbles are classified as shapes. A shape associates a scribble with a class (e.g. `Triangle`) and a set of geometric attributes, such as start- and end-points and bounding box. The recognizer works by looking up values of specific features in fuzzy sets associated to each shape. This process may yield a list of plausible shapes ordered by degree of certainty. If the recognizer cannot associate any shape to the scribble, it will return the `Unknown` shape, together with some basic geometric attributes, such as line-style and “openness”.

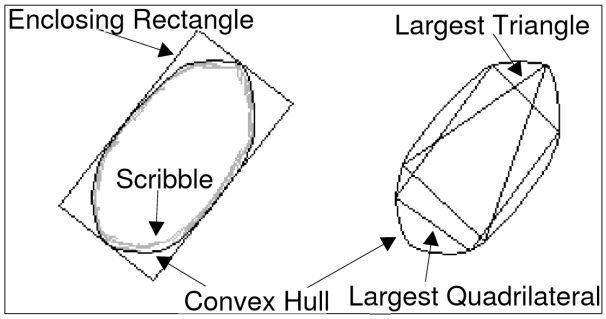


Figure C.2: Polygons used to estimate features.

C.2.1 Geometric Features

After collecting each scribble from the digitizing tablet, we compute the convex hull (ch) of its points, using Graham's scan [O'Rourke 98]. We then use this convex hull to compute three special polygons. The first two are the largest area triangle (lt) and quadrilateral (lq) inscribed in the convex hull [Boyce 85]. The third is the smallest area enclosing rectangle (er) [Freeman 75] (see Figure C.2). Finally we compute the area and perimeter for each special polygon, to estimate features and degrees of membership for each shape class.

To select features that best identify a given shape, we built percentile graphics for each feature. These graphics illustrate the statistical distribution of feature values over the different classes, extracted from sample drawings. For each shape, the solid bar spans the 25% to 75% percentiles, while the line extends from 10% to 90% of all observed values of a given feature.

Our initial selection of features takes into account specific properties of shapes to identify. Associated with these features we infer fuzzy sets from training data, which express the allowable values of a feature for a given shape. Usually one feature alone is not enough to distinguish shapes, yielding incorrect classifications. We then add extra features (with corresponding fuzzy sets) to prevent unwanted classifications, yielding more complex rules as needed. The main features we use are ratios between perimeters and areas of the special polygons described above.

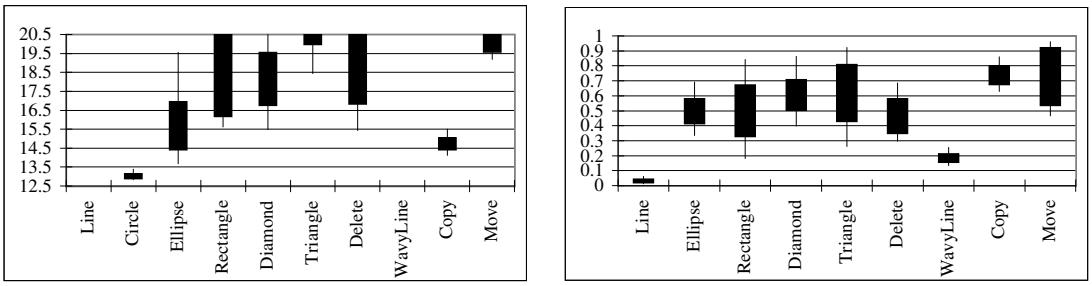


Figure C.3: Percentiles for the P_{ch}^2/A_{ch} (left) and H_{er}/W_{er} (right) ratios.

To distinguish `Circles` from other shapes we use the *Thinness* ratio (P_{ch}^2/A_{ch}), where A_{ch} is the area of the convex hull and P_{ch}^2 is its perimeter squared. The thinness of a circle is minimal, since it is the planar figure with smallest perimeter enclosing a given area, yielding a value near 4π (see Figure C.3-left). In this figure the thinness values for `Lines` and `WavyLines` lie outside the range of values indicated. We chose not to indicate these values in order to make the range of values for `Circles` more visible.

We identify `Lines` using the aspect ratio, which compares the height of the enclosing rectangle (H_{er}) with its width (W_{er}). The H_{er}/W_{er} ratio will have values near zero for lines and bigger values for other shapes (see Figure C.3-right).

In order to identify `Rectangles` we use two ratios. One relates the convex hull to the enclosing rectangle while the other measures the largest quadrilateral that fits inside the convex hull against the enclosing rectangle. For rectangular shapes the area of the convex hull will be very close to that of the enclosing rectangle (A_{er}) and this one will be very close to the largest quadrilateral's (A_{lq}). The A_{ch}/A_{er} and A_{lq}/A_{er} ratios will have values near unity for rectangles (see Figure C.4).

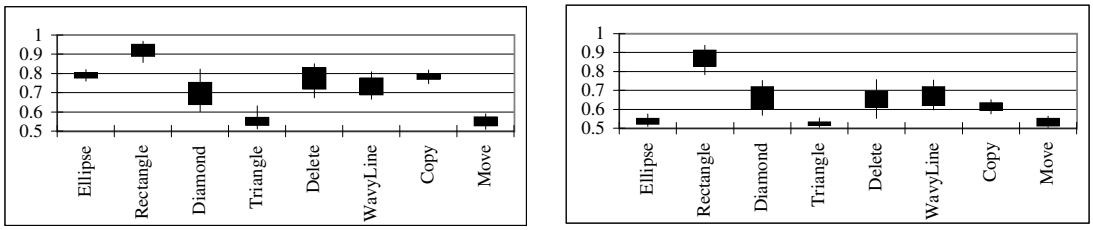


Figure C.4: Percentiles for the A_{ch}/A_{er} (left) and A_{lq}/A_{er} (right) ratios.

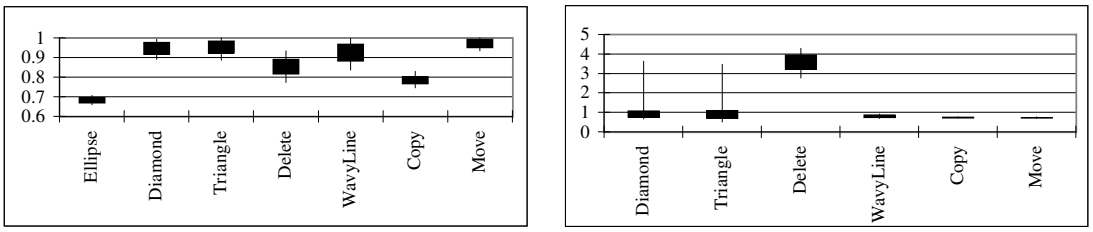


Figure C.5: Percentiles for the A_{lq}/A_{ch} (left) and T_l/P_{ch} (right) ratios.

We identify `Ellipses` by comparing the area of the largest quadrilateral to that of the convex hull. The A_{lq}/A_{ch} ratio will have values near 0.7 for ellipses and bigger values for other shapes (see Figure C.5-left). This is more robust than using A_{ch}/A_{er} which is too ambiguous.

Since this recognizer is intended for interactive use, we are interested in detecting gestures such as a set of zigzag strokes drawn in succession to signify erasing objects underneath. Using our geometry approach, we detect this pattern by comparing the total length of strokes (T_l) drawn to the perimeter of the convex hull (P_{ch}). The T_l/P_{ch} ratio has large values for the `Delete` command, and smaller values for other shapes (see Figure C.5-right).

