

Fachhochschule Augsburg
Fachbereich Informatik
Studiengang Multimedia (Informatik)

Diplomarbeit

Image Retrieval in the Compressed Domain Using JPEG2000

Verfasser:	Alexandra S. Teynor
Erstprüfer:	Prof. Dr. Wolfgang Kowarschick, FH Augsburg
Zweitprüfer:	Dr. Wolfgang Müller, Univ. Bayreuth
Semester:	SS2003
Abgabedatum:	7. Mai 2003

Augsburg University of Applied Sciences
Department of Computer Science
Course of Study: Multimedia (Computer Science)

Diploma Thesis

Image Retrieval in the Compressed Domain Using JPEG2000

Author:	Alexandra S. Teynor
1 st Supervisor:	Prof. Dr. Wolfgang Kowarschick, FH Augsburg
2 nd Supervisor:	Dr. Wolfgang Müller, Univ. Bayreuth
Term:	SS2003
Passed in:	May 7, 2003

“Eine Diplomarbeit zu schreiben
ist wie eine Tür zu öffnen - erst dann
sieht man wie viele Türen hinter der Tür sind.”
Christina Voigt

Image Retrieval in the Compressed Domain Using JPEG2000

by
Alexandra S. Teynor

Submitted to the Department of Computer Science,
Augsburg University of Applied Sciences

Abstract

This thesis shows the development process of JPEG2000 content based image retrieval (CBIR) methods. First, foundations of CBIR as well as JPEG2000 are explained and existing solutions are examined. On this basis, own approaches are developed.

Wavelet coefficients that are extracted from JPEG2000 are used to create different feature sets. To facilitate the development process, the Java program "JP2Feature-Finder" is developed. The distribution of wavelet subband coefficients is modelled as histogram, Generalized Gaussian Density (GGD) and Gaussian Mixture Model (GMM). In order to decide which images are similar to each other, these distributions are compared using Histogram Intersection (HI) and the Kullback-Leibler Divergence (KLD). Color and texture are treated separately. When merging the individual results, they can be weighted independently.

A plug-in for the GNU Image Finding Tool (GIFT) is implemented. Using the GIFT framework, the retrieval methods developed are tested with the TSR2500 image collection consisting of images from the Swiss TV channel Télévision Suisse Romande and ground truth data from the Viper group of the University of Geneva.

Contents

Acknowledgments	5
1 Introduction	7
1.1 Goal of this thesis	8
1.2 Outline of this thesis	8
2 Image retrieval	10
2.1 General setting	10
2.1.1 Definition	10
2.1.2 Levels of image retrieval	10
2.1.3 Semantic gap	11
2.2 CBIR systems	12
2.2.1 Common procedures	12
2.2.2 Query submission	14
2.3 Retrieval techniques	15
2.3.1 Image retrieval in pixel domain	15
2.3.2 Image retrieval in the compressed domain	16
2.3.3 Feature representation	18
2.3.4 Similarity measures	19
3 JPEG2000	23
3.1 Properties	24
3.2 JPEG2000 codec	25
3.2.1 Preprocessing	25
3.2.2 Color transform	26
3.2.3 Wavelet transform	27
3.2.4 Quantization and ranging	34
3.2.5 Tier-1 coding	35
3.2.6 Tier-2 coding	35
3.2.7 File format	36
3.3 Codec software	37

CONTENTS

3.3.1	JJ2000	37
3.3.2	Kakadu	38
3.3.3	JasPer	38
3.4	Wavelet coefficient extraction	39
3.4.1	JasPer software package	39
3.4.2	libjasper library	40
3.4.3	wcextract	40
4	Creating a JPEG2000 feature set	47
4.1	Current approaches	47
4.2	JP2FeatureFinder	48
4.2.1	Core system	48
4.2.2	Graphical user interface	50
4.2.3	Query process and evaluation	52
4.3	Feature set experiments	53
4.3.1	General setting	53
4.3.2	Texture approach	54
4.3.3	Color approach	55
4.3.4	Compound approach	58
4.4	Complexity problem	61
4.4.1	Generalized Gaussian Density	61
4.4.2	KLD for GGD	63
4.4.3	Parameter distribution	64
4.4.4	Fitting inaccuracies	65
4.4.5	Gaussian Mixture Models	70
4.4.6	KLD for GMM	74
4.5	Image quality	80
4.6	Summary	80
5	Integration in GIFT	83
5.1	GIFT	83
5.2	Configuration and startup	84
5.3	JP2 plug-in	87
5.3.1	libGIFTQuJP2	88
5.3.2	libGIFTAcJP2	91
6	Performance evaluation	92
6.1	Image databases	92
6.2	Relevance information	93
6.3	Performance measures	94
6.4	Evaluation	95

CONTENTS

6.4.1	Evaluation of texture retrieval	97
6.4.2	Evaluation of color retrieval	100
6.4.3	Evaluation of the compound approach	101
7	Conclusion	104
7.1	Summary	104
7.2	Remaining issues and future work	105
	Declaration	107
	Abbreviations	108
	Bibliography	110
	Index	114
A	Mathematical Glossary	117
B	JP2FeatureFinder	123

Acknowledgments

In the first place I would like to thank my thesis supervisor Dr. Wolfgang Müller for introducing me to the really challenging subject of CBIR. Although I have not met him in person until the day of my thesis defense, I never felt left alone. He answered all my questions via e-mail in sometimes only minutes. The phone calls we had were packed with information and good discussions. This work would not have been possible without his support.

I also want to thank my thesis supervisor Professor Dr. Wolfgang Kowarschick for supporting me with valuable advice. He helped me focussing on the topic and not getting lost in minor matters.

I would like to thank the University of Geneva and the *Viper* group for being allowed to use the TSR2500 image collection as well as the relevance information for benchmarking. Without this material, chapter 6 would not have been possible in this form.

English not being my mother tongue, Richard Teynor improved this thesis by proofreading parts of it. He familiarized me with the fine details of the English language and especially where the comma belongs.

Many thanks go to Ulrich Heisserer who helped me getting started with \LaTeX and who also pointed me to the GNU scientific library. Oliver Möller gave me helpful hints on various matters, beginning with “elements of style” and not ending with tips on how to test numerical functions. Thanks.

Fortunately life is not all about work, so many thanks go to my friends and my family. They helped me not getting lost in my work room, and offered me welcomed reasons to make a break.

I would like to thank my friend Thomas Lippert for his silent but demanding way that made me give my best. He cooked delicious meals for me while I was working on this thesis, and supported me with the infrastructure of his company Neuland Multimedia GmbH.

Although I do not hope having forgotten someone worth being mentioned, I apologize if I did. Special thanks are devoted to those people.

*Alexandra Teynor
Friedberg, May 2003*

Chapter 1

Introduction

“Wer suchet, der findet.”

This German proverb states that if someone is looking for something, he will find it. Unfortunately this is not always true. Take the internet, where the huge amount of information is simply too large to be searched manually. We have to rely on search engines like, for example, Google, Altavista or Yahoo, who help us with this task, and to hope that they will deliver the desired information. Information retrieval based on text is already quite sophisticated. But things get worse if we are not looking for a document containing a special word or phrase - if we are looking for a special image instead. Who can help us there? The search engines mentioned above usually only retrieve images with a similar name - but it is not clear what you will get when you query for an image called “game”, for example. Consider digital cameras: they produce a huge amount of pictures that can only be identified by a meaningless number. Indexing by name won't help us there either. There is the strong need for some means to retrieve images from a collection by their content, without requiring the user to browse the entire archive.

Image retrieval has been a vivid area of research in the last few years. A lot of different solutions have been developed, and a great variety of academic and commercial systems exist at present time. But we are still at the beginning of the development.

Since image retrieval is merely coming out of its infancy, most techniques used to determine image similarity still work on primitive features derived from pixel information. But images are likely to be stored in compressed form, especially

in large databases. To obtain pixel domain features the images have to be decompressed first. Of course it would be nice to avoid this overhead. Moreover, compression aims to reduce the amount of data while still containing all vital information. From this point of view it should be more efficient to use compressed data directly. Among the variety of different image formats, the emerging standard JPEG2000 seems to be well suited for that task. It uses wavelet transform among other things to reduce the amount of data. This transform contains useful properties for image retrieval. A lot of effort has been spent on developing image retrieval methods using wavelet transform, but few scientists have focussed on the special properties of JPEG2000.

1.1 Goal of this thesis

The main goal of all image retrieval efforts is to help users find what they are looking for ([44] p.21). This thesis tries to contribute knowledge to that task by developing an image retrieval method for JPEG2000 images, without decompressing them entirely.

This job consists of several tasks. First, reasonable information has to be obtained from JPEG2000 files. Then the data can be used to develop feature sets for indexing. Of course these features must be justified and tested. Complexity and storage issues have to be considered. To determine the efficiency of the retrieval method developed, the performance has to be evaluated.

1.2 Outline of this thesis

This thesis is organized in the following way:

The second chapter gives a brief overview about image retrieval. Definitions of vital concepts are given and the general setting is explained. This chapter provides the basis for later chapters.

Chapter 3 deals with JPEG2000 basics. Necessary information about the JPEG2000 codec and wavelet transform is given. Various JPEG2000 decoders are evaluated. The process of obtaining wavelet coefficients from JPEG2000 files is explained.

1.2. OUTLINE OF THIS THESIS

A file format for storing wavelet coefficients while developing the feature sets is described.

The fourth chapter builds on the two previous chapters and combines them. The development of a JPEG2000 retrieval method is shown. In the beginning previous approaches are stated, then they are adapted to the current problem and own solutions are explored. For this purpose a Java framework (JP2FeatureFinder) especially suited for that task is implemented.

Chapter 5 describes the incorporation of the resulting system into GIFT, the GNU Image Finding Tool.

In chapter 6, the performance of the retrieval methods developed is evaluated and compared to the default GIFT algorithm.

Chapter 7 concludes this thesis and shows possibilities for extending this work.

In appendix A, mathematical terms and concepts that are not directly defined in the text are described.

Appendix B shows a class diagram of the JP2FeatureFinder.

Chapter 2

Image retrieval

2.1 General setting

2.1.1 Definition

Content based image retrieval (CBIR) is widely described as:

“A technique for retrieving images from a large database on the basis of automatically derived features such as color, texture or shape.” [35, 56]

It is important to focus on the aspect of “automatically derived features” in order to distinguish CBIR from older techniques for image retrieval using keywords or classification codes, that are usually attached manually. An example for a classification code is the Art and Architecture Thesaurus (AAT) of the Getty Information Institute [1].

2.1.2 Levels of image retrieval

Image retrieval itself can be classified into three levels of increasing complexity following [35] p.7 et seq.:

2.1. GENERAL SETTING

- **Level 1:** *retrieval by primitive features*: Primitive features are features like color, texture, shape or the spatial location of objects. To generate them no additional information is needed except the image itself.
- **Level 2:** *retrieval by logical features* (also called “derived features”): Those features focus on the identity of objects. The user either wants to find an object of a particular type (“show me pictures of an elk”) or an individual object with a given name (“show me a picture of Big Ben”). Here, some additional knowledge is necessary, for example, that a particular structure has been named “Big Ben”.
- **Level 3:** *retrieval of abstract attributes*: This type of query considers the meaning, purpose, and content of an image. Examples are retrieval of images about named events (“pictures of Woodstock”), types of activity (“pictures of a football game”) or of images with a “meaning”, eg. hate, love or friendship. Of course this is a most difficult task, even human beings would often not agree about the particular meaning of an image.

Alternative expressions to describe these three different levels can also be found. For example, Schouten [49] names them “perceptual features”, “semantic features” and “psychological features”. This naming might be slightly more intuitive, since it points out that only retrieval at level 1 is based on pure perceptual properties, where as for level 2 or 3 semantic information is needed. We will use these terms interchangeably.

2.1.3 Semantic gap

To denote the gain in difficulty from level 1 to level 2 or 3 image retrieval, the term “semantic gap” is used. It describes the fact that retrieval by primitive features lacks any semantic meaning, although the user always assigns a meaning to images. So a gap between what the user wants to submit to the system and the features that he really submits exists.

Most people who are searching for a special image formulate their query by asking for logical or abstract features, for example, designers, journalists or multimedia students who need to visualize something. Currently, most systems are only capable of solving a request at level 1, aiding the user only to find visual similar pictures, but leaving him/her the task of assigning a meaning to them.

This thesis deals with low level features only and does not consider semantic meaning. Although considerable research is done to achieve semantic retrieval, we are just at the beginning of developing sophisticated image retrieval systems. Still much can be improved concerning image retrieval by primitive features. Even semantic image retrieval will always rely on primitive features to some extent, so having a good foundation in this area is crucial.

2.2 CBIR systems

Image retrieval is a complex process that comprises many jobs. Content Based Image Retrieval Systems (CBIRS) are entities that try to solve all those tasks. They should basically “help a user find the image(s) he wants from a collection of images” [44] p.21. A more formal goal is given by [54] p.22: “the *minimization of probability of retrieval error.*” This view leads to efforts to model all aspects of CBIR in terms of Bayesian decision theory.

2.2.1 Common procedures

Currently there exist quite a variety of experimental and commercial systems from several universities, research groups and companies. Some of them demonstrate their systems on the internet. A good overview about existing systems is given in [19]. An older but more detailed overview can be found in [57]. Although they all differ significantly, for example, in the retrieval techniques used, the way the user submits a query or how user feedback is evaluated, some common procedures can be described:

- The first task is to automatically extract characteristics or “features” from each image in a collection (see 2.3.1 and 2.3.2). The individual features are combined to feature sets. Those feature sets are usually computed once and then stored in a database.
- If a user wants to query, he has to provide some input. This might happen in a lot of different ways (see 2.2.2). Most commonly the user has to supply an example image. The system tries to evaluate the input, for example, by applying the same algorithms for feature extraction to the input image as

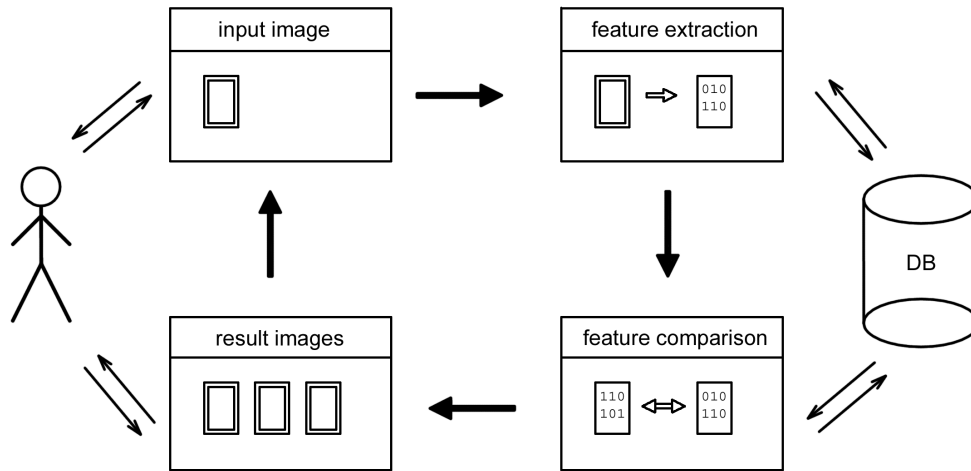


Figure 2.1: An abstract CBIRS

to the collection. If the input image is already in the database, the features might be obtained from storage.

- The features derived from the input are compared to all feature sets in the database. Special techniques like inverted files [44] can reduce complexity considerably. Usually, some similarity function is used to compare the individual features. The goal is to find the images that are most similar, or - in terms of statistics - the probability of retrieval error should be as small as possible.
- The n most similar results (according to the system) are presented to the user. Usually, the user can then refine his search by giving "relevance feedback". This means he tells the system somehow how good the retrieval process was, for example, by marking relevant or irrelevant images. The modified query is sent again to the CBIRS. The user eventually has to repeat this process some times, until the desired image is found.

Real world systems differ from this high level view on CBIRS in some or many respects. For example, not only one input image could be possible, but several images [9]. Learning from user interaction would be desirable to further improve retrieval performance, and some systems already do so [54].

2.2.2 Query submission

Before answers can be given, the right questions have to be asked. This is especially true for CBIRS. Since high level semantic queries of the form “Show me the picture with the red car next to the house” are usually not possible yet, one has to find other means of formulating a query. Possible solutions among others are:

1. The most common solution is to provide an image and ask the system to find similar ones. This technique is called “*Query by example*”. Features are extracted from the query image (if not already stored) and then compared with characteristics of the images in the database using some similarity measure. There still remains the problem of how to get the first image. A possible solution is to present a set of randomly drawn images from the collection to the user, and let him specify similar or dissimilar ones. The query is improved incrementally by giving relevance feedback. A different way to get the first images is to let the user specify preference colors using sliders or palettes (QBIC [35] p.28). The system then returns images with the same relative amounts of these colors.
2. Another, more ambitious approach is “*Query by sketch*”. Here the user is asked to draw a sketch of the particular image he is looking for. This solution has several disadvantages. First of all, most users are not too skilled in drawing something. Furthermore the same sketch can mean different things, as explained in [44] p.24 et seq.. And maybe most important: unless a sketch is almost photo-realistic, the features computed from it will differ significantly from the features derived from a natural image. Some way has to be found to make them comparable. Query by sketch is not feasible for image retrieval using JPEG2000, since it is hardly possible to describe the fine texture details that influence wavelet subband coefficients by a sketch.
3. Often it is desirable not to search for the whole image, but for a certain object or region instead. This approach is called “*Query by region*” and requires some kind of segmentation of the image. It is interesting to note that sophisticated segmentation usually has no advantage over partitioning the image in a fixed number of blocks, since automatic segmentation tends to be very crude [54] p.91.

2.3 Retrieval techniques

This is a brief overview about different techniques used in level 1 image retrieval. Feature types, feature representation and similarity measures are explained. Pixel domain indexing is described first, since knowledge gained from there can be transferred to compressed domain.

The following techniques can be applied either to a *global* or a *local* context. Global context means here that data from the whole image is used; local means that data is only taken from a certain region. To isolate different regions, some sort of segmentation has to be applied to the image. This distinction is especially important when using histograms to represent features, since then all spatial information is lost (see 2.3.3). Using a local context restores the spatial information at least to a certain extent.

2.3.1 Image retrieval in pixel domain

The first image retrieval systems directly used pixel information of images, for example, RGB values or gray level information. The most important features are color and texture since they have a great influence on the overall appearance of an image.

Color

Since color characteristics are most important to judge image similarity, they are used in almost every image retrieval system [57, 55].

When working with color, it is important to consider color representation and human color perception. Many color spaces have been developed to represent colors, where the RGB color space is the one most widely known. A color in this color space consists of three values: red, green and blue (RGB). However, this color space is not perceptually uniform. This means that equal distances in the 3D RGB color space do not yield in equal changes of color perception. For image retrieval purposes, the RGB color data is usually transformed to another color scheme that is more adapted to the human visual system, like the HSV (Hue-Saturation-Value) system. More details on color image retrieval can be found in [50].

2.3. RETRIEVAL TECHNIQUES

Texture

A texture consists of frequent repetition of similar elements called texemes ([48] p.184). Texture is extremely important if one wants to distinguish images with similar color, for example, sea and sky or grass and leaves ([35] p.24). To better analyze texture properties, pixel information is sometimes filtered. Using texture, the boundary to transform domain CBIR techniques can not be drawn clearly. They are explained in the next section. Example means for texture analysis are co-occurrence matrices ([39] p.318), Gabor filters ([44] p.60 seq.), fractals [49], Quadrature Mirror Filters as used in wavelet transform or techniques comparing the relative brightness of selected pairs of pixels (second order statistics [35] p.24). A good overview about different aspects of texture features is given by [39].

Other features

There exist a great variety of other techniques to judge image similarity. For example, it is possible to compute the *shape* of objects within an image. For this purpose the images usually have to be segmented first, for example, by thresholding. Then individual shapes can be retrieved. Techniques used are edge distribution (histograms of the directions of the edges along shape borders, [48] p.240), fourier descriptors [48] or moments ([48] p.242).

A related but not identical concept are *sketch* features. A sketch is an abstract image that contains the outline of objects ([38] p.1).

Features based on the *spatial location* of objects have also been developed. “Spatial layout is the absolute or relative position of color, texture, or shape information” ([56] p.99). For example, geographical information systems make use of this type of features.

2.3.2 Image retrieval in the compressed domain

Image retrieval in the pixel domain requires pixel information. For color photographs, this means we have to process data consisting of at least 3 channels: red, green and blue. The bit depth of a pixel component is usually 8 bit. For example, a color image of size 512x512 with a sample bit depth of 8 bit needs nearly one megabyte of storage, as opposed to maybe 100-200kB when stored in compressed form. For this reason images are seldom stored in pixel format. To avoid the need to decompress images to apply pixel domain techniques, compressed domain indexing (CDI) techniques have been developed.

2.3. RETRIEVAL TECHNIQUES

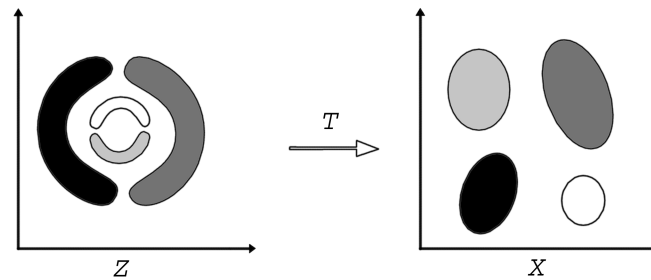


Figure 2.2: Feature transformation from one space to the other (following [54] p.45)

Ideally, CDI not only saves time, but also improves indexing performance. By definition, compression aims to avoid redundant information while still having vital characteristics of the data at hand. In this respect, compression techniques form the perfect setup for feature extraction. Vasconcelos [54] states that feature transformation only makes sense (in respect to minimizing retrieval error) if the different classes contained in the database can be distinguished more easily after transformation, as illustrated in figure 2.2.

JPEG2000 uses wavelet transform, which possesses this property. Information about frequency and space at multiple resolution levels are revealed (see 3.2.3). A variety of wavelet based retrieval techniques have been developed, for example, by Do et al. [34].

Following [38] p.3, CDI techniques can be broadly classified into two categories: transform domain techniques and spatial domain techniques.

Examples for transform domain techniques are methods based on discrete fourier transform (DFT), discrete cosine transform (DCT) and techniques based on wavelets. Since JPEG2000 uses wavelet transform, JPEG2000 indexing falls into this category.

Spatial domain techniques are mainly based on vector quantization (VQ) and fractals [38] p.8.

2.3.3 Feature representation

An important question is how to represent features obtained after analyzing. Since features are not likely to be exactly the same for two different images in the database (at least if they are not the same pictures), direct comparison can not be the goal. Imagine two images where one is a copy of the other, only slightly translated. The color information at many pixel locations will not match, although the images are of course visually very similar. Feature sets should be invariant to translation and rotation as far as possible.

For this reason, usually the distribution or probability of features is considered. For example, histograms provide an estimate for the probability density of features. Citing [54] p.34, a histogram and the corresponding feature probability density is defined as:

“The *histogram* of a collection of feature vectors \mathbf{X} is a vector $f = \{f_1, \dots, f_R\}$ associated with a partition of the feature space \mathcal{X} into R regions $\{\mathcal{X}_1, \dots, \mathcal{X}_R\}$ where f_r is the number of vectors in \mathbf{X} landing on cell \mathcal{X}_r . Assuming a feature space of dimension n and rectangular cells of size $h_1 \times \dots \times h_n$, the histogram provides an estimate of the feature probability density of the form

$$P(\mathbf{X}) = \sum_k \frac{f_k}{F} \mathcal{K}(\mathbf{x} - \mathbf{c}_k) \quad (2.1)$$

where \mathbf{c}_k is the central point of the k^{th} cell, F the total number of feature vectors and $\mathcal{K}(\mathbf{x})$ is a pdf such that

$$\begin{aligned} \mathcal{K}(\mathbf{x}) &> 0, \text{ if } |\mathbf{x}_1| < \frac{h_1}{2}, \dots, |\mathbf{x}_n| < \frac{h_n}{2} \\ \mathcal{K}(\mathbf{x}) &= 0, \text{ otherwise,} \\ \int \mathcal{K}(\mathbf{x}) d\mathbf{x} &= 1. \end{aligned}$$

Although histograms have been proven very useful in representing low level image features, they are far from ideal. They can be seen as non invertible transforms that discard information. In this special case, spatial information is lost. For example, figure 2.3 shows two visually dissimilar pictures that have exactly the same histogram.

Another problem of histograms is their high dimensionality. To capture the probability density accurately, a sufficient number of bins has to be used. If multiple

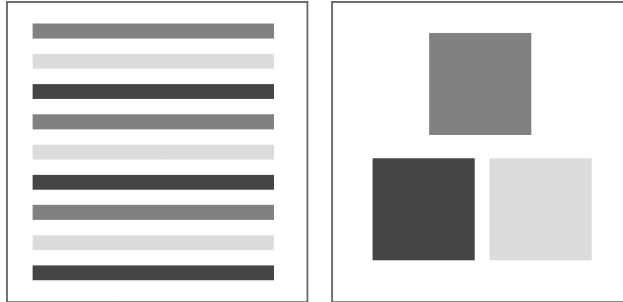


Figure 2.3: Two images with the same histogram

histograms are used to represent image features (for example, to model different subbands of JPEG2000 data) a quite large amount of information has to be stored and compared. It is desirable to model feature distribution in a less expensive way.

To solve this problem, several methods have been proposed in literature. Moments of a probability density function (pdf) can be computed to reduce complexity [37]. Gaussian Mixture Models (GMM) [54] are also a means to model a pdf with less coefficients. If the data is assumed to be Gaussian, as it is the case with wavelet subband data, Generalized Gaussian Densities (GGD) [34] can be used.

2.3.4 Similarity measures

In order to find out which images in a database are similar to a query image, the characteristics have to be compared. For this task similarity functions are used. According to Vasconcelos [54] p.18, a similarity function is a function that maps the space of image classes that compose the database into the space of possible orderings for those classes.

In other literature, these functions are very often called “distance functions”, although few suffice the definition of a distance in a metric sense.

Following [7], a distance d in the space Y is defined as:

$$d : Y \times Y \rightarrow \mathbb{R}_0^+ \quad (2.2)$$

2.3. RETRIEVAL TECHNIQUES

Often a distance is a metric, then it possesses the following properties:

1. d is **defined positive**:

$$\begin{aligned}d(x, y) = 0 &\Leftrightarrow x = y && \forall x, y \in Y \\d(x, y) > 0 &\Leftrightarrow x \neq y && \forall x, y \in Y\end{aligned}\tag{2.3}$$

2. d is **symmetric**:

$$d(x, y) = d(y, x) \quad \forall x, y \in Y\tag{2.4}$$

3. d suffices the **triangle inequality**:

$$d(x_1 + x_2, y) \leq d(x_1, y) + d(x_2, y) \quad \forall x_1, x_2, y \in Y\tag{2.5}$$

The smaller a metric distance is, the more similar two images are.

Several books and papers try to give an overview about different similarity measures, for example: [47, 48, 54]. Many of the similarity functions mentioned are no distances, but belong to the group of “non parametric test statistics” or “information theoretic divergences” [47]. The distance measures used in this thesis are:

Minkowski form distance

An example for a distance in the metric sense is the Minkowski form distance, also known as \mathcal{L}_p -norm. It has been used widely in literature and image retrieval systems.

$$\mathcal{L}_p(P_2, P_1) = \left(\int_{\mathcal{F}} |P_2(x) - P_1(x)|^p dx \right)^{\frac{1}{p}}\tag{2.6}$$

where P_1 and P_2 are probability density functions. The \mathcal{L}_1 norm is given by:

$$\mathcal{L}_1(P_2, P_1) = \int_{\mathcal{F}} |P_2(x) - P_1(x)| dx\tag{2.7}$$

For discrete pdfs (here histograms \mathcal{H} and \mathcal{R} with k bins) it is:

$$\mathcal{L}_1(\mathcal{H}, \mathcal{R}) = \sum_k |h_k - r_k|\tag{2.8}$$

2.3. RETRIEVAL TECHNIQUES

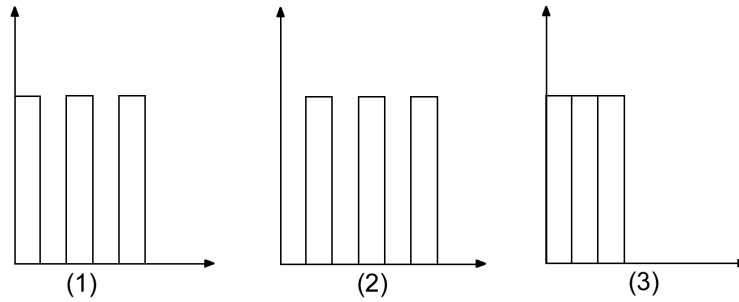


Figure 2.4: Histogram comparison where HI fails (following [48] p.199)

As proven in [48] p.196, it is the same as histogram intersection (HI) on normalized histograms ($\sum_k h_k = \sum_k r_k = 1$):

$$\mathcal{L}_{\cap}(\mathcal{H}, \mathcal{R}) = 1 - \sum_k \min(h_k, r_k) \quad (2.9)$$

Although HI has disadvantages, it is used widely due to its simplicity. Consider histograms where all values are slightly shifted, like histograms (1) and (2) in figure 2.4. Using HI, histogram (3) will be judged more similar to (1) than histogram (2), despite image (2) being visually more similar ([48] p.197).

Kullback-Leibler divergence

From a probabilistic point of view, the Kullback-Leibler divergence (KLD) is much more accurate, as shown in [54]. Assuming the feature vector x is drawn from the statistical population with density $P_i(x)$, the *Kullback-Leibler divergence* or *relative entropy* is defined as ([54] p.28):

$$KL[P_1(x)||P_2(x)] = \int P_1(x) \log \frac{P_1(x)}{P_2(x)} dx \quad (2.10)$$

This is not a metric distance, since the Kullback-Leibler divergence is not symmetrical nor the triangle inequality holds.

2.3. RETRIEVAL TECHNIQUES

When using the KLD on multiple data sets, for example, the histograms of different subbands, the overall similarity measure can be obtained using the chain rule as stated in [34]: It states that between two joint pdfs $P_1(X, Y)$ and $P_2(X, Y)$ the KLD is:

$$KL[P_1(X, Y)||P_2(X, Y)] = KL[P_1(X)||P_2(X)] + KL[P_1(Y|X)||P_2(Y|X)] \quad (2.11)$$

This means if the data sets are assumed to be independent, the resulting KLD is simply the sum of the individual KLDs [34].

Chapter 3

JPEG2000

JPEG2000 is the new image compression standard being developed by the Joint Photographic Experts Group (JPEG) [16], a committee consisting of members of the International Organization for Standardization (ISO) and the International Telecommunication Union Terminal Sector (ITU-T). It is a successor to the initial JPEG standard and addresses some weaknesses of the old standard. It also makes new features available.

The development process of the JPEG2000 standard is still under way. Only the core system, JPEG2000 Part1 (i.e. ISO/IEC 15444-1), has become an international standard in January 2001 [36]. It can be obtained from the ISO [17]. It describes minimal decoder properties and a code stream syntax. A minimal file format has also been specified. The committee is still working on 10 additional parts to the standard. Parts 2-6 will be standardized soon and deal with extensions to the core system (part 2), motion JPEG2000 (part 3), compliance testing (part 4) and reference implementations (part 5). A compound image file format is described in part 6. Work on the remaining parts (7-11) has just begun, so few results have been published. The topics of these parts are security aspects, interactive protocols and APIs, volumetric imaging and wireless applications.

The following sections describe the JPEG2000 standard only to the extent that is necessary for image retrieval. For more details the reader is referred to [53] or the official standard [17].

3.1 Properties

JPEG2000 is an image compression standard that supports both lossy and lossless compression on single and multi component images. Sophisticated compression is achieved not only for continuous tone images, but also for bi-level images (1 bit per sample). Besides superior compression performance, the goal of JPEG2000 was to provide additional features over JPEG:

1. *Random code-stream access and processing*: The code stream can be accessed at varying degrees of granularity. Compressed domain processing is possible, for example, flipping, scaling or translating the image.
2. *Good error resilience*: JPEG2000 is robust to bit errors by the inclusion of resynchronization markers, small independent code blocks and error concealing mechanisms.
3. *ROI*: Special “Regions of Interest” can be coded with higher precision.
4. *Limited memory implementation*
5. *Progressive transmission*: When an image is transmitted via a slow communication link, it can take quite a while until all data has arrived. So it is desirable to be able to display partial data obtained. The JPEG2000 format allows the code stream to be packed using several “progressions”. The data can be ordered following 4 different models:
 - progression by “pixel accuracy”: here the overall quality of the image is improved over time
 - progression by “resolution”: first a thumbnail version of the image is displayed, increasing its size when more data arrives
 - progression by “spatial location”: image data is obtained in raster scan order, i.e. information from the top of the image is transmitted before the bottom
 - progression by “image component”: first the data of a component is transmitted entirely before the data of the next component arrives.

3.2. JPEG2000 CODEC

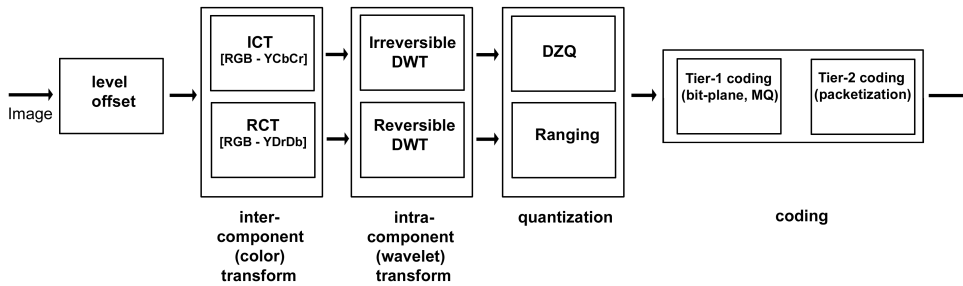


Figure 3.1: Basic structure of the JPEG2000 coding process.

3.2 JPEG2000 codec

This section gives an overview about the JPEG2000 compression/decompression process. Section 3.1 shows the main steps of JPEG2000 coding, with the decompressor basically being a mirror of the compressor structure. The baseline standard provides specific rules for the decompressor, whereas some parts for the compressor are only informative. Despite this fact, the compression process is described here, since after obtaining data from the JPEG2000 file it is still in the form of the particular compression stage.

If an image is too large to be processed at a time, tiling can be used. To achieve tiling, a tiling grid is laid over the image. Single tiles are compressed individually, the results are then combined in the code stream.

The codec has two main paths, one for irreversible compression and one for reversible compression, as can be seen in figure 3.1.

3.2.1 Preprocessing

The preprocessing stage mainly ensures that the data is in a specific format. Imagine an image with n bits per sample. The sample values get centered around zero if the data is not already in this format. To achieve this, an offset of -2^{n-1} is added. The data then has a signed representation in the range: $-2^{n-1} \leq x[\mathbf{n}] < 2^{n-1}$. The reason for this representation is that it facilitates efficient compressor implementation. Different image components are processed individually, since they might have different bit depths. If an offset was necessary, it is marked in the code stream

in order to be undone by the postprocessing stage of the decompressor.

When using irreversible processing, the n -bit sample values are mapped to real numbers in the range from $-\frac{1}{2} \leq x[\mathbf{x}] \leq \frac{1}{2}$, called the “unit range”. Values for reversible processing are not mapped.

3.2.2 Color transform

When presenting an image to the encoder containing at least 3 components with identical bit depths and sizes, the “inter-component” or “color transform” can be applied. The first 3 components are then assumed to represent the red, green and blue color information each. The RGB color information is transformed into one luminance component and two color difference components (chromatic information). The luminance component consists of all color components with the green component weighed more heavily.

Depending on the compression mode the reversible color transform (RCT) or the irreversible color transform (ICT) is applied. These transformations are described in [53] p.421 and [25] p.4.

The ICT is the same as the conventional YC_bC_r transform for image and video signals and transforms real values to real values:

$$\begin{aligned} Y &= 0.299(R) + 0.587(G) + 0.114(B) \\ C_b &= 0.564(B - Y) \\ C_r &= 0.713(R - Y) \end{aligned}$$

The RCT is used for reversible processing and maps integers to integers. It is a reversible approximation of the ICT. It is given by:

$$\begin{aligned} Y &= \lfloor \frac{1}{4}(R + 2G + B) \rfloor \\ D_b &= B - G \\ D_r &= R - G \end{aligned}$$

where $\lfloor x \rfloor$ is the floor function, which returns the largest integer not exceeding x . It is important to note that the representations D_b and D_r need one more bit of precision than the original image samples, whereas Y can still be represented using n bit.

The YC_bC_r color space was developed as part of the recommendation ITU-R BT.601 for worldwide digital component video standard [46]. It is a shifted and offset version of the YUV color model [6]. The reasons for the development of this color space were storage/transmission considerations for image data. Since the human visual system is more sensitive to changes in the brightness than in color, more bits are used to model luminance information than chrominance data (for example, in the ratio of 4:2:2 in television signals).

For image retrieval purposes, the HSV color space might be better suited, since it was designed to be perceptually uniform ([50] p.289). But YC_bC_r is still better than the RGB color model, since at least luminance information is separated from chrominance.