To distinguish `Diamonds` from other shapes we divide the area of the largest triangle that fits inside the convex hull (A_{lt}) by the area of the largest quadrilateral. The A_{lt}/A_{lq} ratio has values between 0.5 and 0.6 for diamonds and bigger ones for other shapes (see Figure C.6-left).

To identify `Triangles` and `Move` gestures we compare the area and perimeter of

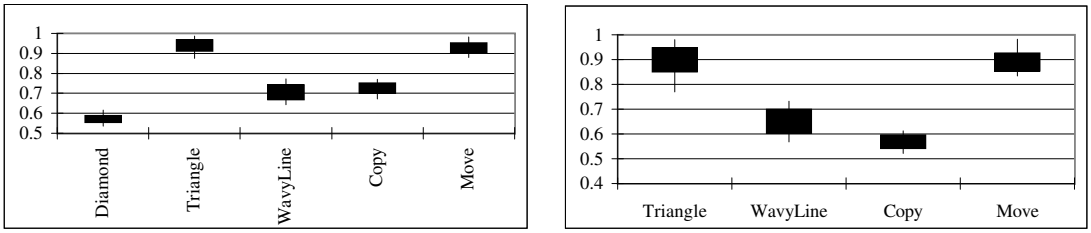


Figure C.6: Percentiles for the A_{lt}/A_{lq} (left) and A_{lt}/A_{ch} (right) ratios.

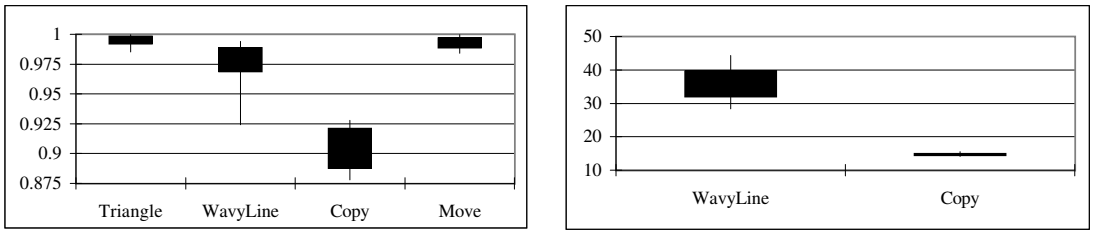


Figure C.7: Percentiles for the P_{lt}/P_{ch} (left) and P_{ch}^2/A_{ch} (right) ratios.

the largest triangle to that of the convex hull. A_{lt}/A_{ch} and P_{lt}/P_{ch} ratios will have values near unity for triangles and moves, and smaller values for other shapes (see Figure C.6-right and C.7-left). We distinguish these two shapes using the openness of the move command. The `Copy` command is recognized using the thinness ratio P_{ch}^2/A_{ch} (see Figure C.7-right). It has values near 15 while others have bigger values.

The `WavyLine` gesture has two attributes that allow the distinction from other shapes. First it is a little "fatter" than lines, so the H_{er}/W_{er} ratio has bigger values (see Figure C.3-right). Second, its total length is smaller than the total length of the `Delete` command, so the T_l/P_{ch} ratio has smaller values (see Figure C.5-right).

To distinguish `Dashed` from solid lines we use the ratio $P_{ch} \times N_s/T_l$ where N_s is the number of strokes in the scribble. Because dashed shapes have a large number of strokes, with a small total length, this ratio exhibits large values for dashed shapes and smaller values for closed shapes. The T_l/P_{ch} ratio separates `Bold` from solid lines. Values bigger than 1.5 indicate redundant strokes (i.e. Bold lines).

Recognizing shapes drawn using overlapping strokes to signify `Bold` line-style conflicts with the `Delete` command, because we use the same feature (T_l/P_{ch}) in the same manner to identify both. To solve this conflict we introduce a heuristic feature to distinguish "filled" from "hollow" shapes. Hollow shapes do not have points near the barycenter. To compute *hollowness* we first calculate a triangle, whose dimensions are roughly 60% of the largest triangle that fits inside the convex hull and shares the same barycenter. We then count scribble points lying inside the smaller triangle. Shapes like `Triangles`,

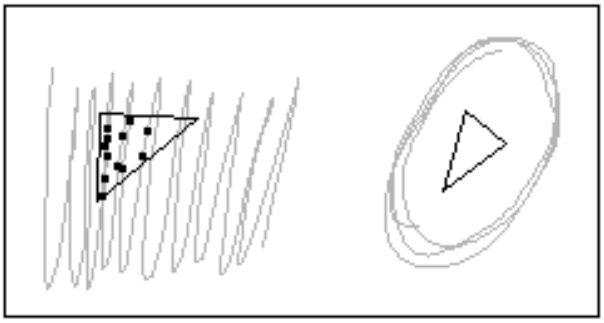


Figure C.8: Hollowness of `Delete` and `Ellipse`.

`Rectangles`, `Circles`, `Ellipses` or `Diamonds` should not have any point inside, but `WavyLines` or `Delete` gestures must have (see Figure C.8).

Figure C.9 lists all shapes identified by the recognizer and the features used by each. As described earlier, conductive rules combine assertions or negations of these features as we shall see in the next section.

	Ach/Aer	Alq/Ach	Alt/Ach	Alq/Aer	Alt/Alq	Plt/Pch	Pch/Per	Plq/Pch	Nstrokes	Hollowness	Pch2/Ach	Her/Wer	Tl/Pch	Pch*Ns/Tl
Line												X		
Arrow									X					
Triangle			X		X	X	X			X				
Rectangle	X			X						X				
Circle										X	X			
Ellipse		X								X	X			
Diamond		X		X	X					X	X			
Delete									X	X		X	X	
WavyLine									X	X		X	X	
Copy			X		X			X	X		X		X	
Move			X			X			X		X		X	X
Cross		X						X	X					

Figure C.9: Features used by each shape.

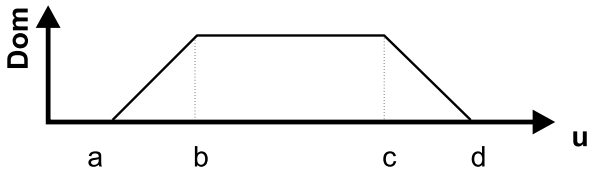


Figure C.10: Definition of a fuzzy set.

C.2.2 Deriving Fuzzy Sets from Training Data

To choose the "best" fuzzy sets describing each shape class we used a training set developed by three subjects, who drew each shape thirty times; ten times using solid lines, ten times using dashed lines and ten times using bold lines. Based on this training set we defined several ratios, combining among others the area and perimeter of the polygons described above.

Each shape is defined by several fuzzy sets, some of which identify the shape while others serve to avoid "wrong" results. We do not use the same number of features for all shapes. Some shapes require no more then one or two features. In general we tend to avoid using "too many" fuzzy sets to characterize a shape. This requires careful data analysis and experimentation to find the "right values", that is, those yielding higher recognition rates and less misrecognitions (false positives). The simplest case is that of a line, which is distinguished by thinness alone.

A fuzzy set is defined by four values, as depicted in Figure C.10. After selecting the features for each shape, we compute these four values based on the percentiles. Values b and c correspond to the 10% and 90% percentiles respectively, and a and d to the "minimum" and "maximum" after we have identified outliers in each distribution. Removing these outliers minimizes confusion between different shape families, which rises from overlap in distributions. Thus, there is a trade-off in designing fuzzy sets from statistical data. If a and d are set very wide apart the recognition rate increases, as well as the number of false positives. "Narrower" values decrease the number of false classifications at the cost of a much lower recognition rate. Abe and Lan [Abe 96] discuss an automated procedure for collecting this information using activation and inhibition hyper-boxes. We

chose to generate rules manually, since their approach is not sufficiently flexible for our purposes.