3.2.3 Wavelet transform

The wavelet transform is one of the core parts of JPEG2000. Using it, multi-resolution decomposition of an image is possible. After transformation, the image is described by a coarse overall shape plus additional information that holds the “details”. The wavelet transform not only analyzes frequency information as, for example, the Fourier transform does, but also retains information on time (i.e. spatial location here). This is very valuable for image retrieval. After transforming the image, it can be coded more efficiently. Since every image component is processed individually, this step in the coding process is also called “intra-component transform”.

Wavelets have become extremely popular in the last few years. There exist a great variety of wavelet classes with different properties. Much literature is available on wavelet theory and applications [32, 51, 52, 53]. A brief introduction on wavelets shall be given here:

Wavelet fundamentals

The definition follows [53] p.248 et seqq.:

Let $L_2(\mathbb{R})$ be a Hilbert space of square-integrable functions on the real line. A wavelet basis for $L_2(\mathbb{R})$ is a family of functions, $\psi_{j,k}(x)$, all derived by translation

and dilation (expansion) of a single “mother wavelet”, $\psi(x)$, according to

$$\Psi_{j,k}(x) = \sqrt{2^j} \psi(2^j x - k) \quad (3.1)$$

such that the $\Psi_{j,k}$ are linearly independent and span $L_2(\mathbb{R})$. The value of j denotes the scale of the wavelet function, k determines the translation. The factor $\sqrt{2^j}$ in equation 3.1 is used to normalize the basis. This is especially important if an orthonormal basis is to be constructed, although in a general setting wavelets do not need to follow this constraint.

An important property of wavelet functions is their local support [32]. Either they have compact support, or they fall off exponentially at infinity. This means only values near x contribute much to the transform value f of the point x . This makes the wavelet transform more useful than the Fourier transform. There the functions used (sine and cosine) have global support, so all terms of the Fourier decomposition contribute to the value f . This fact is important for image retrieval, since loss of spatial information can spoil retrieval performance (see 2.3.3).

Multi-resolution analysis

How are wavelet functions related to multi-resolution analysis? The relationship is explained using material from [29] p.9 et seq. and [51] p.2: For a multi-resolution analysis a set of nested spaces V^j , $j \in \mathbb{Z}$ which approximate $L_2(\mathbb{R})$ is needed:

$$\dots \subset V^{-1} \subset V^0 \subset V^1 \subset V^2 \dots$$

As j increases, the resolution of the functions in V^j increases. For a space V^j a set of orthonormal basis functions $\{\phi(x - k), k \in \mathbb{Z}\}$ called “scaling functions ϕ ” is defined. The scaling functions must follow certain constraints explained in [29] p.9. An important property of the spaces V^j and V^{j+1} is that they are “similar”, this means if the space V^j is spanned by $\phi_{j,k}(x)$, $k \in \mathbb{Z}$ then the space V^{j+1} is spanned by $\phi_{j+1,k}(x)$, $k \in \mathbb{Z}$. The space V^{j+1} is generated by the functions $\phi_{j+1,k}(x) = \sqrt{2} \phi_{j,k}(2x)$.

Since V^j is nested, any function in V^0 can be written as a linear combination of the basis functions $\sqrt{2} \phi(2x - k)$ from V^1 . In particular:

$$\phi(x) = \sum_k h(k) \sqrt{2} \phi(2x - k) \quad (3.2)$$

3.2. JPEG2000 CODEC

where the coefficients $h(k)$ are defined as the inner product $\langle \phi(x), \sqrt{2}\phi(2x - k) \rangle$. A new vector space W^j can be defined that is the *orthogonal complement* of V^j in V^{j+1} . This new vector space W^j can be used to represent the parts of a function in V^{j+1} that can not be represented in V^j . It is defined:

$$\psi(x) = \sum_k (-1)^k h(1 - k) \sqrt{2}\phi(2x - k) \quad (3.3)$$

The basis functions used for W^j are wavelet functions and are constructed following equation 3.1.

In terms of signal processing, $\psi(x)$ can be expressed:

$$\psi(x) = \sum_k g(k) \sqrt{2}\phi(2x - k) \quad (3.4)$$

The sequences $\{h(k), k \in \mathbb{Z}\}$ in equation 3.2 and $\{g(k), k \in \mathbb{Z}\}$ in equation 3.4 are called “Quadrature Mirror Filters” in the terminology of signal analysis ([29] p.10). The relationship between h and g is given by:

$$g(k) = (-1)^k h(1 - k) \quad (3.5)$$

Again in signal processing language, the sequence $h(k)$ is known as the *low pass* or *low band* filter, while $g(k)$ is called *high pass* or *high band* filter ([29] p.10).

Filter banks

Block transform of data has the disadvantage that data is processed in individual blocks ([53] p.160 et seqq.). So correlation between neighboring blocks cannot be exploited (which would make sense for natural images). Filter banks do not have this restriction, being a convolutional transform. A filter bank has a structure as shown in figure 3.2.

The source sequence of the image is filtered by using m distinct analysis filters and then down sampled by the factor m . The down sampling operation discards

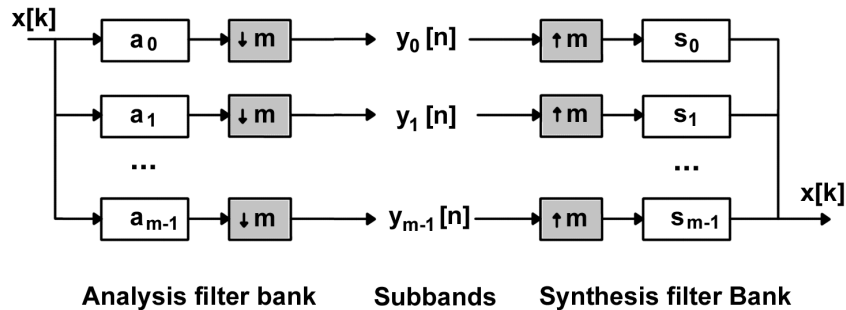


Figure 3.2: Filter bank realization of a convolutional transform and its inverse (following [53] p.164)

every m^{th} value. If $m = 2$ we obtain a low-pass version of the input data and a high-pass portion. Quadrature Mirror Filters as described above can be used to filter the signal.

Discrete Wavelet Transform

The goal of the Discrete Wavelet Transform (DWT) is to obtain a multi-resolution analysis of the original data. Wavelet functions and filter banks can be used for that task.

The DWT can be described as a “dyadic tree-structured subband transform” ([53] p.247), where dyadic means $m = 2$. The tree structure is obtained by recursively applying filter banks to the low-pass portion of the data, which is a minimized (down-sampled) version of the original data. The high-pass portion contains the information to reconstruct the original data. Figure 3.3 illustrates the resulting tree structure.

2D Discrete Wavelet Transform

So far the DWT was applied only to one-dimensional data. Images are more likely to be seen as two dimensional functions. To construct a 2D wavelet transform, the 1D wavelet transform is applied to the rows and the columns of the image, so

3.2. JPEG2000 CODEC

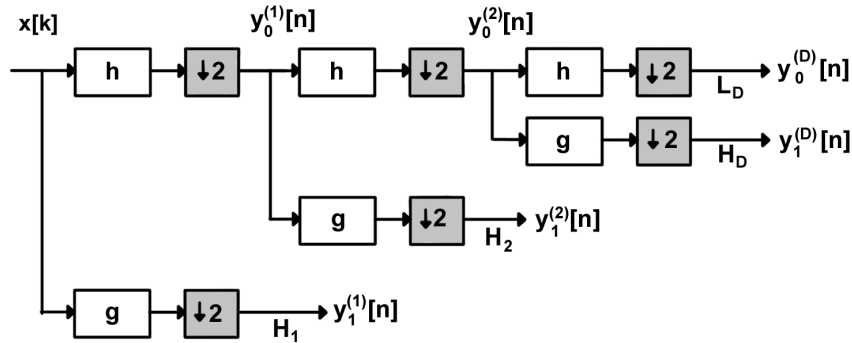


Figure 3.3: Analysis filter bank of a one dimensional tree structured subband transform (following [53] p.179)

four subbands are obtained at each decomposition level. Only the subband that is low-pass filtered twice is processed further. The process is illustrated in figure 3.4.

The resulting subbands are named according to the filter operation that was performed on them:

- LL_D : Horizontal and vertical low-pass filtering is performed. The resulting image is a reduced size version of the original image. The other subbands hold the data to reconstruct the original image.
- HL_D : Horizontal high-pass, vertical low-pass filtering. This operation most strongly reacts to vertical structures of the image.
- LH_D : Horizontal low-pass, vertical high-pass filtering. Here the horizontal lines and edges are most strongly responded to.
- HH_D : Horizontal and vertical high-pass filtering. Diagonal elements of the image have strongest influence.

Only the LL subbands are further decomposed for D levels. JPEG2000 allows D to be in the range of $0 \leq D \leq 32$, but D is usually in the range of 5-6.

A more graphical representation of the tree structure is shown in figure 3.5.

3.2. JPEG2000 CODEC

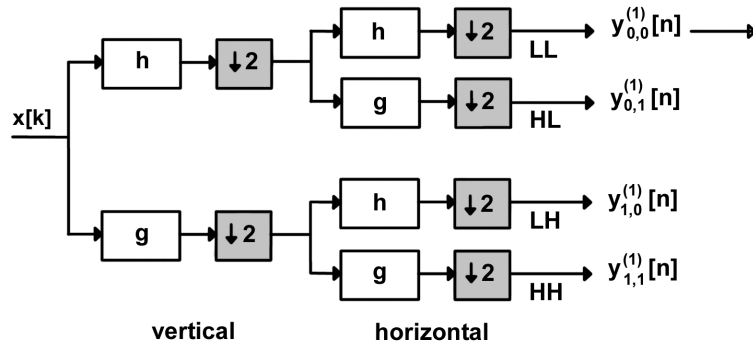


Figure 3.4: Filter bank structure for a 2D tree structured subband transform (following [53] p.180)

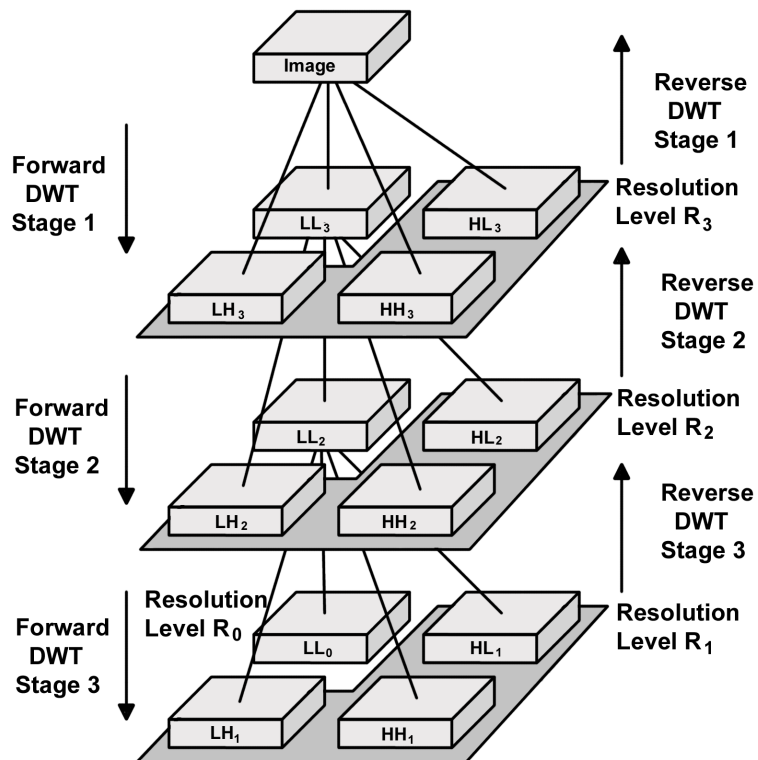


Figure 3.5: Three level, two dimensional DWT (following [53] p.330)

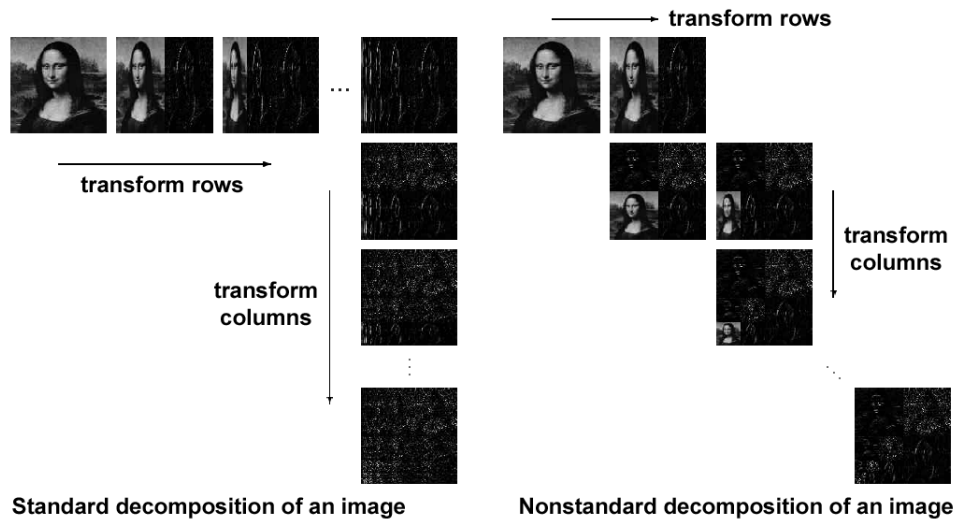


Figure 3.6: Different decomposition schemes (from [51])

It is not important in which order those transformations are performed, as long as all steps are fully executed, as shown in figure 3.6.

Wavelet coefficient distribution

As described above, the wavelet coefficients obtained after transforming an image hold the detail information that was lost in low-pass filtering. When the input signal is rather flat, i.e. the gradient is small, the wavelet coefficients are small too, since the filtered signal differs only slightly from the input signal. Since natural images usually contain many smooth regions, most coefficients are close to zero or even zero. The more textured an image is, the more wavelet coefficients will have a high absolute value. It turns out that the coefficients are gaussian-like distributed. The exact shape depends on the input signal.

Wavelets in JPEG2000

In the JPEG2000 standard only two different wavelet analysis kernels are described. However, in Part 2 of the standard it should be possible to use different wavelet kernels.

For irreversible processing the CDF 9/7 kernels are used. They belong to the first member of the Cohen-Daubechies-Feauveau family of odd-length linear phase wavelets with maximal regularity ([53] p.434).

The DWT transform used for reversible processing is derived from the 5/3 spline transform ([53] p.435). Here integer values are mapped to integers and can be used for lossless and lossy coding.

3.2.4 Quantization and ranging

Quantization is one of the main steps to reduce image data by ensuring that only minimal precision is used to represent wavelet coefficients according to the desired compression rate. Quantization is only applied to the irreversible path, since it discards information. After the DWT is applied, some data samples might exceed the nominal range of $-\frac{1}{2}$ to $\frac{1}{2}$. The parameter G (“guard”) accounts to that fact and defines larger bounds: all data samples might now be in the range of $-2^{G-1} < y_b[\mathbf{x}] < 2^{G-1} \forall b$. The value of G is usually 1 ([53]p.437) and is recorded in the code stream marker segment. After the new data range is determined, the quantization process can start. The coefficients are quantized per subband using deadzone scalar quantization. This is described in detail in [53] p.111 et seqq. Compression rate depends on the the quantizer step size. A different quantizer step size may be used for every subband. The step sizes are chosen in conjunction with rate control and are again recorded in a marker segment.

If the reversible path is used, no quantization is performed. Instead, quantizer step sizes are fixed to 1. Since in the reversible path transform data may also exceed the original precision of 2^B bit, ranging has to be performed though. The transform data range depends on the type of the subband. The nominal gains X_b of the analysis kernels are ([53]p.441):

$$X_{LL_d} = 0, X_{LH_d} = X_{HL_d} = 1, X_{HH_d} = 2$$

We see that every time high-pass filtering is done, the maximal range of the transform data is enlarged by a bit. Extra guard bits might be added to deal with out of range data. G is again stored in the code stream marker segment. The resulting maximal data range is then:

$$-2^{B-1+X_b+G} < y_b[n] < 2^{B-1+X_b+G}$$

3.2.5 Tier-1 coding

In order to perform tier-1 coding, the subband data has to be partitioned into code blocks as described in [53] p.458 et seqq. The EBCOT (Embedded Block Coding with Optimal Truncation) paradigm used in JPEG2000 relies upon relatively small code blocks that can be coded independently. The height and width of a code block must be a power of 2 and the product of the sides may not exceed 4096. The partition in code blocks depends on another, superimposed partition called “precincts”. Precincts must consist of an integer of code blocks, so code blocks have to be arranged to meet this requirement. The precinct partition does not affect coding, but is important for the packet organization in tier-2 coding.

Once the data is partitioned, coding can start. Each code block is independently coded by a bit plane coding technique similar to SPIHT (set partitioning in hierarchical trees, [53] p.313 et seqq.). In contrast to SPIHT, where two coding passes are applied, JPEG2000 uses three coding passes: a) significance pass b) refinement pass and c) cleanup pass. The purpose of these different coding passes is explained in detail in [26].

At each coding pass, the generated symbols can be entropy coded, using arithmetic coding, more specifically the MQ coder of the JBIG2 standard. This process is described in [53] p.473.

3.2.6 Tier-2 coding

The main purpose of tier-2 coding is to organize the data into packets. These packets are then combined and form the final code stream. A packet consists basically of a header and a body. In the header, information is stored to recover the data correctly. The body part contains - if not empty - coding pass data. Which data is contained exactly in the packet is determined by the various partitions of the image in tiles, components, resolution levels, subbands, precincts and quality layers. All these partitions have different purposes, for example, they ensure the EBCOT-paradigm works (quality layers). The packets can be arranged by using one of the progression types described.

All packets of an individual tile (if no tiling is applied the packets of the whole image) are grouped together to an entity. A tile header is added and contains additional coding information. All those “tile-data-groups” are added to the code

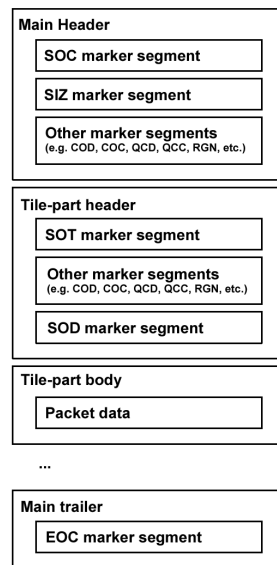


Figure 3.7: Code stream structure (following [25] p.13)

stream. If more tiles exist, it is also possible to interleave tile data, then a tile part header is added to each sub-group to denote the tile they are belonging to. The code stream itself also has a header, containing information about the size and the coding defaults of the following data, for example.

The header consist of several marker segments signaling either coding information or the begin of data. E.g the SOC marker segment indicates “start of code stream”. The SIZ marker segment contains image and tile sizes. Figure 3.7 shows the basic structure of the code stream.

For a detailed description on code stream organization, the reader is referred to [53] p.525 et seqq. or the final standard [17].

3.2.7 File format

The code stream described in the last section is sufficient to store and decode image information. However the JPEG2000 standard specifies a minimal file format being able to provide additional information some applications may require, for example, specification of the color space or ownership information. The file

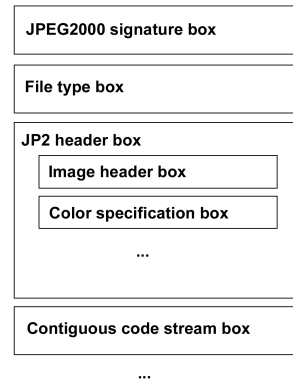


Figure 3.8: File format structure (following [25] p.14)

format wraps the code stream. The file format header consists of a number of boxes having a specific format, and containing different information. The first box supplied is the JPEG2000 signature box indicating that the following file is a JPEG2000 file. The exact purpose of the different boxes and their possible ordering is explained in [53] p.573 et seqq. In figure 3.8, an example file format is displayed.

3.3 Codec software

In order to be able to use JPEG2000 files for image retrieval, a way had to be found to extract the wavelet coefficients from those files. This means essentially that packetization, coding and quantization had to be undone. To improve development speed, existing software should be modified to obtain the data. The following implementations were tested concerning their suitability:

3.3.1 JJ2000

JJ2000 is a JPEG2000 coder/decoder that was developed by the Canon Research Centre France (CRF), the Swiss Federal Institute of Technology (EPFL) and ERICSSON Research. It is implemented in Java and is an official reference imple-

3.3. CODEC SOFTWARE

mentation for Part 5 (reference software) of the JPEG2000 standard. It conforms to the JPEG2000 final committee draft. The software is well documented using javadoc. It only supports the following file formats: PGM(raw), PPM(raw) and PGX. To immediately display decoded data an embedded viewer can be used.

However, the project was officially stopped in September 2001, the last official release being 4.1. A non official version 5.1 was developed by Eastman Kodak to fix some file format incompatibilities.

Since this codec is not developed further, it is not a suitable choice concerning the future development process of JPEG2000. More information on JJ2000 can be obtained from [15].

3.3.2 Kakadu

Kakadu is a C++ implementation of the JPEG2000 standard. It was written by David Taubmann, who is also one of the authors of [53]. In fact the Kakadu software tested was on a CD added to the book. According to the author, it fully conforms to the ISO/IEC 15444-1 standard. It also provides an image viewer (`kdu_show`). Several other features are included in this software package, for example, a Java native interface for Kakadu or a JPEG2000 internet protocol. Many useful comments were added to the source code.

This was by far the most “commercial” system to be found. High license costs must be paid in order to be allowed to use and redistribute the code.

More information on Kakadu can be obtained from [18].

3.3.3 JasPer

This coder/decoder software is written in the C language by Michael Adams. On his home page [14] not only the software is available, but there are also some other useful documents about JPEG2000 [25, 26]. The development of JasPer was supported by Image Power, Inc. and the National Sciences and Engineering Research Council of Canada. In addition to the coder/decoder software an image comparison program and a viewer is provided. JasPer supports a large range of image formats, for example, ppm, bmp, jpeg, ras or pgx. Like JJ2000, JasPer

3.4. WAVELET COEFFICIENT EXTRACTION

is an official reference implementation for the JPEG2000 standard part 5. The developer documentation is restricted to very few comments in the source code, but the structure of the code is well organized. The software is distributed freely when the provided copyright notice is added. JasPer was chosen to be the right framework to develop the wavelet coefficient extraction program.

More information on JasPer can be obtained from [14].

3.4 Wavelet coefficient extraction

Once JasPer was found to be the right codec, the JasPer framework was analyzed in more detail, examining the code and having [26] as only reference.

3.4.1 JasPer software package

The software consists of a library (`libjasper`) providing most functions and 4 application programs using the library. For full functionality, the JasPer package depends on some additional software. The free IJG JPEG Library [12] is needed to convert jpeg images, and the viewer provided by JasPer depends on the OpenGL and GLUT libraries [22]. These packages have to be obtained and installed first if the specific functionality is desired.

The main application program is `jasper`: it can transcode images from any supported format to another. It is not restricted to de-/encode JPEG2000 images. Various input/output options allow the user to control the coding/decoding process. For example, the compression rate, code block size, tiling properties or the number of resolution layers can be specified when encoding a JPEG2000 image. Jasper was used to generate the test files for experimenting and testing.

A 5th application program was added to the package: the `wcextract` utility. It takes a JPEG2000 file as an input and generates a file with the wavelet coefficients of the image as an output. Of course the data produced by `wcextract` can also be fed directly into another program generating features for the specific file.

3.4.2 libjasper library

The libjasper library consists of two main parts: first, the base/core code that provides a framework for coding/decoding. Second, it provides a number of codec drivers that handle coding/decoding for a specific format. Jasper being that modular, it was sufficient to look at the base code and the files concerned with JPEG2000 decoding.

3.4.3 wcextract

For building the `wcextract` program, the `jasper` program was taken as an example. When transcoding an image from JPEG2000 to any other format, the JPEG2000 data is decoded first. The image is then represented in an internal format: the struct `jas_image_t` defined in `jas_image.h`. This struct contains, for example, information about the image size, components, color model and of course the component data. After the image is obtained in this intermediate format, it can be coded into another representation.

The path of decoding the image was followed until quantization was undone and the reversible DWT should be applied. This is where the data is in the right format. A way had to be found to stop the decoding process there and return the data. In the following, the original decoding process is described up to that point. Afterwards the changes made to obtain the data are explained.

As already known from the introductory section about JPEG2000 all data is contained in the JPEG2000 code stream. The `jp2` file format is just a wrapper around it. So first the file format encoding has to be undone. The main function for decoding the file format is:

```
jas_image_t *jp2_decode(jas_stream_t *in, char *optstr)
```

It can easily be seen that this function gets a data stream and input options (for example, user specified decoding information) and returns a pointer to the `jas_image_t` struct needed for further processing. All information provided by various file format boxes are stored in a `jp2-decoder` struct (`jp2_dec_t`). Necessary information and the ordering of the boxes is checked.

3.4. WAVELET COEFFICIENT EXTRACTION

The most important box concerning decompression is the box that marks the beginning of the code stream ("contiguous code-stream box"). When that box is found, JPEG2000 code stream decompression starts. Again there exists a main entry point for JPEG2000 code stream decoding:

```
jas_image_t *jpc_decode(jas_stream_t *in, char *optstr)
```

A decoder struct is created (`jpc_dec_t`) in order to store intermediate data and to keep track of the decompression state. The decoder object contains a reference to the input data stream and the options. Using the function

```
int jpc_dec_decode(jpc_dec_t *dec)
```

the code stream decompression process starts. Essentially this is a loop that searches for the next marker segment, checks if it is valid at that point and then performs an associated action. For example, coding parameters are retrieved and then stored in the decoder object.

```
/* Process the marker segment. */
if (mstabent->action) {
    ret = (*mstabent->action)(dec, ms);
} else {
    /* No explicit action is required. */
    ret = 0;
}
```

As already mentioned, the data is encoded on a tile by tile basis. Every tile header or tile part header ends with a SOD (start of data) marker segment, which indicates that individual data packets follow. When the SOD marker segment is retrieved, the following function is called:

```
int jpc_dec_process_sod(jpc_dec_t *dec, jpc_ms_t *ms)
```

If the code stream consists of whole tiles, the decoding process for the current tile can start. Otherwise tile parts exist and the code stream is processed further until

3.4. WAVELET COEFFICIENT EXTRACTION

all parts of a particular tile are retrieved. The tile data is collected/stored in the struct `jpc_dec_tile_t`. It contains data from all components and all subbands of a tile. The decoding of a tile is done by the function:

```
int jpc_dec_tileddecode(jpc_dec_t *dec, jpc_dec_tile_t *tile)
```

Here all steps of the coding process can be nicely seen:

First tier-1 coding is undone:

```
jpc_dec_decodeblks(dec, tile);
```

Then for every subband of a tile dequantization is performed:

```
jpc_dequantize(band->data, band->absstepsize);
```

Now we have reached the point where the data is exactly in the right format: it contains wavelet coefficients. The next step of the function `jpc_dec_tileddecode` would be to perform the inverse wavelet transform and the inverse color transform if necessary.

To be able to stop at that point, the existing functions were modified. Usually, right after decoding image information, the decoder object and all associated information is destroyed. Only the data in `jas_image_t` is returned. My approach was not to return the decoded image, but the decoder object itself. All coding parameters (for example, data precision, number of guard bits etc.) are stored there including the intermediate data (the wavelet coefficients).

In the following, the modified functions are explained. To distinguish them from the original version, the suffix `_wc` (for wavelet coefficients) was added to the name of the corresponding libjasper functions.

```
jpc_dec_t *jp2_decode_wc(jas_stream_t *in, char *optstr);
```

Similar to the original functions, the following functions perform necessary decoding tasks plus returning a pointer to the decoder object `jpc_dec_t`:

```
jpc_dec_t *jpc_decode_wc(jas_stream_t *in, char *optstr)
```

```
int jpc_dec_decode_wc(jpc_dec_t *dec)
```

3.4. WAVELET COEFFICIENT EXTRACTION

While decoding the code stream in `jpc_dec_decode_wc`, some marker segments have to be processed differently:

```
/* Process the marker segment. */
if (mstabent->action) {
    if(ms->id == JPC_MS_SOD) {
        //-----> process start of data in a different way
        ret = jpc_dec_process_sod_wc(tile_mem,dec,ms);
    } else if(ms->id == JPC_MS_EOC) {
        //-----> process end of code stream in a different way
        ret = jpc_dec_process_eoc_wc(dec,ms);
    } else {
        //in all other cases process as usual
        ret = (*mstabent->action)(dec, ms);
    }
} else {
    /* No explicit action is required. */
    ret = 0;
}
```

SOD: As it was the case in the original SOD processing, now tile or tile part data is available. The function `jpc_dec_process_sod_wc(dec, ms)` calls

```
jpc_dec_tileddecode_wc(dec, tile)
```

if all data of a tile is available. Otherwise it returns until all tile part data is obtained. The function `jpc_dec_tileddecode_wc(dec, tile)` stops after de-quantization and returns to the calling function, leaving the transform coefficients untouched.

EOC: The end of code stream marker signals the end of the data. There usually cleanup operations like the destruction of the decoder structure are performed. Since this object is needed to retrieve the wavelet data, it is not destroyed in

```
jpc_dec_process_eoc_wc(dec,ms)
```

but returned to the calling functions. Once the decoder structure is obtained, the wavelet coefficients can be processed. Three options are available at the moment:

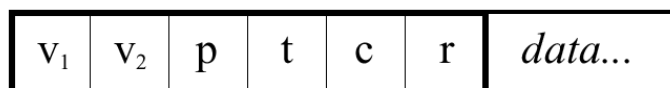
3.4. WAVELET COEFFICIENT EXTRACTION

1. **Generate a wavelet coefficient file:** The wavelet coefficients can be written to a file in order to be used by other applications. A specific file format provides additional information like the number of bits used to represent the wavelet coefficients. They can be specified in the program.

For this thesis only reversible processing was used, excluding the distortion introduced by irreversible processing. As already described in the introductory part about JPEG2000 (more specifically in 3.2.2 and 3.2.4), the precision of the wavelet coefficients in the reversible path is bigger than the original sample bit depth. The maximum precision used can be computed using information obtained from the decoder object.

Due to the transformations performed by the JPEG2000 encoding process different subbands have different value ranges (see 3.2.4). In order to facilitate later processing, they were made equal to another. The first try was to map the individual values to a specific range. But it turned out that this is not satisfactory. The maximum values possible might be as big as the decoder states, but this is only true for very rare values. The great majority of values is very close to 0. So mapping the values to a range of, for example, the original sample bit depth (which is sufficient to represent most values) squeezes most values even more around zero. So a different approach was taken: the values exceeding the specified range were clipped to the nearest representable value. This produced much better data having maximum precision in the area around zero. Only few values in very textured images have to be clipped.

The file format used for the `wc` files looks like this: it has a header part and a body where subband data packets follow. Every header box is 32 bit.



where

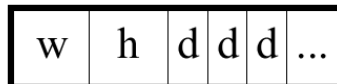
- v is the format version number of the file. It is divided into a major version number v_1 and a minor version number v_2 . The version number used at the time of this writing is 1.0.
- p denotes the precision used to represent the wavelet coefficients. The default value is 8, but larger precisions are also possible.
- t is the number of tiles used in the image. Currently only files with no tiling can be processed reasonably by the software developed. When files with more tiles are used, only the first tile is considered.

3.4. WAVELET COEFFICIENT EXTRACTION

- c tells how many components are in the image. For RGB color images the value is 3.
- r is the number of resolution levels of the wavelet transform.

After the header the data follows. The subbands are written one after the other, starting with subband LL_0 of the first component of the first tile. At larger resolutions the ordering of the bands is as follows: HL_d, LH_d, HH_d . All data from one component is written consecutively, starting with component 0.

The subband entities have their own header denoting the height (h) and the width (w) of the specific subband.



The precision of the data is denoted by the p box of the file header. At the moment p is set to 8 bit, since most images use a precision of 8 bit per color component sample. This bit depth is sufficient to represent most values and little enough to avoid to create too large wc files.

2. **Generate a portable gray map:** A graphical representation of the hierarchical decomposition of an image is produced. The wavelet coefficients of the different subbands are displayed as an image. Figure 3.9 shows an example.
3. **Generate a RGB image from subband LL_0 :** The data from subband LL_0 of all three components represents a minimized version of the original image. The data can be put together again and then transformed to the RGB space. An example for an original and a minimized image can be seen in figure 4.5.

After processing, cleanup operations have to be performed, since the destruction of the decoder information was delayed to obtain the data:

```
// necessary clean up:  
wcCleanUp(dec_jpc);
```


3.4. WAVELET COEFFICIENT EXTRACTION

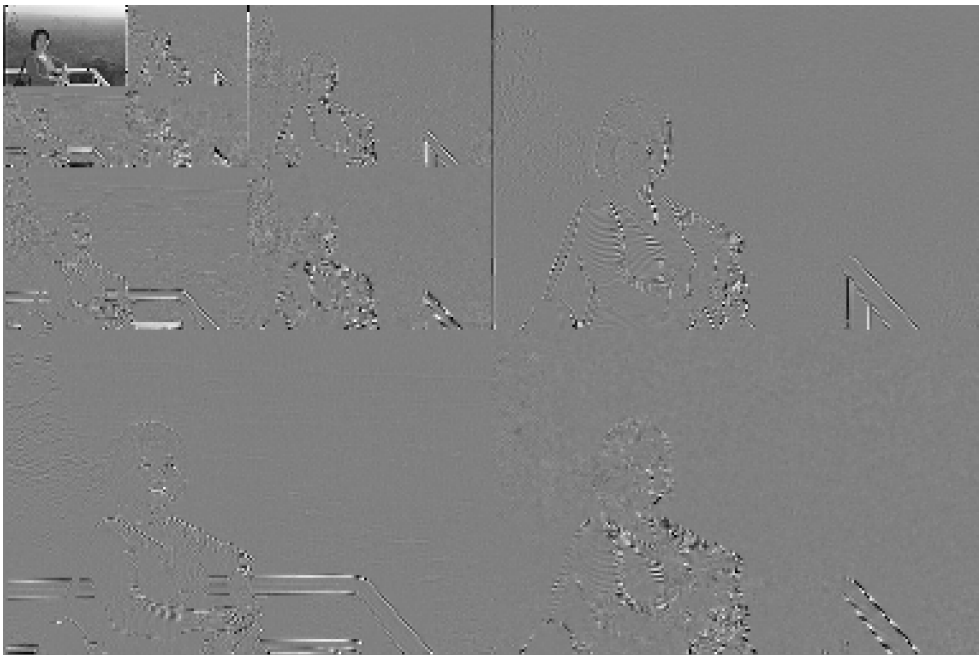


Figure 3.9: Graphical representation of the wavelet coefficients of the first 4 resolution levels of component Y. The original image is displayed in figure 4.5.

Chapter 4

Creating a JPEG2000 feature set

Having subband data available, the next task was to create a feature set using wavelet coefficients.

4.1 Current approaches

Mandal et al. [38] give a short overview of indexing techniques using wavelets. Other methods are described by [58] or [34]. In most cases, the pictures are transformed from pixel to wavelet domain right for indexing purposes. Most methods focus on retrieval performance, not storage issues. Usually general wavelet based techniques are considered, only Xiong [58] mentions JPEG2000 files explicitly.

A disadvantage of some of those techniques is that they need special wavelet transforms (e.g. [38] p.7), making them unsuitable for JPEG2000 since in the current standard two types of wavelet transforms are fixed. Others use algorithms that have to be trained (e.g. [38] p.6). This is feasible for specialized image collections, but not ideal for arbitrary image collections.

A striking fact was that almost all techniques surveyed focused exclusively on texture discrimination. Of course subbands are very well suited for that task, but general purpose image retrieval was not considered.

The approach most suitable for JPEG2000 image retrieval was given by Mandal, Aboulnasr and Panchanathan [37] and later by Do and Vetterli [34]. They propose to compare histograms of directional subbands. To reduce complexity, both suggest to model the histograms as Generalized Gaussian Densities (GGD). Mandal et al. additionally propose to create histograms of the uncompressed image by using Legendre moments.

4.2 JP2FeatureFinder

In order to develop a JPEG2000 image retrieval method, a way had to be found to quickly create, view and evaluate subband feature sets. A test framework especially tailored for that purpose was designed.

The JP2FeatureFinder was written in Java with a Swing interface to be platform independent and to have language support for graphical user interface (GUI) development. For a detailed description on how to use the JP2FeatureFinder the reader is referred to the README file provided with the software.

4.2.1 Core system

The architecture follows the example of GIFT [44], omitting certain aspects like feature access, client server communication or queries with multiple images. However, the system is modular enough that those components could be added with reasonable effort. The main reasons for creating the system was that it provides useful tools for developing JPEG2000 feature sets and that it is quickly to modify.

In the following, the main classes and their functions are described. A more detailed description can be found in the javadoc documentation provided with the software. The class diagram of the JP2FeatureFinder can be found in appendix B.

- **Feature/FeatureSet:** Each JPEG2000 image is represented by a feature set, containing several features. Those features have a unique ID, a type (histogram feature, GGD feature, etc.) and one or more values, depending on the type.