C.2.3 Deriving Results from Fuzzy Sets

After collecting input points from the tablet and computing the special polygons from scribble data, the recognizer calculates the degree of membership (dom) for each shape class. This degree is the result of ANDing together degrees of membership for the relevant fuzzy sets. In the following paragraphs we exemplify how to build rules that classify shapes. The first rule identifies `Lines` while the second defines `Diamonds`.

The rule on the left of Figure C.11 is the simplest rule used in our recognizer. It ascertains that "very thin" ($H_{er}/W_{er} \simeq 0$) scribbles should be classified as `Lines`. A more complicated rule (Figure C.11-right) recognizes `Diamonds`, where AND denotes the conjunction of fuzzy predicates $\mu_x(f \texttt{AND}\ g) = \min(\mu_x(f), \mu_x(g))$ and NOT is defined by $\mu_x(\texttt{NOT}\ f) = 1 - \mu_x(f)$.

The algorithm distinguishes between `Solid`, `Dashed` and `Bold` styles after identifying "basic" shapes. This enables us to treat line-style as an attribute orthogonal to shape, making the design of our recognizer more modular and easier to add different shapes in the future.

```
IF Scribble IS VERY THIN
THEN
     Shape IS A Line
```

```
IF Scribble IS LIKE Diamond AND
   Scribble IS NOT LIKE Ellipse AND
   Scribble IS NOT LIKE Bold Line AND
   Scribble IS NOT LIKE Rectangle
   THEN
        Shape IS A Diamond
```

Figure C.11: Example rules to classify shapes.

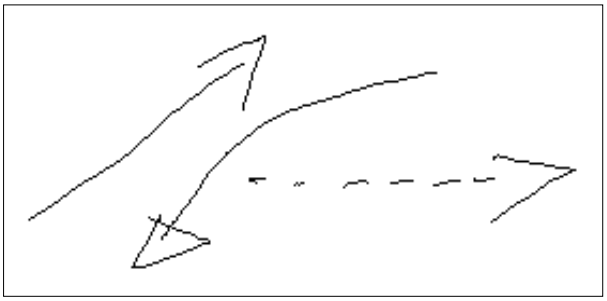


Figure C.12: Different types of `Arrows`.

C.2.4 Re-segmentation

An approach based entirely on global geometric properties has some limitations. Even though `Arrows` or `Crosses` cannot be recognized using this approach, it would be useful if our recognizer identified them, since they are commonly used in most diagram notations. In order to recognize these shapes we must look for new properties that characterize them. Among those, we can consider the small number of strokes, or the existence of a stroke that uniquely identifies the shape, or a distinct spatial relation between strokes. For example `Arrows` are built of a variable set of strokes terminated by a last stroke which is either a `Triangle` or a `Move` shape, as shown in Figure C.12. Finally, a `Cross` consists of two intersecting strokes (that must be `Lines`).

The next two rules presented in Figure C.13 show how we classify these shapes. The first rule identifies `Arrows` while the second defines `Crosses`. We analyze these new properties by re-segmenting the original scribble and by applying the recognition process to some specific strokes, e.g. the last stroke in the `Arrow`. Re-segmentation allows recognizing new gestures that could not be identified using just geometric properties.

```
IF NumStrokes >= 2 AND
   (LastStrk IS LIKE Triangle OR
   LastStrk IS LIKE Move)
THEN
     Shape IS A Arrow
```

```
IF NumStrokes == 2 AND
   FirstStrk IS LIKE Line AND
   SecondStrk IS LIKE Line AND
   FirstStrk INTERSECT SecondStrk
THEN
     Shape IS A Cross
```

Figure C.13: Rules to identify `Arrows` and `Crosses`.

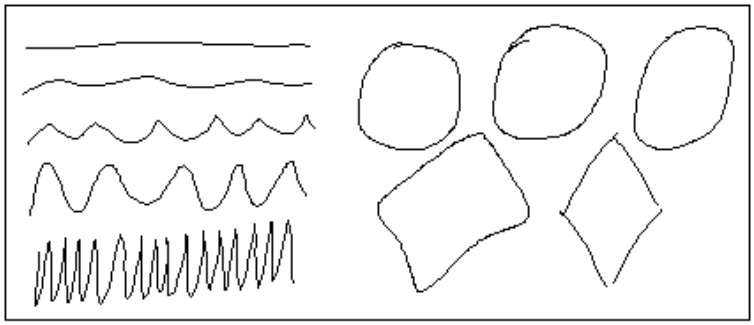


Figure C.14: Ambiguity cases among shapes.

C.2.5 Ambiguity

Considering the shapes identified by the recognizer, we present four special cases which can yield ambiguous results. Ambiguity exists between `Lines` and `WavyLines`, `WavyLines` and `Deletes`, `Circles` and `Ellipses`, `Diamonds` and `Rectangles`. These cases are presented in Figure C.14.

Humans solve this natural ambiguity between geometric shapes, by identifying more than one shape and making the final distinction based on the surrounding context or using feedback from others. The recognizer described in this paper deals with ambiguity between shapes in a similar way, i.e. when it can not uniquely identify a geometric shape, it returns a list of plausible candidates. The application can then choose the best candidate using context information. The ambiguities between shapes are modeled naturally using fuzzy logic to associate degrees of certainty to recognized shapes. Figure C.15 illustrates corresponding fuzzy sets for the ambiguous cases shown in Figure C.14.

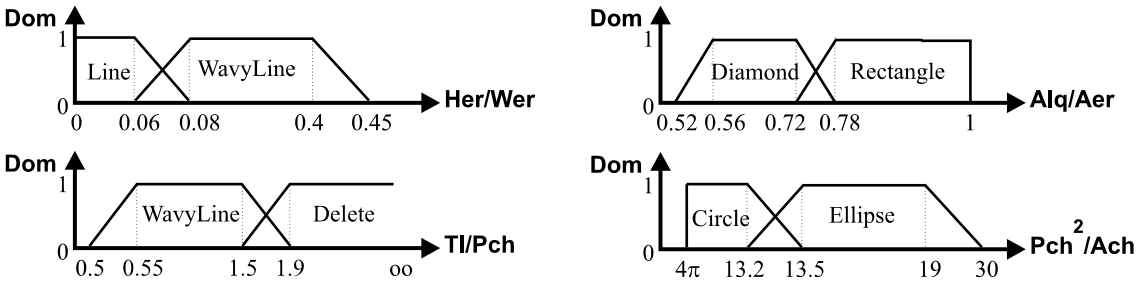


Figure C.15: Fuzzy sets representing the ambiguity cases supported by the recognizer.