- **FeatureMaker:** Feature makers are entities that generate feature sets for JPEG2000 images using wavelet coefficient files. These files are generated by `wcextract` and then stored to save processing time. Of course in a working CBIRS they do not have to be stored, since good feature sets should be sufficient to represent an image. A feature maker factory returns a specific feature maker that can be used for the whole image collection.
- **WeightingFunction:** A weighting function is essentially the implementation of a similarity measure. Usually every feature in the search image gets associated with a weighting function depending on the feature type. The corresponding features of the images in the collection are compared using this similarity measure. The contribution of every weighter to the final ranking of each image is collected in a score board.
- **ScoreBoard:** In a so called “score board” the current score for each image is collected.
- **Normalizer:** When the scores for every document in the score board are fixed, they are normalized following a convention used in GIFT. There the query image has the score of 1 and all other images have smaller values, with the most relevant ones being close to one. If HI is used, the normalizing function looks like:

$$s_{norm} = \frac{s}{s_{search}}$$

where s_{search} is the score of the search image, i.e. the sum of all bin values, and s is the score of the image to be normalized.

For the KLD the following normalizing function is used:

$$s_{norm} = \frac{1}{1 + |s|}$$

- **QueryEngine:** The term query engine is taken from GIFT, where it denotes an entity that is capable of processing queries ([44] p33). In a basic query engine, a list of weighting functions is built according to the features of the query image. The images are compared on a feature by feature basis using the weighting functions. The results are stored in the score board and then normalized. An image collection sorted by relevance is thus returned to the caller.

A compound query engine is also available, consisting of a list of different query engines each processing a specialized query. The results are then merged following a scheme given in the normalizer.

4.2. JP2FEATUREFINDER

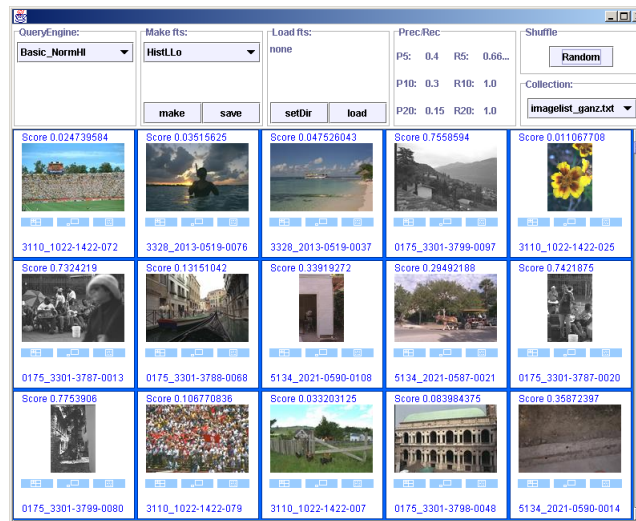


Figure 4.1: The user interface of the JP2FeatureFinder

4.2.2 Graphical user interface

The GUI consists basically of two parts: a control section in the upper part, and a query/result section at the bottom. The query section contains a scroll pane with image panels of all pictures in the current collection. For testing purposes it is useful to be able to browse the whole collection. Of course this is only feasible for relatively small image collections up to 1500 images.

The controls at the top have various purposes. For a better overview they are grouped in functional units.

- *Collection*: First, it is necessary to load an image collection by choosing from a drop down list. File names stored in a collection file are used for loading, considering a root location specified by a config file. The scroll pane is filled with image panels representing the individual images.
- *Query Engine*: Different query engines with various weighting functions and normalizers can be chosen.
- *Make features*: The feature makers available can be chosen from a drop down list. By pressing the Button “make” features are generated for all images in the current collection. Since feature generation can take quite a long time, they can be saved by pressing the button “save”. The path given




Figure 4.2: Image panel with information and controls

by the section “*Load Features*” is used for that purpose. The file format used for storing the features is the same as the GIFT feature file format (.fts).

- *Load features*: Previously generated and stored features can be loaded again. The path to the directory the feature sets are in must be given. After pressing the button “load” the system sets the features for each image in the current collection.
- *Precision/recall*: It is possible to add relevance information to an image by providing a list of images that are similar by human judgement. If such information is available (in the current test collection only few images possess such information) precision/recall values for the first 5, 10 and 20 images retrieved are displayed. This can be a measure for the quality of the query. See section 6.3 for details on precision and recall.
- *Shuffle*: By pressing the button “random” the images are shuffled randomly. In this way a start image can be chosen to begin with the query.

Image panels

The image panels display the name, a thumbnail version of the JPEG2000 image and the current score of the image. Three buttons on every image panel provide useful functions for developing a JPEG2000 feature set:

1. *Subband plotter* : By clicking this button, a new window opens and the distribution of the wavelet coefficients of each subband is shown in a

4.2. JP2FEATUREFINDER

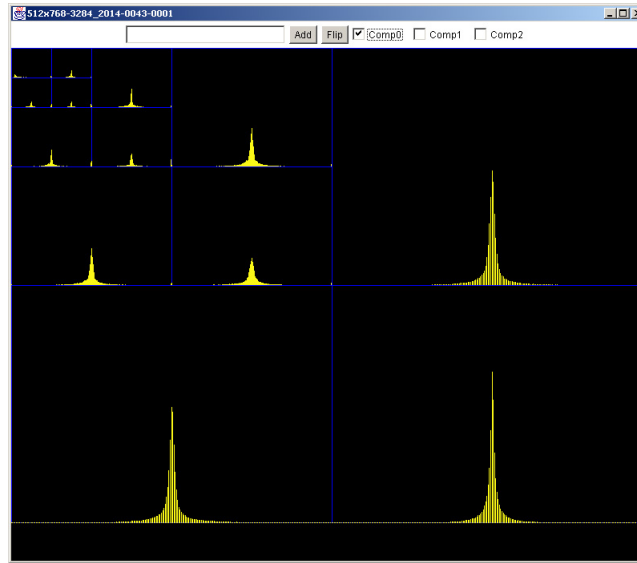




Figure 4.3: Subband histograms displayed in the JP2FeatureFinder

diagram. Opening several windows, the distribution of different images can be compared. The data of all three components can be watched at the same time or individually by checking the corresponding boxes. If the user wants to compare distributions more precisely, it is possible to enter the name of another wavelet coefficient file. The histograms are then displayed in a single window. By pressing the button “flip” the order of the histograms displayed is changed. This is important if the distribution in the front hides large parts of the histogram in the back.

2. *Enlarge* : This button opens a new window that displays the image in its original size, allowing the user to view more details.
3. *View feature set* : If features are already set or computed for this image, they can be displayed in a text window. The Feature ID is followed by the feature type and the feature value.

4.2.3 Query process and evaluation

By clicking anywhere on an image panel except for the three buttons, the search process for that image starts using the query engine currently selected. So far, only queries for a single image are possible. The image panels are reordered with

the search image at the first position (score 1.0) and the other images following in descending order according to the score.

Two ways were used to evaluate the accuracy of the query process: first, relevance information for selected images was entered, so that precision/recall information could be computed. These values were recorded to compare different feature sets and different query methods. The second way of rating retrieval quality was by the general visual impression of the images retrieved. Image similarity was simply judged by personal impression, using the symbols \oplus , \odot and \ominus . An overview about the precision/recall rates achieved for three of the test images with selected algorithms is given in table 4.1. The values stated there were used to judge the retrieval performance only roughly, a detailed test of the resulting methods is performed in chapter 6.

4.3 Feature set experiments

4.3.1 General setting

As stated above, the (in our opinion) most promising articles about wavelet based image retrieval are [37] and [34]. There the distribution of wavelet coefficients of corresponding subbands is compared. In this thesis, this approach is also taken and different modifications are tested. In the following experiments the data used was derived from JPEG2000 images encoded reversibly with 5 resolution layers and no tiling applied. The JPEG2000 images were generated from jpeg pictures of the benchathlon project [2]. A subset of 1406 images from this collection was used.

The first try was to interpret corresponding transform coefficients in the components Y , D_b and D_r of a subband as a “subband color”. These subband colors were then used to create a histogram for each subband. The color space was partitioned into 162 different colors by dividing Y into 18, D_b and D_r into 3 bins each. This is similar to the *HSV* color space partition used in GIFT and could be tested within the existing GIFT framework with only minor changes. But it showed quickly that this approach was not efficient. Neither HI nor the KLD performed well on the data. The precision/recall values and the visual appearance were quite bad.

4.3. FEATURE SET EXPERIMENTS

A better approach had to be found, treating all data together in this manner was obviously not good enough. To be able to better understand the performance of certain features, the role of the individual components and subbands had to be evaluated. GIFT uses a set of relatively simple features (color and texture) which are then combined. Picard et al. [40] previously stated that it is better to model similarity using different feature groups (which can even be highly specialized) instead of looking for the ultimate feature type and one similarity measure. So it was tried to model color and texture independently and then combine the results.

4.3.2 Texture approach

To distinguish the texture discrimination ability of different components, subbands from component Y , D_b and D_r were analyzed independently. For this purpose, histograms of individual subbands of all three components were created, using bin sizes in the range of 20-80. The subband histograms were compared using the L_1 -distance (HI) as well as the KLD.

The subbands of Y (component 0) have much more discriminating power from a structural point of view than subbands of D_b or D_r . The reason for this is that the luminance component Y contains information consisting of all RGB-color components, representing a black/white version of the original image, discarding color information and putting emphasis on the texture. The other components describe the difference of the color values, or more precisely the transform coefficients of those values. To judge structure similarity it is sufficient to use only subbands from component Y , which leads to reduced complexity and better results.

Several tests showed that considering only subbands from component Y works well if one is only interested in structurally similar images. Some of the test results are stated in table 4.1. Figure 4.4 shows an example query result. Since emphasis is put on texture retrieval, black/white images with similar structure were also found in the database.

It is possible to further reduce the number of subbands to be compared. Subband LL_0 differs from all other subbands in meaning and coefficient distribution, being only low-pass filtered. Coefficients are usually not gaussian distributed as is the case with all other subbands. To have only subbands with equal properties, for example, for further processing the data as shown in 4.4, it is possible to omit subband LL_0 without spoiling texture retrieval results.

4.3. FEATURE SET EXPERIMENTS

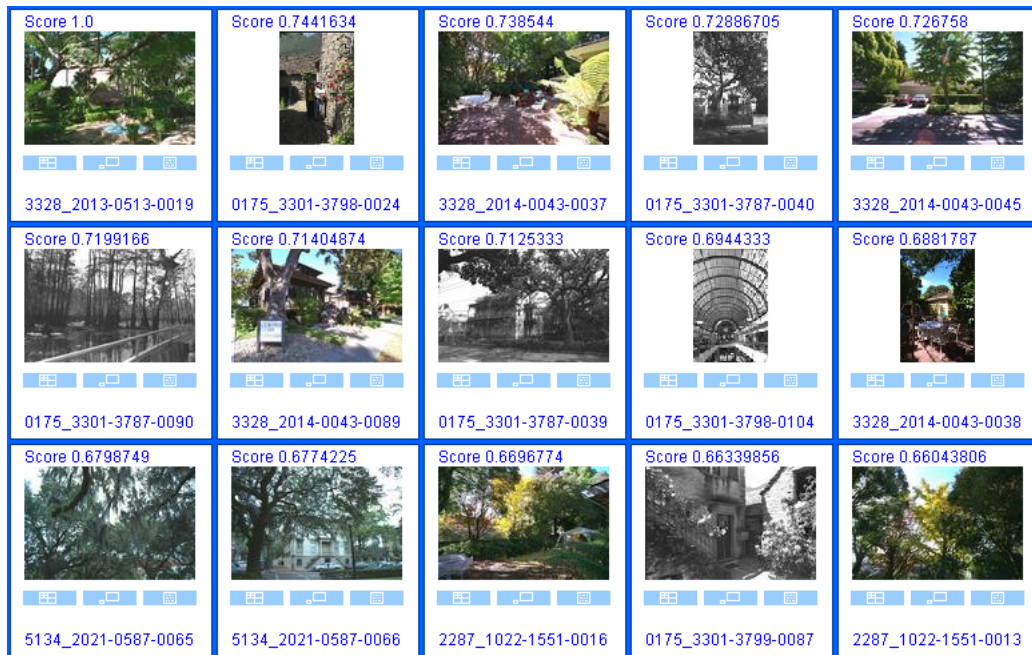


Figure 4.4: Query for texture

4.3.3 Color approach

Using only subbands from component Y discards all color information. Since for humans color is very important to judge the overall similarity of a picture, color information had to be brought back into consideration again. Color data is still contained in the subbands LL_0 of each color component. The values of those bands were only transformed low pass, so they still represent YD_bD_r color information.

To see this, the data from all three components can be put together again. An inverse color transform reveals the reduced size version of the original image. In figure 4.5, the original image is shown on the left, where as the subband LL_0 is shown on the right. The example subband image is only 38x32 pixels wide, but is enlarged to the original size to better view the reduction of pixels. Of course a smaller image usually contains a smaller number of colors. Color reduction was already nicely performed by the wavelet transform.

The application of an inverse color transform was only performed to show the property of the subband LL_0 . The colors are equally represented in the YD_bD_r color space. The YD_bD_r colors were again used to create a histogram. The color

4.3. FEATURE SET EXPERIMENTS

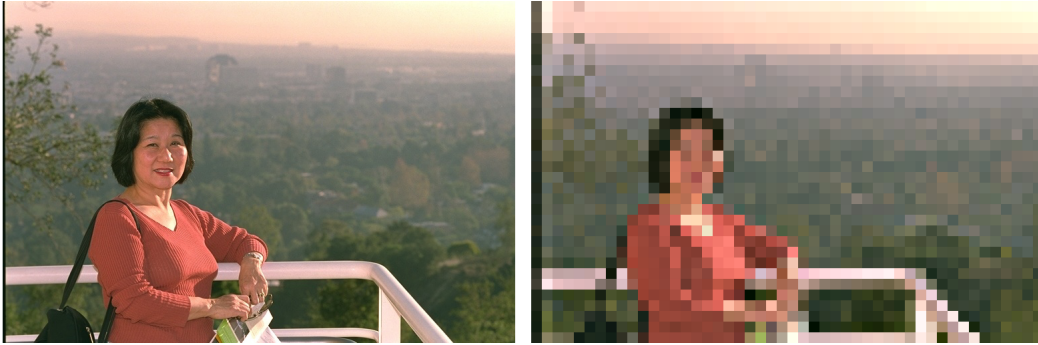


Figure 4.5: original image and enlarged subband LL_0

space was partitioned into 320 partitions, by dividing the individual components into 5:8:8 parts. Since emphasis is put on the colors, the chrominance components were divided finer. Other partitions were tested, but this one was a good tradeoff between accuracy and complexity. Selected test results can be seen in table 4.1.

Additional tests with color information transformed to the HSV space brought slightly better results. As already described in 3.2.2, the HSV color space is adapted better to the human visual system. But since compressed domain techniques are evaluated in this thesis, the transform to the HSV space had to be avoided.

Figures 4.6 and 4.7 show image queries based on color, ignoring texture information. 4.6 uses the YD_bD_r color space, where as 4.7 uses the HSV color space. The HSV based query (4.7) produces slightly better results. Of course all problems of color histograms as described in 2.3.3 like the loss of spatial context have to be faced when using this method.

4.3. FEATURE SET EXPERIMENTS

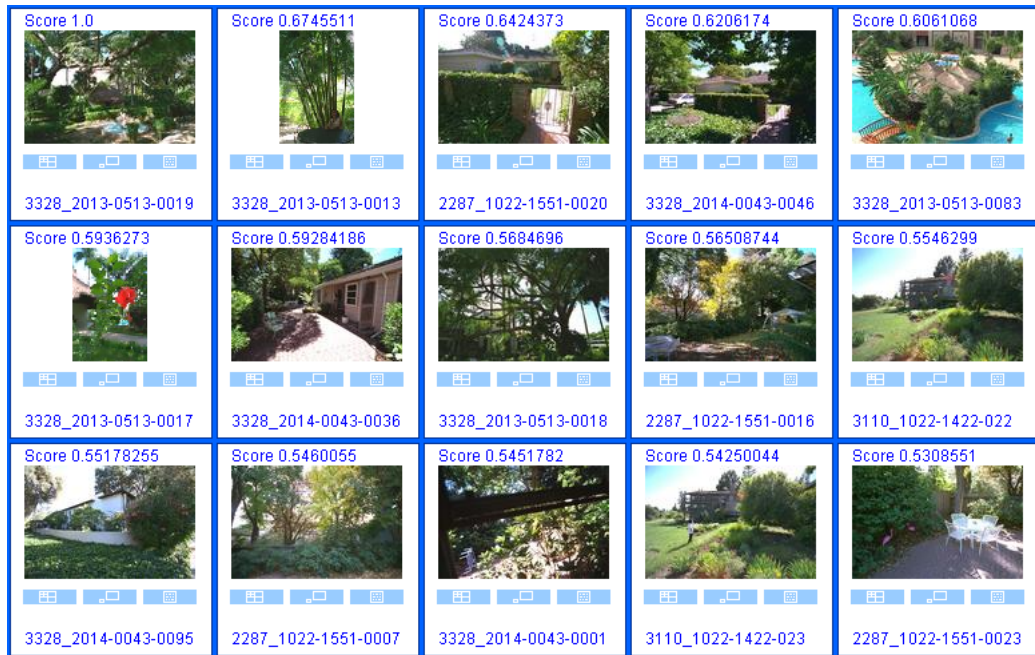


Figure 4.6: YD_bY_r color query

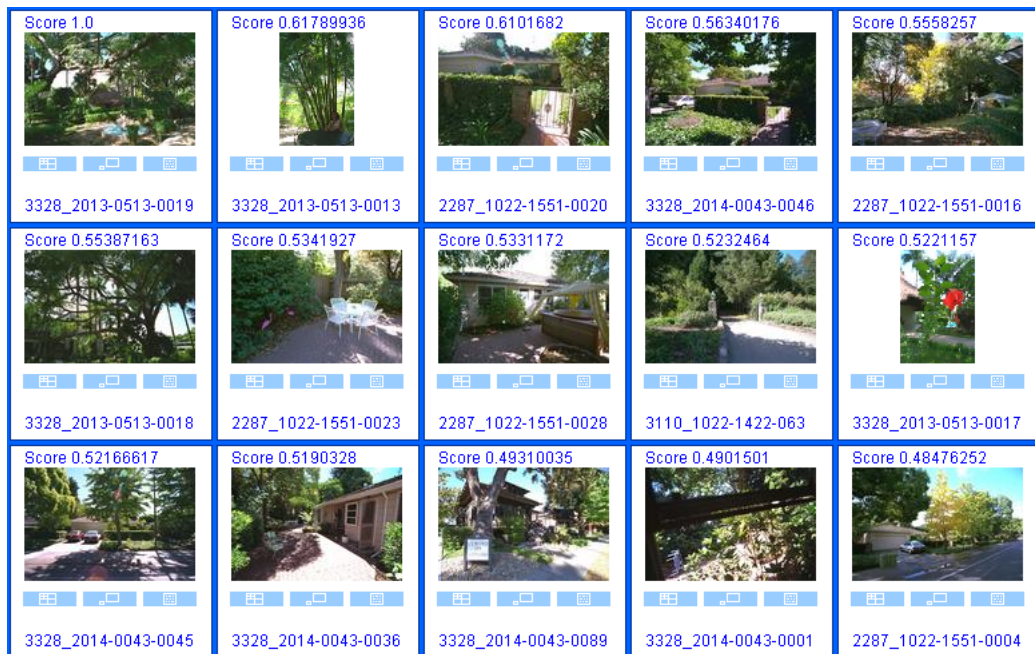


Figure 4.7: HSV color query

4.3.4 Compound approach

Having found a way to model color and texture independently, the results had to be combined to obtain a retrieval system that works well on arbitrary images.