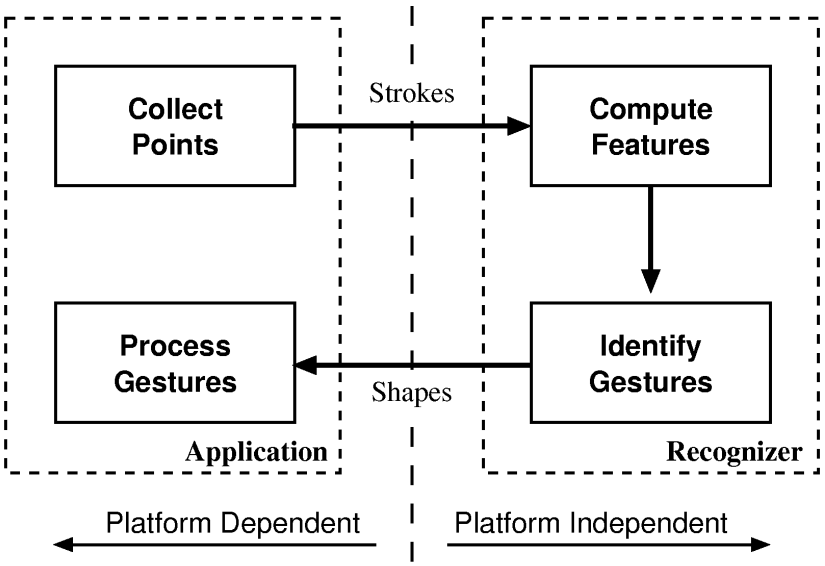


Figure C.16: A simplified version of the CALI architecture.

C.3 Architecture

The CALI library was developed to be platform independent. Actually, there are two packages available [Fonseca 00a], one for Linux and another for MS Windows.

Figure C.16 shows the main blocks of the recognizer as well as the blocks to develop on the application side. One of the blocks, on the application side, is responsible for collecting the individual points of the strokes, while the other is responsible for receiving and manipulating the shapes returned by the recognizer. The code developed on the application side is machine dependent, unless we use a graphical package, like the wxWindows or the Qt toolkit.

The first block of the recognizer receives strokes from the application and computes the corresponding geometric features. The second identifies the correct shape based on the values computed before. The recognized shapes are inserted in a list, order by degree of certainty, and returned to the application.

Figure C.17 presents the main interface classes exposed to clients of the CALI library, which we describe below.

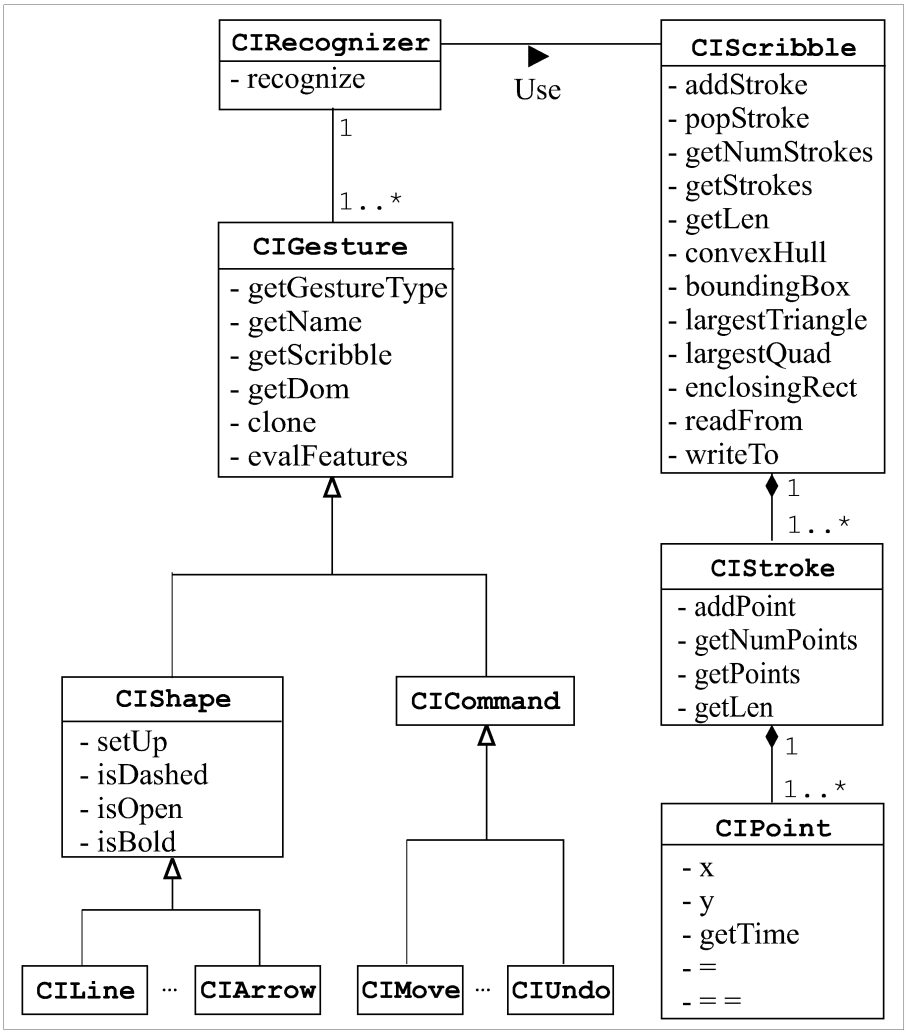


Figure C.17: Class diagram.

CIRecognizer Main component of the library that interacts directly with calligraphic applications identifying hand drawn scribbles. This class implements a recognizer of geometric shapes and gesture commands based mainly on geometric information.

CIGesture Defines all the recognized entities, shapes and commands. The objects of this class have the original scribble and the recognition degree of certainty associated to them.

CIShape A Shape is a special case of gesture, that models all geometric shapes (Line, Circle, Rectangle, etc.). All instances of this class have attributes, like open, dashed or bold, and a geometric definition (a set of points).

CICommand A special case of gesture, but without attributes or geometric definition. Commands do not have a visual representation and usually trigger an action.

CIScribble This class represents a scribble, which is built from a set of strokes. From a scribble we can compute some special polygons, like the Bounding Box, the Convex Hull, the Largest Triangle, etc., used during the recognition process.

CIStroke Defines a stroke composed by a set of points. It has methods to add points, to know the number of points, to get the points and to compute the total length of the stroke.

CIPoint Models a bidimensional point with a time stamp.

C.4 Experimental Evaluation

To evaluate the recognition algorithm, we asked nine subjects to draw each multi-stroke shape 40 times, using solid, dashed and bold lines, 30 times each uni-stroke shape and a simple Entity/Relationship diagram with 22 shapes. All these drawings yield a total of 4068 shapes. Subjects were told that the experiment was meant to test recognition, so they did not try to draw "unnatural" shapes.

We used a Wacom PL-300 LCD digitizing tablet and a cordless stylus to draw the shapes. We gave a brief description of the recognizer to the users, including the set of recognizable shapes. We also told them about the multi-stroke shape recognition capabilities and the independence of changes with rotation or size. Novice subjects had a short practice session in order to become acquainted to the stylus/tablet combination. During the drawing session the recognizer was turned off in order not to interfere with data collection and to avoid any kind of adaptation from the user.

The recognizer successfully identified 95.8% of the scribbles drawn considering just the first shape identified. It is fast: each scribble requires, on average, less than 50 ms (using a Pentium II @ 233 MHz) to be recognized, from feature vector computation to

Drawn \ Recognized	Shapes	Line	Arrow	Triangle	Rectangle	Diamond	Circle	Ellipse	Delete	WaviLine	Copy	Move	Cross	Unknown
	Line	**97.7**			0.5				1.4	0.2				0.2
	Arrow	0.5	**90.2**	0.3						0.3				8.7
	Triangle	0.3	0.8	**97.8**						0.6				0.6
	Rectangle	0.2	0.5		**96.9**	1.4		0.2						0.7
	Diamond		0.8		8.4	**87.8**		0.3	0.3	0.5				1.9
	Circle				0.3		**97.2**	2.5						
	Ellipse						1.1	**97.6**	0.5					0.8
	Delete					0.4			**98.9**					0.7
	WavyLine	2.0							2.7	**94.6**				0.7
	Copy										**99.6**	0.4		
	Move			3.0								**94.8**		2.2
	Cross												**97.0**	3.0

Figure C.18: Confusion matrix (values in percentages).

final classification.