The idea was to perform two queries and then to merge the results. Three different methods were tested:

1. **Combination depending on rank:** The first try was to simply consider the ranking of the pictures in the individual result sets. The new score of an image is the mean of the positions in the result sets of different queries. A new result list was built depending on the new scores. Theoretically, any number of result lists can be merged in this way.

A drawback of this method is that considering only the relative position of an image discards information. The distance between following images are all made equal. It is not possible any longer to determine if two subsequent images are both very close to the search image in having a very high score, or if only one has a high score and the other one is quite dissimilar.

2. **Combination depending on KLD values:** As already described in section 2.3.4, it is possible to combine different KLDs from multiple data sets using the chain rule. So when using the KLD for computing color and texture similarity, the values can be added to obtain a compound measure.
3. **Combination depending on the score:** The third possibility tries to improve the first method: compound scores of the images are computed by calculating the mean of the scores obtained. This produces better results since the relative distances between the images are considered.

In either method it is useful to be able to emphasize color or texture properties by providing a weight factor. This weight factor changes the influence of one approach. It makes sense to allow the user to specify this weight factor freely in order to express his preference. By setting the weight factor for a component to zero, it is even possible to consider only texture or color similarity.

4.3. FEATURE SET EXPERIMENTS

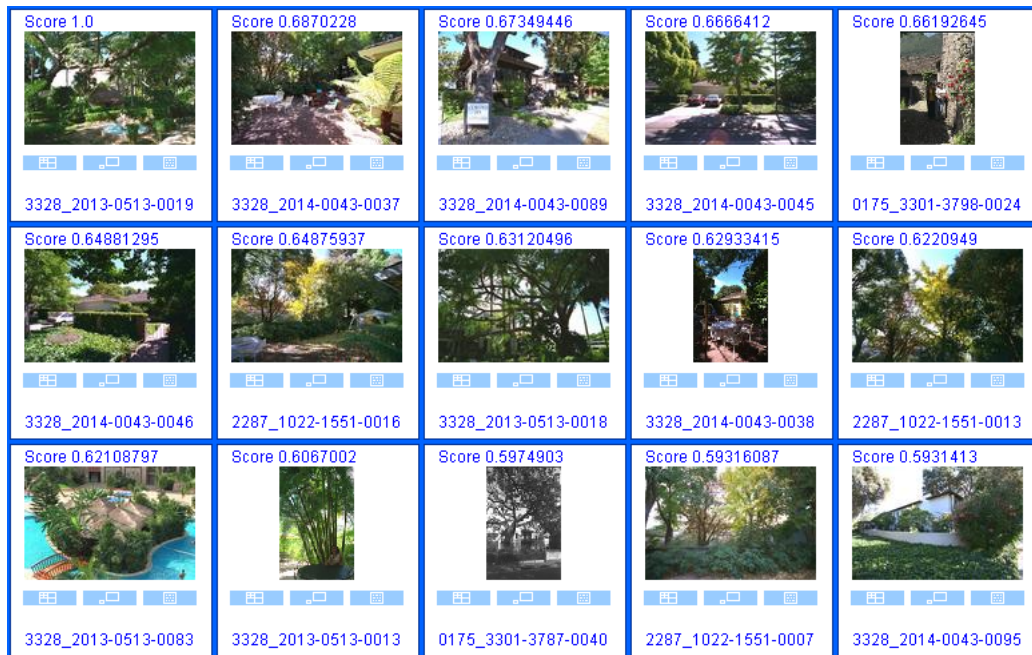


Figure 4.8: combined query on score with texture weight 4:1

Many tests showed that there is no universally valid weighting scheme for obtaining good results. For example, the best result for the previous example was obtained by weighting texture information 4 times more than color, in other examples no weighting (1:1) already gave good results. In a general setting, weighting of 1(color):2(texture) is a good value to start with, allowing the user to refine the values later.

	woman/child 3328_2014-0043-0076			musicans 0175_3301-3787-0006			woman/bath 3328_2013-0519-0013			visual
	P/R 5	P/R 10	P/R 20	P/R 5	P/R 10	P/R 20	P/R 5	P/R 10	P/R 20	
YCC13-3-3 (HI)	0.6/0.38	0.3/0.38	0.25/0.62	0.2/0.33	0.1/0.33	0.1/0.66	0.4/0.4	0.2/0.4	0.1/0.4	⊖⊖
YCC40-10-10 (HI)	1.0/0.63	0.5/0.63	0.25/0.63	0.2/0.33	0.2/0.66	0.1/0.66	0.8/0.8	0.4/0.8	0.2/0.8	⊖
bands c0 (HI) 80 bins 12 subbands	1.0/0.63	0.5/0.63	0.25/0.63	0.4/0.66	0.3/1.0	0.15/1.0	0.4/0.4	0.2/0.4	0.1/0.4	⊕
bands c0 (KLD) 80 bins 12 subbands	0.6/0.38	0.3/0.38	0.25/0.63	0.2/0.33	0.1/0.33	0.1/0.66	0.4/0.4	0.2/0.4	0.2/0.8	⊕
bands c1 (HI) 80 bins 12 subbands	0.4/0.25	0.3/0.38	0.15/0.38	0.4/0.66	0.2/0.66	0.1/0.66	0.4/0.4	0.2/0.4	0.15/0.6	⊖
bands c0 (HI) 40 bins 12 subbands	0.6/0.38	0.3/0.38	0.15/0.38	0.4/0.66	0.2/0.66	0.1/0.66	0.4/0.4	0.3/0.6	0.2/0.8	⊖
bands c0 (HI) 20 bins 12 subbands	0.6/0.38	0.3/0.38	0.15/0.38	0.2/0.33	0.2/0.66	0.15/1.0	0.6/0.6	0.3/0.6	0.2/0.8	⊖
bands c0 (HI) 80 bins 13 subbands	0.6/0.38	0.3/0.38	0.15/0.38	0.6/1.0	0.3/1.0	0.15/1.0	0.4/0.4	0.2/0.4	0.15/0.6	⊕
bands LL_0 (HI) 18x3x3	0.2/0.13	0.1/0.13	0.1/0.25	0.6/1.0	0.3/1.0	0.15/1.0	0.4/0.4	0.5/1.0	0.25/1.0	⊖
bands LL_0 (KLD) 18x3x3	0.2/0.13	0.2/0.25	0.1/0.25	0.0/0.0	0.0/0.0	0.05/0.33	0.8/0.8	0.5/1.0	0.25/1.0	⊖
bands LL_0 (KLD) 5x8x8	0.4/0.25	0.2/0.25	0.1/0.25	0.4/0.66	0.3/1.0	0.15/1.0	0.8/0.8	0.5/0.8	0.25/0.8	⊕⊕
bands LL_0 (KLD) 5x5x5	0.4/0.25	0.2/0.25	0.1/0.25	0.4/0.66	0.3/1.0	0.15/1.0	0.8/0.8	0.4/0.8	0.25/0.8	⊕
bands c0 (KLD) 80 bins 12 bands + LL_0 (KLD) 5x8x8 + score norm	0.4/0.25	0.3/0.25	0.15/0.38	0.6/1.0	0.3/1.0	0.15/1.0	0.8/0.8	0.4/0.8	0.2/0.8	⊕⊕
bands c0 (KLD) 80 bins 12 bands + LL_0 (KLD) 5x8x8 + list norm	0.4/0.25	0.2/0.25	0.1/0.25	0.6/1.0	0.3/1.0	0.15/1.0	0.8/0.8	0.4/0.8	0.4/0.8	⊕
bands c0 (KLD) 80 bins 12 bands + LL_0 (KLD) 5x8x8	0.4/0.25	0.2/0.25	0.1/0.25	0.6/1.0	0.3/1.0	0.15/1.0	0.8/0.8	0.4/0.8	0.2/0.8	⊖

Table 4.1: P/R results for selected feature sets and algorithms. P/R rates are stated after the first 5,10 and 20 images were retrieved. In the last column the quality of the retrieval performance by visual impression is given.

4.4 Complexity problem

Although the retrieval method developed above performed well from a quality point of view, the processing time was not good, even for relatively small image collections with about 1000 pictures. The time for processing a query on the test image collection (1406 images) using the JP2FeatureFinder was 25 seconds¹. Comparing subband histograms, especially with KLD, is expensive, since a lot of values have to be processed.

4.4.1 Generalized Gaussian Density

Following [34], a solution to this problem can be to model the pdf of the wavelet coefficients as a Generalized Gaussian Density (GGD). It is defined as:

$$P(x; \alpha, \beta) = \frac{\beta}{2\alpha\Gamma(\frac{1}{\beta})} e^{-(|x|/\alpha)^\beta} \quad (4.1)$$

where $\Gamma(\cdot)$ is the Gamma function: $\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt$, $z > 0$

The distribution can be modelled by using only two parameters, α and β . α is sometimes called the “scale” parameter, since it denotes the width of the pdf peak. β is referred to as the “shape” parameter, since it is inversely proportional to the decreasing rate of the peak ([34] p.149).

As shown in [34] p.150, the parameters α and β can be approximated using the maximum likelihood estimator². $\hat{\beta}$ can then be computed by solving the equation

$$1 + \frac{\Psi(1/\hat{\beta})}{\hat{\beta}} - \frac{\sum_{i=1}^L |x_i|^{\hat{\beta}} \log|x_i|}{\sum |x_i|^{\hat{\beta}}} + \frac{\log(\frac{\hat{\beta}}{L} \sum_{i=1}^L |x_i|^{\hat{\beta}})}{\hat{\beta}} = 0 \quad (4.2)$$

numerically. $\Psi(\cdot)$ is the Digamma or Psi function, i.e. $\Psi(z) = \Gamma'(z)/\Gamma(z)$. The solution can be found using the Newton-Raphson iterative procedure. The value

¹using a computer with an AMD Athlon XP 1800+ processor

²The maximum likelihood estimate for a parameter μ is denoted $\hat{\mu}$.

4.4. COMPLEXITY PROBLEM

for β is incrementally improved until a desired accuracy is obtained. Using good start values can speed up the process tremendously.

Appendix A of [34] shows this process in detail:

Take the left hand side of (4.2) as $g(\hat{\beta})$. The next better value for β , β_{k+1} can be found based on the previous guess of β_k , following:

$$\beta_{k+1} = \beta_k - \frac{g(\beta_k)}{g'(\beta_k)} \quad (4.3)$$

where

$$\begin{aligned} g'(\beta) = & -\frac{\Psi(1/\beta)}{\beta^2} - \frac{\Psi'(1/\beta)}{\beta^3} + \frac{1}{\beta^2} \\ & - \frac{\sum_{i=1}^L |x_i|^\beta (\log|x_i|)^2}{\sum_{i=1}^L |x_i|^\beta} + \frac{\left(\sum_{i=1}^L |x_i|^\beta \log|x_i|\right)^2}{\left(\sum_{i=1}^L |x_i|^\beta\right)^2} \\ & + \frac{\sum_{i=1}^L |x_i|^\beta \log|x_i|}{\beta \sum_{i=1}^L |x_i|^\beta} - \frac{\log\left(\frac{\beta}{L} \sum_{i=1}^L |x_i|^\beta\right)}{\beta^2} \end{aligned} \quad (4.4)$$

To solve the problem, additionally to the gamma function (Γ) and the digamma function (Ψ), the first polygamma (or trigamma) function is needed (Ψ'). To compute gamma, a library function of CERN colt [5] was used, for the di- and trigamma functions C gsl (GNU scientific library [10]) functions were ported to Java.

To find a good start value for b , a moment matching procedure can be used. [34] states that for a GGD the ratio of the mean absolute value to standard derivation is a steadily increasing function of β :

$$\mathcal{F}_M(\beta) = \frac{\Gamma(2/\beta)}{\sqrt{\Gamma(1/\beta)\Gamma(3/\beta)}} \quad (4.5)$$

If $m_1 = (1/L) \sum_{i=1}^L |x_i|$ and $m_2 = (1/L) \sum_{i=1}^L x_i^2$ are estimates for the mean absolute value and the variance, then the initial guess for β can be given by:

$$\bar{\beta} = \mathcal{F}_M^{-1}\left(\frac{m_1}{\sqrt{m_2}}\right) \quad (4.6)$$

4.4. COMPLEXITY PROBLEM

This equation is solved by generating a lookup table with the values of the function \mathcal{F}_M and the corresponding values of β .

Once a good value for β is obtained ($\Delta < 10^{-6}$), $\hat{\alpha}$ can be computed by using the equation:

$$\hat{\alpha} = \left(\frac{\beta}{L} \sum_{i=1}^L |x_i|^\beta \right)^{1/\beta} \quad (4.7)$$

Experiments of [34] as well as our results show that usually 3-5 iterations are necessary to obtain a good value for β .

4.4.2 KLD for GGD

Having now obtained only two parameters to describe the pdf of the wavelet coefficients, storage space and retrieval complexity is greatly reduced. In order to find out how similar two GGD distributions are, they can again be compared using the KLD. The KLD for GGDs is given by [34]:

$$\begin{aligned} KL[P(x; \alpha_1, \beta_1) || P(x; \alpha_2, \beta_2)] = & \log \left(\frac{\beta_1 \alpha_2 \Gamma(1/\beta_2)}{\beta_2 \alpha_1 \Gamma(1/\beta_1)} \right) \\ & + \left(\frac{\alpha_1}{\alpha_2} \right)^{\beta_2} \frac{\Gamma((\beta_2 + 1)/\beta_1)}{\Gamma(1/\beta_1)} \\ & - \frac{1}{\beta_1} \end{aligned} \quad (4.8)$$

As it was the case with the KLD on histograms of different subbands, here again a chain rule can be applied to compute the overall similarity of two images I_1 and I_2 . It is computed as the sum of the distances across all subbands:

$$KL[I_1 || I_2] = \sum_{j=1}^B KL[P(x; \alpha_1^{(j)}, \beta_1^{(j)}) || P(x; \alpha_2^{(j)}, \beta_2^{(j)})] \quad (4.9)$$

with B denoting the number of subbands.

4.4.3 Parameter distribution

When investigating the distribution of the α and β parameter across all subbands of all images in the database³, we came across an interesting property of the distribution of the α parameter: it follows the Zipf law. It was named after the linguist George Zipf who surveyed the occurrence of words in long English texts. Citing [30] p.4, the law is:

“If one counts the occurrences of each distinct word in a long enough text and plots the number of occurrences as a function of the rank, r , i.e. the position of the word in a table ordered with respect to the number of occurrences from the most to the least frequent word, one finds that $f(r)$, the number of occurrences of the word of rank r , and its rank are related by

$$f(r) \approx A_1 r^{-\zeta}, \quad (4.10)$$

where A_1 is constant and $\zeta \approx 1$.”

Word frequency follows this rule not only in the English, but also in other languages, even artificial ones like programming languages [31]. Population densities of cities or web server access statistics have also been observed to follow this law. Wentian Li gives an overview about documents published on that topic on a web-page [23]. It would be interesting to evaluate if this fact can be exploited for image retrieval, for example, by discarding very common parameters from the search.

A graphical representation of parameter distribution is shown in figure 4.9. When plotting the frequency of individual α values of the GGD on a double logarithmic scale, the values approximately form a line.

³12 parameters per each of the 1406 images: this yields 16872 parameters

4.4. COMPLEXITY PROBLEM

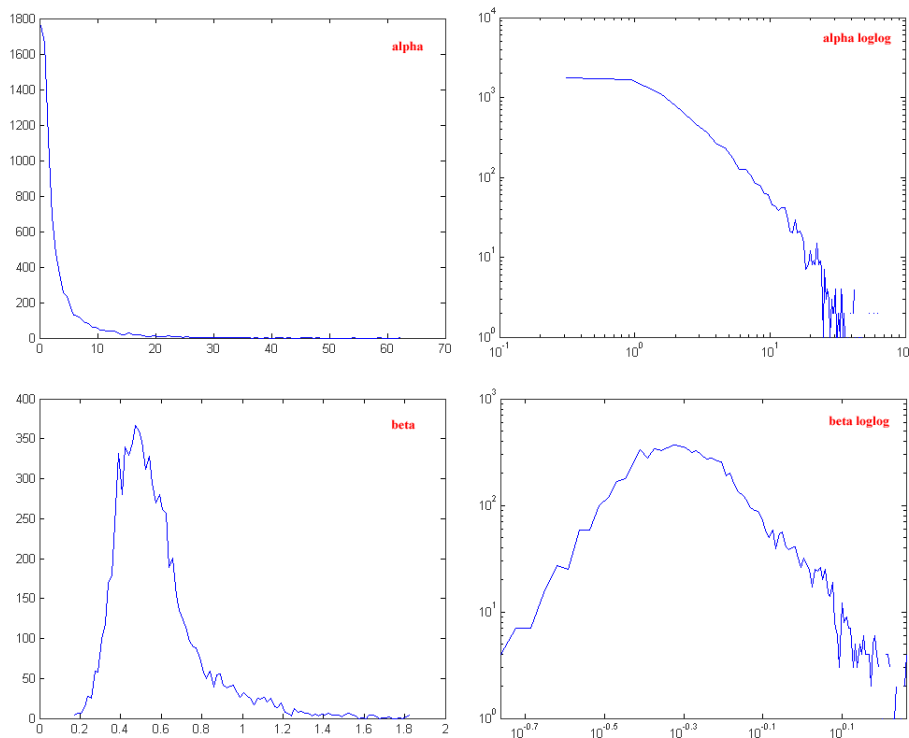


Figure 4.9: Distribution of the α and β parameter of GGDs.

4.4.4 Fitting inaccuracies

Using GGD parameters, the gain in speed was great, answering queries in a fraction of the processing time used for histograms. The processing time for a query on the test image collection was now only 0.2 seconds⁴. Although retrieval performance was good, the results differed from those obtained on subband histograms. For this reason, the quality of the GGD fits was analyzed.