A cursory analysis of the confusion matrix, shown in Figure C.18, reveals that `Diamonds` are often confused with `Rectangles`, and have the lowest recognition rate. `Arrows` are other shape which exhibits low recognition rate. The former is due to the ambiguity between `Rectangles` and `Diamonds` that favors `Rectangles` and the latter is due to incorrect drawing (single-stroke) of arrows by users. We can also identify other cases of confusion between shapes, such as `Circles` with `Ellipses`, `Moves` with `Triangles` and finally `WavyLines` with `Lines` and with `Deletes`. In fact, the confusion between these shapes is both an acceptable and *intuitive* behavior.

Since ambiguity is one of the main characteristics of our recognizer, we prefer to consider the top three shapes identified instead of just the most likely one. Using this, when we take this route, the recognition rate increases to 97%.

C.5 Summary

We have described a simple, fast and robust approach to recognize multi–stroke geometric shapes drawn with different line styles and rotated at arbitrary angles. Additionally,

it also identify a set of gesture commands, that can be used to trigger actions. Our intent was to provide more a means to support calligraphic interaction rather than a totally robust and "foolproof" approach to reject shapes outside the domain of interest.

The recognizer uses fuzzy rules to classify geometric shapes and introduced re-segmentation to identify higher-level patterns such as `Arrows` and `Crosses`. Additionally, it uses fuzzy logic to model ambiguities between shapes.

The high recognition rates (97%) and fast response characteristic (less than 50ms in a Pentium II) of this recognizer make it very usable in interactive applications. Finally, we have developed a library of software components, making the source code of this recognizer publicly available at `http://immi.inesc-id.pt/cali`.

Bibliography

[Abe 96] Shigeo Abe and Ming-Shong Lan. Efficient Methods for Fuzzy Rule Extraction From Numerical Data. In C. H. Chen, editor, *Fuzzy Logic And Neural Networks Handbook*, pages 7.1–7.33. IEEE Press, 1996.

[Aigrain 96] P. Aigrain, H.J. Zhang, and D. Petkovic. Content-Based Representation and Retrieval of Visual Media: A State-of-the-Art Review. *Multimedia Tools and Applications*, 3(3):179–202, November 1996.

[Allen 83] J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[Apte 93] Ajay Apte, Van Vo, and Takayuki Dan Kimura. Recognizing Multistroke Geometric Shapes: An Experimental Evaluation. In *Proceedings of the Sixth ACM Symposium on User Interface Software and Technology (UIST'93)*, pages 121–128, Atlanta, GA, USA, 1993.

[Badel 92] A. Badel, J. P. Mornon, and S. Hazout. Searching for Geometric Molecular Shape Complementary Using Bidimensional Surface Profiles. *Journal of Molecular Biology*, 10:205–211, December 1992.

[Bakergem 90] D. V. Bakergem. Image Collections in The Design Studio. In *The Electronic Design Studio: Architectural Knowledge and Media in the Computer Age*, pages 261–272. MIT Press, 1990.

[Berchtold 96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 28–39, Bombay, India, September 1996.

[Berchtold 97a] S. Berchtold, D.A. Keim, and H.P. Kriegel. Using extended feature object for partial similarity retrieval. *The International Journal on Very Large Data Bases*, 6(4):333–348, 1997.

[Berchtold 97b] Stefan Berchtold and Hans-Peter Kriegel. S3: Similarity in CAD Database Systems. In *Proceedings of the International Conference on Management of Data (SIGMOD'97)*, pages 564–567, Tucson, Arizona, USA, May 1997. ACM Press.

[Berchtold 98] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegel. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. In *Proceedings of the International Conference on Management of Data (SIGMOD'98)*, pages 142–153, Seattle, Washington, USA, June 1998. ACM Press.

[Berchtold 00a] Stefan Berchtold, Christian Bohm, H. V. Jagadish, Hans-Peter Kriegel, and Jorg Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, pages 577–588, San Diego, USA, 2000.

[Berchtold 00b] Stefan Berchtold, Daniel Keim, Hans-Peter Kriegel, and Thomas Seidl. Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):45–57, 2000.

[Bespalov 03] Dmitry Bespalov, Ali Shokoufandeh, William C. Regi, and Wei Sun. Scale-space representation of 3d models and topological matching. In *ACM Symposium on Solid Modeling and Applications*, pages 208–215. ACM, June 2003.

[Bezdek 92] James C. Bezdek and Sankar K. Pal. *Fuzzy Models for Pattern Recognition : Methods that Search for Structures in Data*. IEEE Press, NY, 1992.

[Bomze 99] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. *The Maximum Clique Problem"*, volume 4 of *Handbook of Combinatorial Optimization*. Kluwer Academy Pub., 1999.

[Boyce 85] J. E. Boyce and D. P. Dobkin. Finding Extremal Polygons. *SIAM Journal on Computing*, 14(1):134–147, February 1985.

[Bunke 98] Horst Bunke and Kim Shearer. A Graph Distance Metric Based on the Maximal Common Subgraph. *Pattern Recognition Letters*, 19:255–259, 1998.

[Bunke 02] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A Comparison of Algorithms for Maximum Common Subgraph on Randomly Connected Graphs. In *Proceedings of the IAPR Workshop on Structural and Syntactic Pattern Recognition*, volume 2396 of *Lecture Notes in Computer Science*, pages 123–132. Springer-Verlag, 2002.

[Carson 97] C. Carson. Region-Based Image Querying. In *Proceedings of the IEEE Workshop on Content-Based Access to Image and Video Libraries*, pages 42–49, San Juan, Puerto Rico, 1997.

[Chakrabarti 99] Kaushik Chakrabarti and Sharad Mehrotra. The Hybrid Tree: An index structure for high dimensional feature spaces. In *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*, pages 440–447, 1999.

[Chang 87] S. K. Chang, Q. Y. Shi, and C. W. Yan. Iconic Indexing by 2D Strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.

[Chang 99] S. K. Chang, B. Perry, and A. Rosenfeld. *Content-Based Multimedia Information Access*. Kluwer Press, 1999.

[Chen 03] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On Visual Similarity Based 3D Model Retrieval. *Computer Graphics Forum*, 22(3):223–232, September 2003.

[Chua 97] Tat Seng Chua, Kian-Lee Tan, and Beng Chin Ooi. Fast Signature-Based Color-Spatial Image Retrieval. In *Proceedings of the IEEE International Conference on Multimedia Computers and Systems*, pages 362–369. IEEE Press, 1997.

[Clayton 91] M. Clayton and H. Wiesenthal. Enhancing the Sketchbook. In *Proceedings of the Association for Computer Aided Design in Architecture (ACADIA'91)*, pages 113–125, Los Angeles, CA, 1991.

[Consortium 00] SmartSketches Consortium. SmartSketches Project (IST-2000-28169). `http://sketch.inesc.pt/`, 2000.

[Cvetković 97] Dragos Cvetković, Peter Rowlinson, and Slobodan Simić. *Eigenspaces of Graphs*. Cambridge University Press, United Kingdom, 1997.

[Davies 97] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, 1997.

[Do 95] Ellen Y. Do. What's in a Diagram that a Computer Should Understand? In *Proceedings of The Sixth International Conference on Computer Aided Architectural Design Futures (CAADF'95)*, pages 103–114. The Global Design Studio, 1995.

[Do 98] Ellen Y. Do. *The right tool at the right time*. PhD Thesis, Georgia Institute of Technology, September 1998.

[Eakins 99] John P. Eakins and Margaret E. Graham. Content-based Image Retrieval - A report to the JISC Technology Applications Programme. Technical report, Institute for Image Data Research, University of Northumbria at Newcastle, January 1999.

[Egenhofer 89] Max J. Egenhofer. A Formal Definition of Binary Topological Relationships. In W. Litwin and H. Schek, editors, *Third International Conference on Foundations of Data Organization and Algorithms (FODO'89)*, volume 367 of *Lecture Notes in Computer Science*, pages 457–472. Springer-Verlag, Paris, France, June 1989.

[Egenhofer 91] Max J. Egenhofer. Point-Set Topological Spatial Relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[Egenhofer 92] Max J. Egenhofer and Khaled K. Al-Taha. Reasoning about Gradual Changes of Topological Relationships. In A. Frank, I. Campari, and U. Formentini, editors, *Theory and Methods of Spatio-Temporal Reasoning in Geographic Space*, volume 639 of *Lecture Notes in Computer Science*, pages 196–219. Springer-Verlag, Pisa, Italy, September 1992.

[Elad 01] Michael Elad, Ayellet Tal, and Sigal Ar. Content Based Retrieval of VRML Objects - An Iterative and Interactive Approach. In *Proceedings of the sixth Eurographics workshop on Multimedia 2001*, pages 97–108, Manchester, UK, September 2001.

[Enser 93] P. Enser and C. McGregor. Analysis of Visual Information Retrieval Queries. British Library Research and Development Report, 6104, 1993.

[Fonseca 00a] Manuel J. Fonseca and Joaquim A. Jorge. CALI : A Software Library for Calligraphic Interfaces. INESC-ID, available at `http://immi.inesc-id.pt/cali/`, 2000.

[Fonseca 00b] Manuel J. Fonseca and Joaquim A. Jorge. C_{AL}I: Uma Biblioteca de Componentes para Interfaces Caligráficas. In *Actas do 9º Encontro Português de Computação Gráfica*, Marinha Grande, Portugal, February 2000.

[Fonseca 00c] Manuel J. Fonseca and Joaquim A. Jorge. Experimental Evaluation of an on-line Scribble Recognizer. In *Proceedings of the 11th Portuguese Conference on Pattern Recognition (RecPAD'00)*, pages 23–30, Porto, Portugal, May 2000.

[Fonseca 00d] Manuel J. Fonseca and Joaquim A. Jorge. Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In *Proceedings of the 9th International Conference on Fuzzy Systems (FUZZ-IEEE'00)*, volume 1, pages 291–296, San Antonio, USA, May 2000.

[Fonseca 01] Manuel J. Fonseca and Joaquim A. Jorge. Experimental Evaluation of an on-line Scribble Recognizer. *Pattern Recognition Letters*, 22(12):1311–1319, 2001.

[Fonseca 02a] Manuel J. Fonseca and Joaquim A. Jorge. Towards Content-Based Retrieval of Technical Drawings through High-Dimensional Indexing. In *Proceedings of the 1st Ibero-American Symposium in Computer Graphics (SIACG'02)*, pages 263–270, Guimarães, Portugal, July 2002.

[Fonseca 02b] Manuel J. Fonseca, César Pimentel, and Joaquim A. Jorge. CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, pages 51–58, Palo Alto, USA, March 2002.

[Fonseca 03a] Manuel J. Fonseca and Joaquim A. Jorge. Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA'03)*, pages 267–274, Kyoto, Japan, March 2003. IEEE Computer Society Press.

[Fonseca 03b] Manuel J. Fonseca and Joaquim A. Jorge. Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases.

(Extended version of DASFAA'03 paper), INESC-ID, IMMI, January 2003.

[Fonseca 03c] Manuel J. Fonseca and Joaquim A. Jorge. Towards Content-Based Retrieval of Technical Drawings through High-Dimensional Indexing. *Computers and Graphics*, 27(1):61–69, January 2003.

[Fonseca 04a] Manuel Fonseca, Bruno Barroso, Pedro Ribeiro, and Joaquim Jorge. Sketch-Based Retrieval of ClipArt Drawings. In *Proceedings of the Advanced Visual Interfaces (AVI'04) (To appear)*, Gallipoli, Italy, May 2004. ACM Press.

[Fonseca 04b] Manuel J. Fonseca, Bruno Barroso, Pedro Ribeiro, and Joaquim A. Jorge. Retrieving ClipArt Images by Content. In *Proceedings of the 3rd International Conference on Image and Video Retrieval (CIVR'04) (To appear)*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[Fonseca 04c] Manuel J. Fonseca, Bruno Barroso, Pedro Ribeiro, and Joaquim A. Jorge. Retrieving Vector Graphics Using Sketches. In Andreas Butz, Antonio Krüger, and Patrick Olivier, editors, *Proceedings of the 4th International Symposium on Smart Graphics (SG'04)*, volume 3031 of *Lecture Notes in Computer Science*, pages 66–76. Springer-Verlag, May 2004.

[Fonseca 04d] Manuel J. Fonseca, Alfredo Ferreira Jr., and Joaquim A. Jorge. Content-Based Retrieval of Technical Drawings. *Special Issue of International Journal of Computer Applications in Technology (IJCAT) "Models and methods for representing and processing shape semantics" (To appear)*, 2004.

[Freeman 75] Herbert Freeman and Ruth Shapira. Determining the Minimum-area Encasing Rectangle for an Arbitrary Closed Curve. *Communications of the ACM*, 18(7):409–413, July 1975.

[Freeman 78] H. Freeman and A. Saghri. Generalized Chain Codes for Planar Curves. In *Proceedings of the International Joint Conference on Pattern Recognition*, pages 701–703, Kyoto, Japan, November 1978.

[Funkhouser 03] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. A search engine for 3d models. *ACM Transactions on Graphics*, 22(1), January 2003.

[Gaede 98] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[Garey 79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, 1979.

[Giugno 02] Rosalba Giugno and Dennis Shasha. GraphGrep: A Fast and Universal Method for Querying Graphs. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR'02)*, pages 112–115, 2002.

[Goodrum 00] Abby A. Goodrum. Image Information Retrieval: An Overview of Current Research. *Informing Science, Special Issue on Information Science Research*, 3(2):63–67, 2000.

[Gross 95] Mark D. Gross. Indexing Visual Databases of Designs with Diagrams. In A. Koutamanis, H. Timmermans, and I. Vermeulen, editors, *Visual Databases in Architecture*, pages 1–14, Avebury: Aldershot, UK, 1995.

[Gross 96a] Mark Gross and Ellen Do. Demonstrating the Electronic Cocktail Napkin: a paper-like interface for early design. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'96)*, pages 5–6, 1996.

[Gross 96b] Mark D. Gross. The Electronic Cocktail Napkin - A computational environment for working with design diagrams. *Design Studies*, 17(1):53–69, 1996.

[Gu 95] Chuang Gu. *Multivalued Morphology and Segmentation-based Coding*. Phd thesis, Signal Processing Lab. of Swiss Federal Institute of Technology at Lausanne, 1995.