In the following section, examples of images and the corresponding histograms of inner (resolution level 1) and outer (resolution level 4) subband coefficients of component 0 are displayed. Additionally fitted curves depending on the α and β parameters are shown. Please note that the scale of the y-axis is only the same in blocks that appear together. Otherwise they are adapted to the specific image group. Images containing a mixture of textured regions are usually modelled well by GGDs. In order to find out where difficulties lie, three special cases were ana-

⁴using a computer with an AMD Athlon XP 1800+ processor

4.4. COMPLEXITY PROBLEM

lyzed: heavily textured images, smooth images and images with hard partitioned regions containing smooth and textured regions.

Texture images

One extreme case to be found in an image database are images that contain a lot of structure and fine details, like the picture of a cactus garden or the memorial stone surrounded with plants shown in figure 4.10. The distribution of the wavelet parameters is widely spread and modelled sufficiently good by the GGD. The outer subbands are more smoothly distributed than the inner subbands, since more values contribute to the statistic.

Smooth images

The other extreme are images that contain a lot of smooth regions and hardly any rapid changes in color or contrast. Gradient distribution of color prevails. Here most of the wavelet coefficients have a value around 0 (or around 128 in the unsigned case). This results in a very large peak, decreasing rapidly. A distribution like that is not modelled accurately, the red curve shows that the real value of the center peak is much greater in reality than the approximation by the GGD curve.

Example fits are shown in figure 4.11. The picture of the eye is slightly blurred in the region of the lid, revealing more details (values are more widely distributed) at the lower resolution subband. The underwater image of the fish however is so cloudy that no significant structure can be found, even in the lowest resolution subband.

Arbitrary images

Common images are likely to contain either part - very vivid, textured areas and smooth regions - like a sky or another gradient area. In certain cases the two “types” of distribution can be distinguished in the histogram: Quite a number of coefficients have a big distance to the center, building a kind of base. The other values are very close to the center, building a large peak in the middle of the base. This type of distribution can not be captured accurately with a single GGD model, as can be seen in figure 4.12.

To illustrate the influence of the different parts of the picture “crowd and sky” from figure 4.12, it was cut into two halves. The wavelet coefficient histogram of each sub-image reveals the contribution of coefficients to the original histogram. It can be seen that the smooth region (sky) contributes only values in the center, where as the coefficients of the part showing the crowd are much more widespread.

4.4. COMPLEXITY PROBLEM

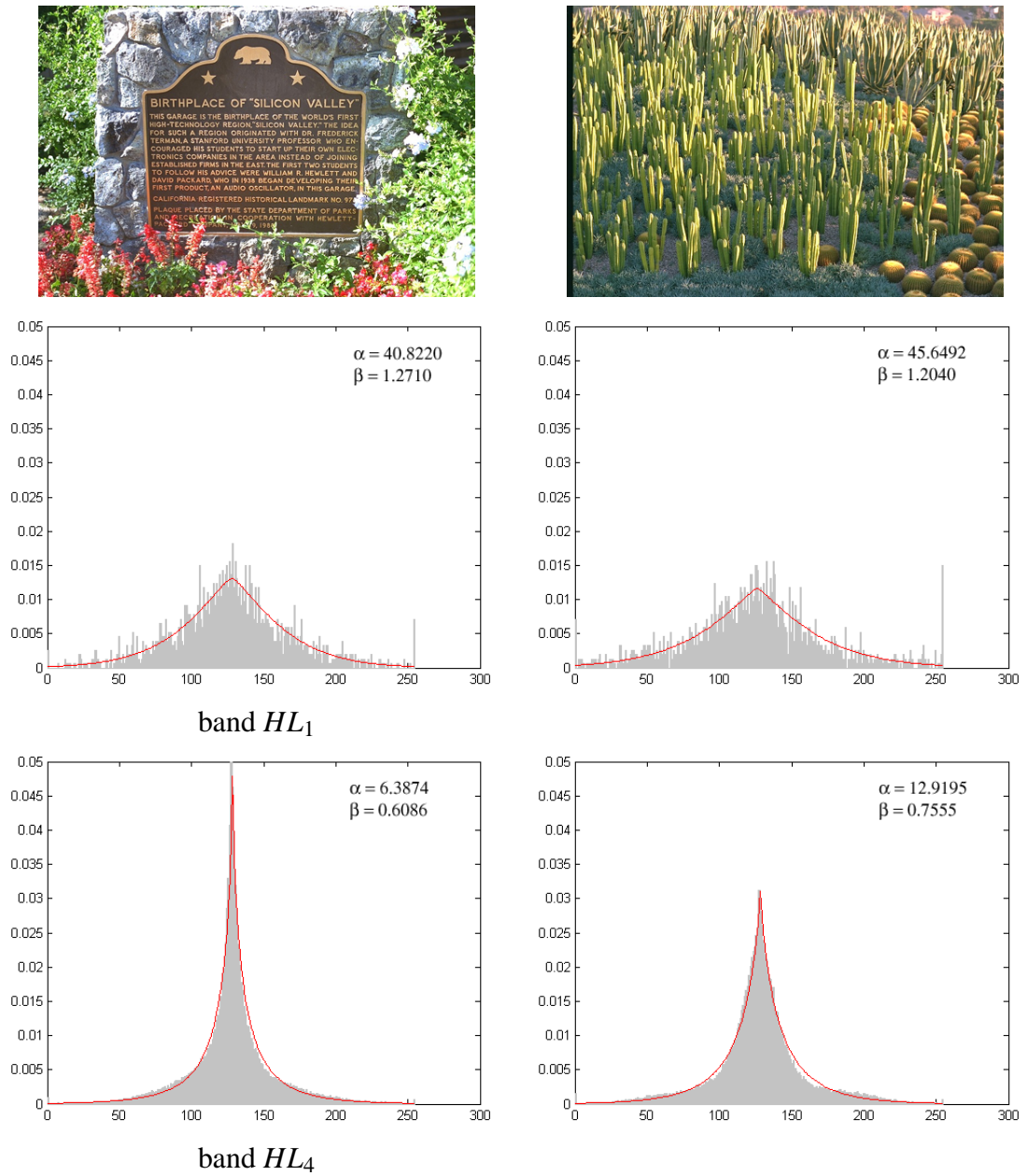


Figure 4.10: Histograms of wavelet coefficients and fitted GGD curves for heavily textured images

4.4. COMPLEXITY PROBLEM

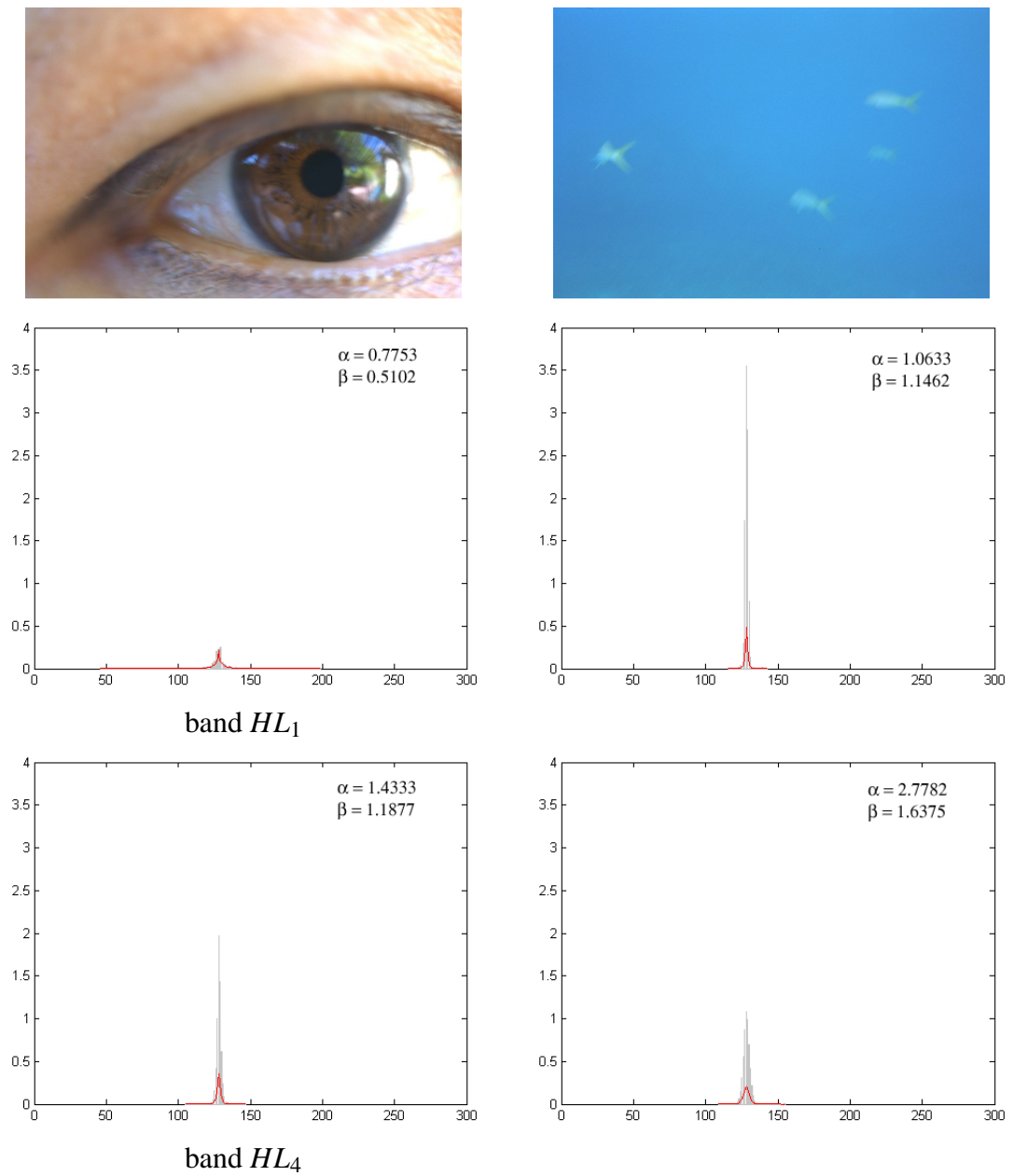


Figure 4.11: Histograms of wavelet coefficients and fitted GGD curves for very smooth images

4.4. COMPLEXITY PROBLEM

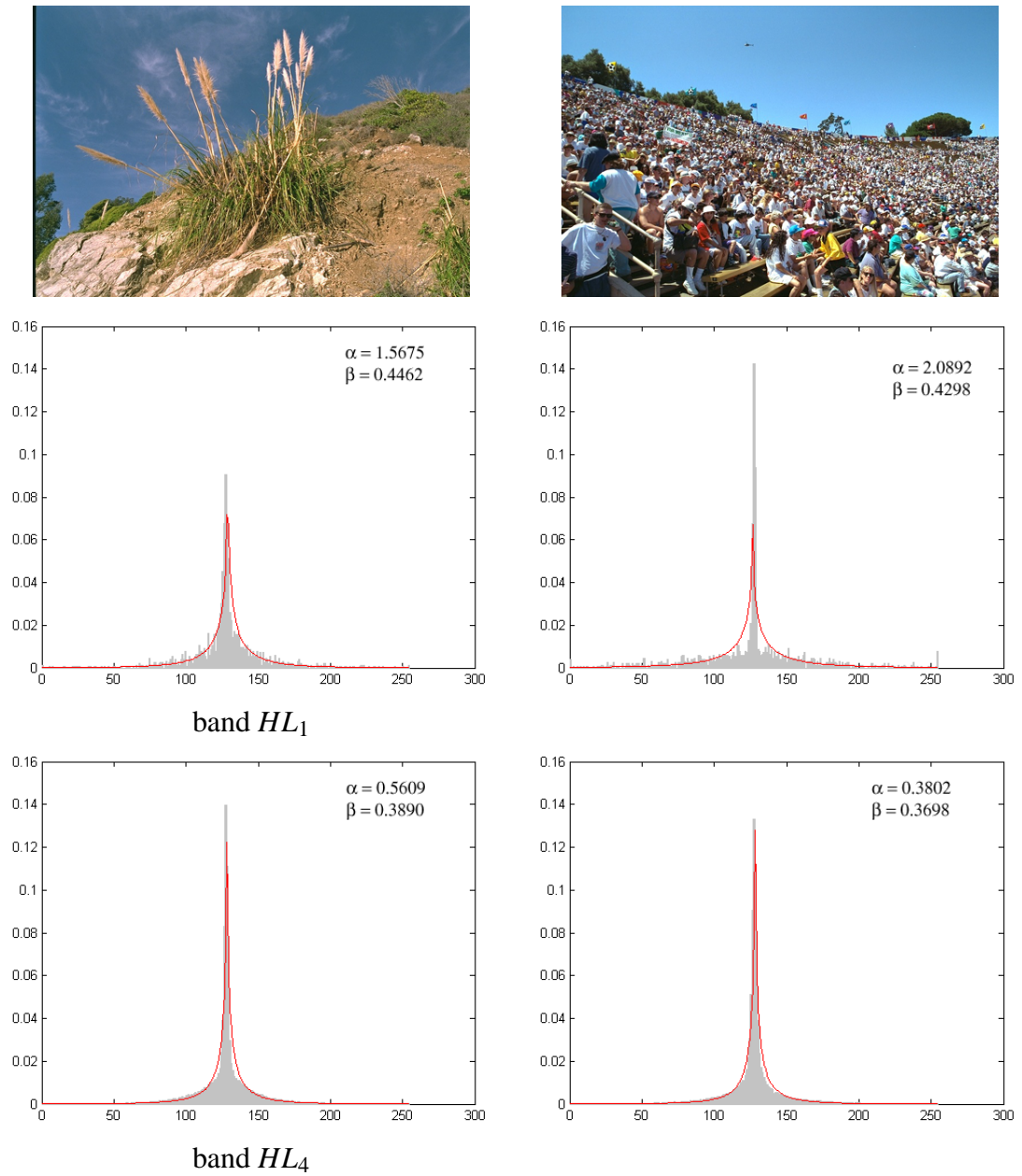


Figure 4.12: Histograms of wavelet coefficients and fitted GGD curves for arbitrary images

4.4. COMPLEXITY PROBLEM

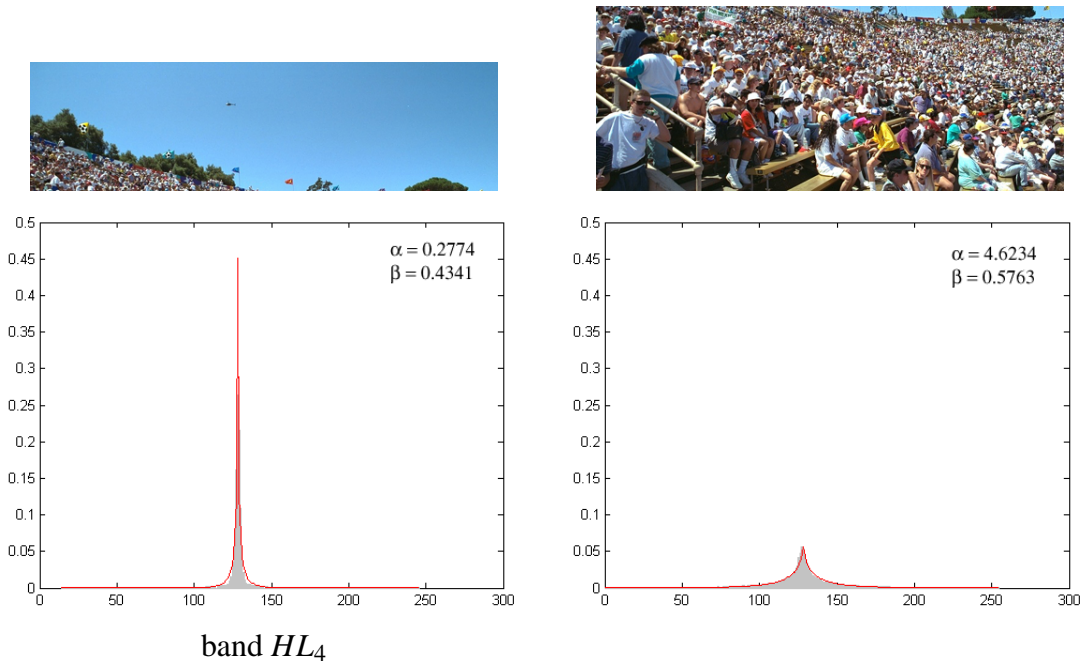


Figure 4.13: Coefficient contribution of selected parts of an image.

4.4.5 Gaussian Mixture Models

To improve fitting performance, another model to represent feature densities was investigated: Mixture Models. Following Blimes [28] p.3, a mixture density has the form:

$$P(x; \Theta) = \sum_{i=1}^N w_i P_i(x; \theta_i) \quad (4.11)$$

where the parameters are $\Theta = (w_1, \dots, w_N; \theta_1, \dots, \theta_N)$, such that $\sum_{i=1}^N w_i = 1$, and each P_i is a density function parameterized by θ_i . The function used can be any valid probability density function, i.e. any set of non-negative functions integrating to one ([54] p.70). Gaussian densities seem to be well suited for that task, so a commonly used Mixture Model is the Gaussian Mixture Model (GMM).

According to S. Bengio [27], a Gaussian Mixture is defined as follows:

- A Gaussian Mixture Model is a distribution

4.4. COMPLEXITY PROBLEM

- The likelihood given a Gaussian distribution is

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{|x|}{2}} \sqrt{|\Sigma|}} e^{(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))} \quad (4.12)$$

where μ is the mean and Σ is the covariance matrix of the Gaussian. Σ is often diagonal.

- The likelihood given a GMM is

$$P(x) = \sum_{i=1}^N w_i \mathcal{N}(x; \mu_i, \Sigma_i) \quad (4.13)$$

where N is the number of Gaussians and w_i is the weight of Gaussian i , with

$$\sum_i w_i = 1 \text{ and } \forall i : w_i \geq 0 \quad (4.14)$$

The difficulty is to determine the parameters that model a given distribution accurately. The method commonly used is the Expectation-Maximization (EM) algorithm that was developed by Dempster et al. 1977. The algorithm is explained by [48] p.229 et seq.:

Given a data set $D = \{x^1, \dots, x^M\}$ of feature points, the probability that point x^u is generated by the distribution equation 4.13 is

$$P(x^u; \Theta) = \sum_{i=1}^N w_i \mathcal{N}(x^u; \theta_i) \quad (4.15)$$

where θ_i consists of μ_i and Σ_i . The probability that the whole data set is generated by the distribution equation 4.13 is

$$\prod_{u=1}^M \sum_{i=1}^N w_i \mathcal{N}(x^u; \theta_i) \quad (4.16)$$

This also describes the likelihood $H(\Theta; x)$ that the vector Θ describes the distribution of points in D . If we could maximize the value of H , then we would

4.4. COMPLEXITY PROBLEM

obtain the optimal parameter vector Θ . Usually the logarithm of H is taken. The log-likelihood is:

$$\mathcal{L}(\Theta; x) = \sum_{u=1}^M \log P(x^u; \Theta) \quad (4.17)$$

Using the EM algorithm, Θ can be determined. The basic idea of the EM algorithm is to try to find out which of the data points x^u was generated by which of the N Gaussian distributions. Then it would be easy to estimate the parameters Gaussian by Gaussian. The ideal number of Gaussians N to capture the distribution properly can be estimated by applying the principle of *minimum description length* ([48] p.321).