[Gudivada 97] Venkat N. Gudivada and Vijay V. Raghavan. Modeling and retrieving images by content. *Information Processing and Management*, 33(4):427–452, 1997.

[Harel 87] David Harel. StateCharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[Heesch 03] Daniel Heesch, Alexei Yavlinsky, and Stefan M. Rüger. Performance Comparison of Different Similarity Models for CBIR with Relevance Feedback. In E. Bakker, T. Huang, M. Lew, N. Sebe, and X. Zhou,

editors, *Proceedings of the International Conference on Image and Video Retrieval (CIVR'03)*, volume 2728 of *Lecture Notes in Computer Science*, pages 456–466, Urbana-Champaign, IL, USA, July 2003. Springer.

[Henrich 89] A. Henrich, H.-W. Six, and P. Widmayer. The LSD tree: Spatial access to multidimensional point and non-point objects. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB'89)*, pages 45–53, 1989.

[Henrich 98] A. Henrich. The LSDh-tree: An access structure for feature vectors. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pages 362–369, 1998.

[Hu 62] Ming-Kuei Hu. Visual Pattern Recognition by Moment Invariants. *IRE Transactions on Information Theory*, 8:179–187, 1962.

[Hu 77] M. K. Hu. Visual Pattern Recognition by Moment Invariants. In *Computer Methods in Image Analysis*, pages 113–121. IEEE Computer Society, 1977.

[Huet 01] Benoit Huet, Gennarino Guarascio, Nicky Kern, and Bernard Merialdo. Relational Skeletons for Retrieval in Patent Drawings. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'01)*, volume 2, pages 737–740, Thessaloniki, Greece, October 2001.

[Idris 97] F. Idris and S. Panchanathan. Review of Image and Video Indexing Techniques. *Journal of Visual Communication and Image Representation*, 8(2):146–166, 1997.

[Jeannin 00] S. Jeannin, L. Cieplinski, J. R. Ohm, and M. Kim. MPEG-7 Visual part of Experimentation Model Version 7.0. Technical report, ISO/IEC JTC17SC29/WG11/N3521, Beijing, China, July 2000.

[Jiang 99] Xiaoyi Jiang and Horst Bunke. Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognition*, 32:1273–1283, 1999.

[Johnson 85] M. Johnson. Relating metrics, lines and variables defined on graphs to problems in medicinal chemistry. In Y. Alavi, G. Chartrand, L. Lesniak, D. Lick, and C. Wall, editors, *Graph Theory and Its Applications to Algorithms and Computer Science*, pages 457–470. Wiley, New York, 1985.

[Jorge 94] Joaquim A. Jorge. *Parsing Adjacency Grammars for Calligraphic Interfaces*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York - USA, December 1994.

[Jorge 99] Joaquim A. Jorge and Manuel J. Fonseca. A Simple Approach to Recognize Geometric Shapes Interactively. In *Proceedings of the Third Int. Workshop on Graphics Recognition (GREC'99)*, pages 251–258, Jaipur, India, September 1999.

[Jorge 00] Joaquim A. Jorge and Manuel J. Fonseca. A Simple Approach to Recognize Geometric Shapes Interactively. In A. K. Chhabra and D. Dori, editors, *Graphics Recognition – Recent Advances*, volume 1941 of *Lecture Notes in Computer Science*, pages 266–274. Springer-Verlag, September 2000.

[Katayama 97] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the International Conference on Management of Data (SIGMOD'97)*, pages 369–380. ACM Press, 1997.

[Kauppinen 95] Hannu Kauppinen, Tapio Seppanen, and Matti Pietikainen. An Experimental Comparison of Autoregressive and Fourier-Based Descriptors in 2D Shape Classification. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 17(2):201–207, 1995.

[Kim 00] Hae-Kwang Kim and Jong-Deuk Kim. Region-based shape descriptor invariant to rotation, scale and translation. *Signal Processing: Image Communication*, 16(1-2):87–93, September 2000.

[Kimia 95] B. B. Kimia, A. Tannenbaum, and S. W. Zucker. Shapes, shocks, and deformations, i: The components of two-dimensional shape and the reaction-diffusion space. *International Journal of Computer Vision*, 15(3):189–224, 1995.

[Kounalakis 93] M. Kounalakis and D. Menuez. Defying gravity: The making of newton. Beyond Words Publishing Co, 1993.

[Kupeev 94] Konstantin Y. Kupeev and Haim J. Wolfson. On Shape Similarity. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR'94)*, volume 1, pages 227–231, 1994.

[Kupeev 96] Konstantin Y. Kupeev and Haim J. Wolfson. A New Method of Estimating Shape Similarity. *Pattern Recognition Letters*, 17(8):873–887, July 1996.

[Landay 96] James A. Landay. *Interactive Sketching for the Early Stages of User Interface Design*. PhD thesis, Carnegie Mellon University, Computer Science, Pittsburgh - USA, December 1996.

[Lau 02] Rynson Lau and Ben Wong. Web-Based 3D Geometry Model Retrieval. *World Wide Web: Internet and Web Information Systems*, 5(3):193–206, 2002.

[Leung 02] Wing Ho Leung and Tsuhan Chen. User-Independent Retrieval of Free-Form Hand-Drawn Sketches. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP'02)*, volume 2, pages 2029–2032, Orlando, Florida, USA, May 2002. IEEE Press.

[Leung 03a] Howard Wing Ho Leung. *Representations, Feature Extraction, Matching and Relevance Feedback for Sketch Retrieval*. PhD Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 2003.

[Leung 03b] Wing Ho Leung and Tsuhan Chen. Hierarchical Matching for Retrieval of Hand-Drawn Sketches. In *Proceedings of the IEEE International Conference on Multimedia and Exposition. (ICME'03)*, volume 2, pages 29–32, Baltimore, USA, July 2003. IEEE Press.

[Lin 94] K.-I. Lin, H. Jagadashi, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *The VLDB Journal*, 3(4):517–543, 1994.

[Lomet 90] D. Lomet and B. Salzberg. The hb-tree: A multiattribute indexing mechanism with good garanteed performance. *ACM Transactions on Database Systems*, 15(4), 1990.

[Lou 04] Kuiyang Lou, Sunil Prabhakar, and Karthik Ramani. Content-based Three-Dimensional Engineering Shape Search. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, pages 754–765, Boston, USA, April 2004.

[Lu 99] G. J. Lu and A. Sajjanhar. Region-Based Shape Representation and Similarity Measure Suitable for Content-Based Image Retrieval. *Multimedia System*, 7:165–174, 1999.

[Ma 95] W. Y. Ma and B. S. Manjunath. A Comparison of Wavelet Transform Features for Texture Image Annotation. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'95)*, pages 256–259. IEEE Press, 1995.

[Markey 88] K. Markey. Access to Iconographical Research Collections. *Library Trends*, 37(2):154–174, 1988.

[Mehrotra 93] R. Mehrotra and J. E. Gary. Feature-Based Retrieval of Similar Shapes. In *Proceedings of the 9th International Conference on Data Engineering (ICDE'93)*, pages 108–115, Vienna, Austria, 1993.

[Mehtre 97] B. M. Mehtre, M. S. Kankanhali, and W. F. Lee. Shape Measures for Content Based Image Retrieval: A Comparison. *Information Processing and Management*, 33(3):319–337, 1997.

[Messmer 95] B. T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. PhD Thesis, Institut fur Informatik und angewandte Mathematik, Universitat Bern, Switzerland, 1995.

[Messmer 99] B. T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern recognition*, 32:1979–1998, 1999.

[Mokhtarian 96] F. Mokhtarian, S. Abbasi, and J. Kittler. Efficient and Robust Retrieval by Shape Content through Curvature Scale Space. In *International Workshop on Image Databases and Nultimedia Search*, pages 35–42, Amsterdam, The Netherlands, 1996.

[Müller 99] Stefan Müller and Gerhard Rigoll. Searching an Engineering Drawing Database for User-specified Shapes. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'99)*, pages 697–700, Bangalore, India, 1999.

[Nabil 96] Mohammad Nabil, Anne H.H. Ngu, and John Shepherd. Picture Similarity Retrieval Using the 2D Projection Interval Representation. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):533–539, August 1996.

[Ooi 87] B. C. Ooi, R. Sacks-Davis, and K. J. McDonell. Spatial K-D-tree: An indexing mechanism for spatial databases. In *Proceedings of the 11th Annual IEEE International Computer Software and Applications Conference (COMPSAC'87)*, pages 433–438, 1987.

[O'Rourke 98] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2nd edition, 1998.

[Park 99] Jong Park and Bong Um. A New Approach to Similarity Retrieval of 2D Graphic Objects Based on Dominant Shapes. *Pattern Recognition Letters*, 20:591–616, 1999.

[Pelillo 99] M. Pelillo, K. Siddiqi, and S. W. Zucker. Matching hierarchical structures using association graphs. *IEEE Pattern Analysis and Machine Intelligence*, 21(11):1105–1120, November 1999.

[Persoon 77] Eric Persoon and King-Sun Fu. Shape Discrimination Using Fourier Descriptors. *IEEE Trans. on Systems, Man and Cybernetics*, 7(3):170–179, 1977.

[Raymond 02] John W. Raymond, Eleanor J. Gardiner, and Peter Willet. RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs. *The Computer Journal*, 45(6):631–644, 2002.

[Rubine 91] Dean Harris Rubine. *The Automatic Recognition of Gestures*. PhD thesis, Carnegie Mellon University, Computer Science, Pittsburgh - USA, December 1991.

[Rui 96] Yong Rui, Alfred C. She, and Thomas S. Huang. Modified Fourier Descriptors for Shape Representation - A Practical Approach. In *Proceedings of the First International Workshop on Image Databases and Multimedia Search*, 1996.

[Rui 99] Yong Rui, Thomas S. Huang, and Shih-Fu Chang. Image Retrieval: Current Techniques, Promising Directions, and Open Issues. *Journal of Visual Communication and Image Representation*, 10(1):39–62, March 1999.

[Safar 00a] Maytham Safar, Cyrus Shahabi, and Chung hao Tan. Resiliency and Robustness of Alternative Shape-Based Image Retrieval Techniques. In *Proceedings of IEEE International Database Engineering and Applications Symposium (IDEAS'00)*, pages 337–348, 2000.

[Safar 00b] Maytham Safar, Cyrus Shahabi, and Xiaoming Sun. Image Retrieval By Shape: A Comparative Study. In *Proceedings of the IEEE International Conference on Multimedia and Exposition. (ICME'00)*, pages 141–144, 2000.

[Sakurai 00] Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura, and Haruhiko Kojima. The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, pages 516–526, Cairo, Egypt, 2000.

[Sarkar 96] S. Sarkar and K.L. Boyer. Quantitative Measures of Change Based on Feature Organization: Eigenvalues and Eigenvectors. Technical report, Image Analysis Research Lab, University of South Florida, 1996.

[Seloff 90] G. A. Seloff. Automated Access to the NASA-JSC Image Archive. *Library Trends*, 38(4):682–696, 1990.

[Shearer 00] Kim Shearer, Horst Bunke, and Svetha Venkatesh. Video Indexing and Similarity Retrieval by Largest Common Subgraph Detection using Decision Trees. *Pattern Recognition*, 34:1075–1091, 2000.

[Shepherd 99] J. Shepherd, X. Zhu, and N. Megiddo. A fast indexing method for multidimensional nearest neighbor search. In *SPIE Conference on Storage and Retrieval for Image and Video Databases*, pages 350–355, 1999.

[Shokoufandeh 99] A. Shokoufandeh, S. Dickson, K. Siddiqi, and S. Zucker. Indexing Using a Spectral Encoding of Topological Structure. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 2491–2497. IEEE Computer Society, 1999.

[Smeulders 00] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-Based Image Retrieval at the End of the Early Years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[Smith 95] John R. Smith and Shih-Fu Chang. Single Color Extraction and Image Query. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'95)*, pages 528–531. IEEE Press, 1995.

[Smith 96] John R. Smith and Shih-Fu Chang. Automated Binary Texture Feature Sets for Image Retrieval. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP'96)*. IEEE Press, 1996.

[Striker 95] M. Striker and M. Orengo. Similarity of Color in Images. In W. R. Niblack and R. C. Jain, editors, *Proceedings of SPIE Storage and Retrieval for Image and Video Databases*, pages 381–392, 1995.

[Sutherland 63] Ivan E. Sutherland. Sketchpad: A Man-Machine Graphical Communication System. In *Spring Joint Computer Conference*, pages 2–19. AFIPS Press, 1963.

[Ulgen 95] F. Ulgen, A. Flavell, and N. Akamatsu. Geometric Shape Recognition with Fuzzy Filtered Input to a backpropagation Neural Network. *IEICE Transations Information & Systems*, E78-D(2):174–183, February 1995.

[Ullmann 76] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, January 1976.

[Weber 98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, pages 194–205, New York, USA, 1998.

[White 96a] David A. White and Ramesh Jain. Similarity Indexing: Algorithms and Performance. In *Proceedings SPIE Storage and Retrieval for Image and Video Databases*, pages 62–73, 1996.

[White 96b] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE'96)*, pages 516–523, 1996.

[Yu 01] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In *The VLDB Journal*, pages 421–430, 2001.

[Yu 04] Cui Yu, Stéphane Bressan, Beng Chin Ooi, and Kian-Lee Tan. Querying high-dimensional data in single dimensional space. *The International Journal on Very Large Data Bases*, 13(2):120–147, May 2004.

[Zhang 01] D. S. Zhang and G. Lu. Content-Based Shape Retrieval Using Different Shape Descriptors: A Comparative Study. In *Proceedings of the IEEE International Conference on Multimedia and Exposition. (ICME'01)*, pages 317–320, Tokyo, Japan, 2001.

[Zhang 02] D. S. Zhang and G. Lu. A Comparative Study of Three Region Shape Descriptors. In *In Proceedings of the Sixth Digital Image Computing ? Techniques and Applications (DICTA'02)*, pages 86–91, Melbourne, Australia, January 2002.

[Zhang 03] D. S. Zhang and G. Lu. A Comparative Study of Curvature Scale Space and Fourier Descriptors. *Journal of Visual Communication and Image Representation*, 14(1):41–60, 2003.

[Zhang 04] Rui Zhang, Beng Chin Ooi, and Kian-Lee Tan. Making the Pyramid Technique Robust to Query Types and Workloads. In *Proceedings of the International Conference on Data Engineering (ICDE'04)*, pages 313–324, 2004.

[Zhao 93] Rui Zhao. Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors. In *Proceedings of the INTERCHI'93: ACM Conference on Human Factors in Computing Systems*, pages 95–100, Amsterdam, Netherlands, 1993.