To determine which point x^u was generated by which Gaussian, a set of hidden variables y_r^u is introduced, such that $y_r^u = 1$ if x^u was generated by the r^{th} Gaussian, and zero otherwise.

Now μ_r and Σ_r for the r^{th} Gaussian could be determined using the formulas for the single Gaussian, that is:

$$\mu_r = \frac{1}{\sum_u y_r^u} \sum_u y_r^u x^u \quad (4.18)$$

and

$$\Sigma_r = \frac{1}{\sum_u y_r^u - 1} \sum_u y_r^u (x^u - \mu_r)(x^u - \mu_r)^T \quad (4.19)$$

The weight w_r is then given by

$$w_r = \sum_u \frac{y_r^u}{|D|} \quad (4.20)$$

The value of the variables y_r^u are not known, so they have to be estimated, too. But if we have the values for y_r^u then the computation for Θ is easy. On the other hand,

4.4. COMPLEXITY PROBLEM

if Θ was known one could estimate y_r^u by:

$$y_r^u = \frac{1}{\sum_{i=1}^N w_i \mathcal{N}(x^u; \theta_i)} w_r \mathcal{N}(x^u; \theta_r) \quad (4.21)$$

This problem can be solved iteratively: On the basis of initial parameters for Θ better values for Θ can be obtained. Of course good start values are needed to get good end results. Several methods exist for initialization, for example, (hierarchical) K-Means or Gaussian splitting [28]. Once start values are available, the algorithm given in [48] p.230 can be used:

Assume that the number N of Gaussians in the model is known. Also, let Θ be an initial estimate for the parameter vector. Initialize $stop = false$.

1. *while* $\neg stop$
2. *for each point* x^u
3. $S_{x^u} \leftarrow \sum_{i=1}^N w_i \mathcal{N}(x^u; \theta_i)$
4. *for* $r = 1, \dots, N$
5. $y_r^u \leftarrow \frac{1}{S_{x^u}} w_r \mathcal{N}(x^u; \theta_r)$
6. *for* $r = 1, \dots, N$
7. $w_r \leftarrow \frac{1}{|D|} \sum_u y_r^u$
8. $\mu_r \leftarrow \frac{1}{\sum_u y_r^u} \sum_u y_r^u x^u$
9. $\Sigma_r \leftarrow \frac{1}{\sum_u y_r^u - 1} \sum_u y_r^u (x^u - \mu_r)(x^u - \mu_r)^T$
10. $\mathcal{L}(\Theta) = \sum_u \log \sum_{r=1}^N w_r \mathcal{N}(x^u; \theta_r)$
11. *if* $|\frac{\mathcal{L}(\Theta) - \mathcal{L}^{old}}{\mathcal{L}^{old}}| < \epsilon$
12. $stop = true$
13. *else*
14. $\mathcal{L}^{old} \leftarrow \mathcal{L}(\Theta)$

See [27] and [28] for proof of convergence and exact derivation of the formulas.

In order not to have too much values to store and compare, we did without exact estimation of the number N of Gaussian mixtures, but fixed the number to 4 instead. Experiments showed that this is sufficient to model the shape of the wavelet coefficient distribution.

4.4. COMPLEXITY PROBLEM

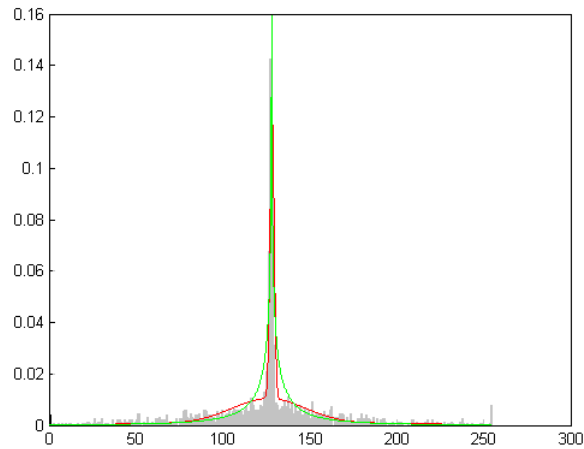


Figure 4.14: Fitting quality of GMM(red) and GGD(green) curves on subband histograms of images with different textures.

For initialization purposes, the data space (0-255) was partitioned into 4 regions, creating a histogram with 4 bins. The initial values for μ_r^{start} were fixed to the bin centers, and the start values for Σ_r^{start} were determined using the variance of the data. The Java implementation for the JP2FeatureFinder was based on Matlab code from [24].

Comparisons of the GMM data fits and the GGD estimate showed that the GMM method is superior in the cases where the GGD had troubles: images with hard partitioned smooth and textured regions as shown in figure 4.12. Figure 4.14 uses data from subband HL_1 of the image “crowd and sky”.

In the other cases where the GGD performed well, especially with very textured images, the difference is not significant. Figures 4.15 and 4.16 show that the fitting quality is comparable. For those plots the data was taken from the fish image of figure 4.11 and the memorial stone image from figure 4.10.

4.4.6 KLD for GMM

In order to use GMMs for image retrieval, the similarity between two GMMs has to be determined. This can be done using the KLD. Unfortunately determining the KLD between mixture models is no simple task. Following Do [33], the Monte

4.4. COMPLEXITY PROBLEM

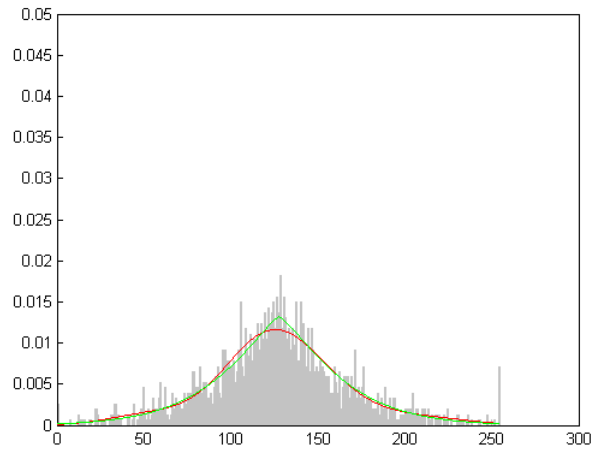


Figure 4.15: Fitting quality of GMM(red) and GGD(green) curves on subband histograms of textured images.

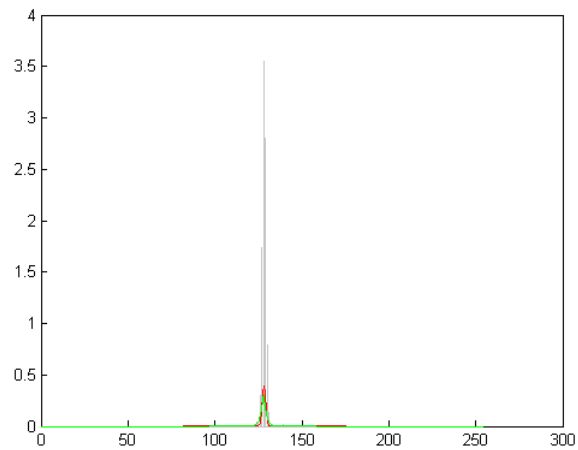


Figure 4.16: Fitting quality of GMM(red) and GGD(green) curves on subband histograms of smooth images.

4.4. COMPLEXITY PROBLEM

Carlo method can be used to approximate the integral in equation 2.10, so the equation can be rewritten as:

$$KL[P_1||P_2] = E_{P_1}[\log P_1(X)\log P_2(X)] \quad (4.22)$$

If now a set of data samples $\{x_1, x_2, \dots, x_N\}$ is generated randomly and independently based on the model density P_1 , then the KLD can be approximated by:

$$KL[P_1||P_2] \approx \frac{1}{N} \sum_{n=1}^N [\log P_1(x_n) - \log P_2(x_n)] \quad (4.23)$$

A drawback of this method is that the number of N must be large to get a good approximation. So this method is only used here to evaluate the quality of image retrieval based on GMMs and KLD. In order to be used in a working CBIRS, other, faster methods have to be found. A query on the test image collection needed an average time of 6 seconds⁵. This is still better than direct comparison of subband histograms, but worse than the GGD method. An other disadvantage of calculating the KLD with this method is that the similarity between two images can change from query to query since the random data samples and thus the estimated value of the KLD can change.

However, using this method it could be shown that retrieval performance for the “trouble” images can be improved. For the other images few difference could be observed. This can be explained by looking at the quality of the data fits: the textured and smooth images were approximated equally well, the improvement was mainly made with images containing different types of textures. Example queries were performed with three types of images: smooth images (the fish image from figure 4.11), textured images (the memorial stone of figure 4.10) and an image with clearly divided smooth and textured areas (image with crowd and sky from figure 4.12). For all those images queries were performed using GGD and GMM fits and the results were compared. The similarity was determined using the KLD, and the first 10 images retrieved were compared. See figure 4.17, 4.18 and 4.19 for the results of the queries.

⁵using a computer with an AMD Athlon XP 1800+ processor

4.4. COMPLEXITY PROBLEM

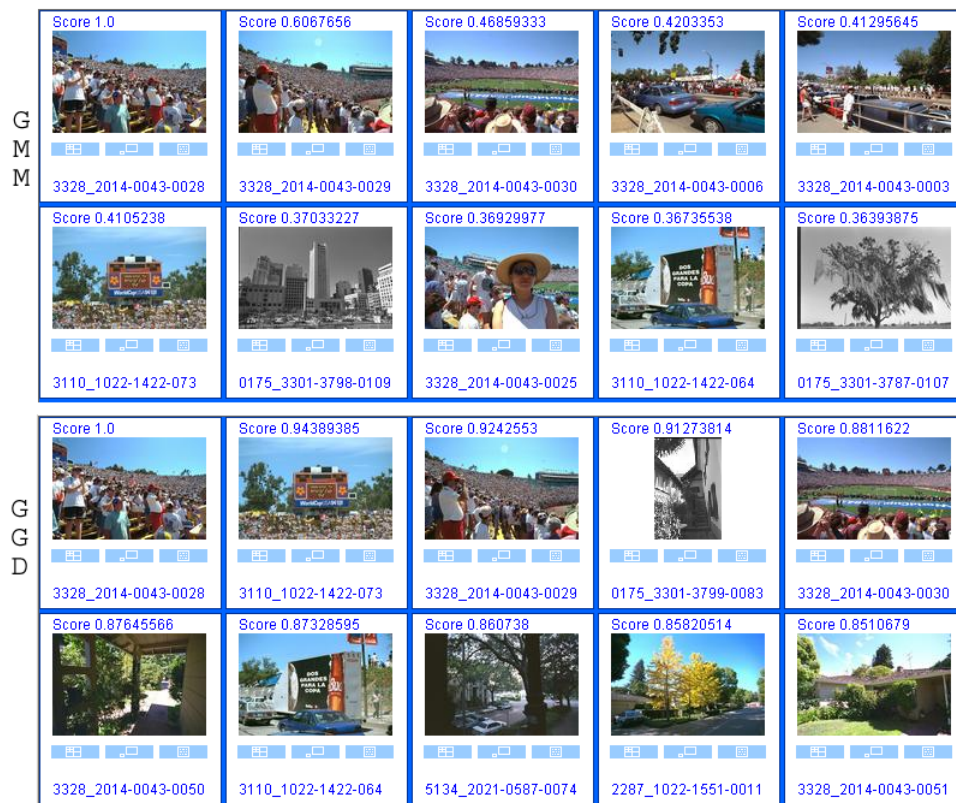


Figure 4.17: Texture query for an image with smooth and textured regions using GGD and GMM data fits.

4.4. COMPLEXITY PROBLEM

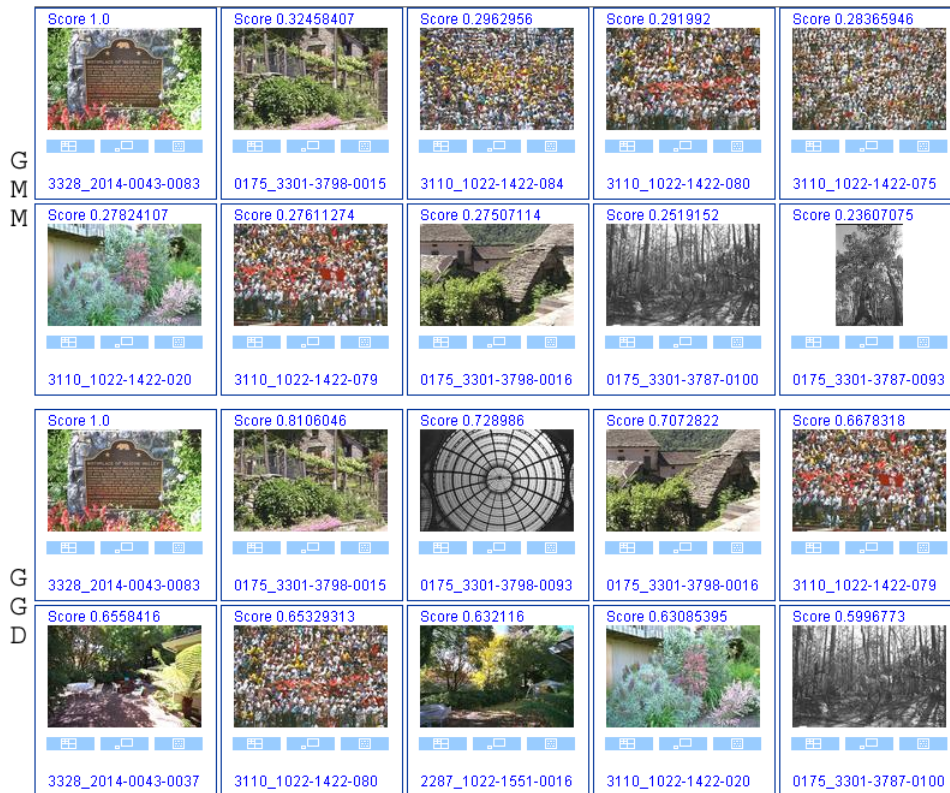


Figure 4.18: Texture query for a heavily textured image using GGD and GMM data fits.

4.4. COMPLEXITY PROBLEM

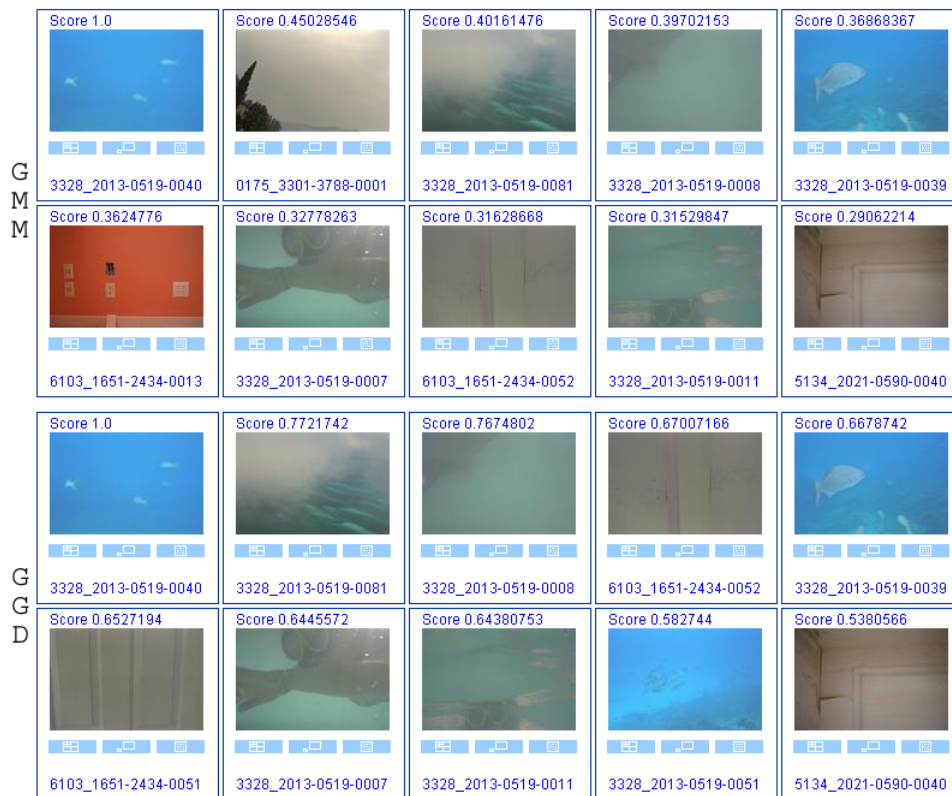


Figure 4.19: Texture query for a smoothly textured image using GGD and GMM data fits.

4.5 Image quality

A drawback of the wavelet coefficient method is that it is quite vulnerable to different image qualities. In figure 4.20, two visually similar images having different quality are compared. The bad image is very blurred and some regions are missing because of a reflection. It can easily be seen that the wavelet coefficient distributions are quite different. The blurred image produces values very near to the center forming a large peak, whereas the other image produces coefficients that have a greater distance from the center.

This is an inherent problem of the subband coefficient method. Images are only likely to be found if they are about the same quality. A solution to this problem might be to preprocess (filter) the images before indexing, but superior performance will not be obtained for different images qualities.

4.6 Summary

To summarize the knowledge gained in the development process described in this chapter, the results are concluded here:

1. Image similarity can be measured by comparing wavelet coefficient distribution of corresponding subbands.
2. The wavelet coefficient method is sensible to different image qualities. Good retrieval results can only be achieved for images with approximately the same quality.
3. For color based retrieval, the color information contained in subband LL_0 of each component is used. The original YD_bD_r color space is reduced, where a partition of $5 \times 8 \times 8$ performed best in experiments. The values are quantized to this range. Histograms using these quantized values can be created and compared using HI or KLD.
4. Structural similarity can be judged by only considering subbands from component Y , omitting subband LL_0 . 80 bins per histogram performed best in the experiments. As a distance measure, HI and KLD can be used. The processing time was insufficient though, since direct comparison of histograms has a high complexity.

4.6. SUMMARY

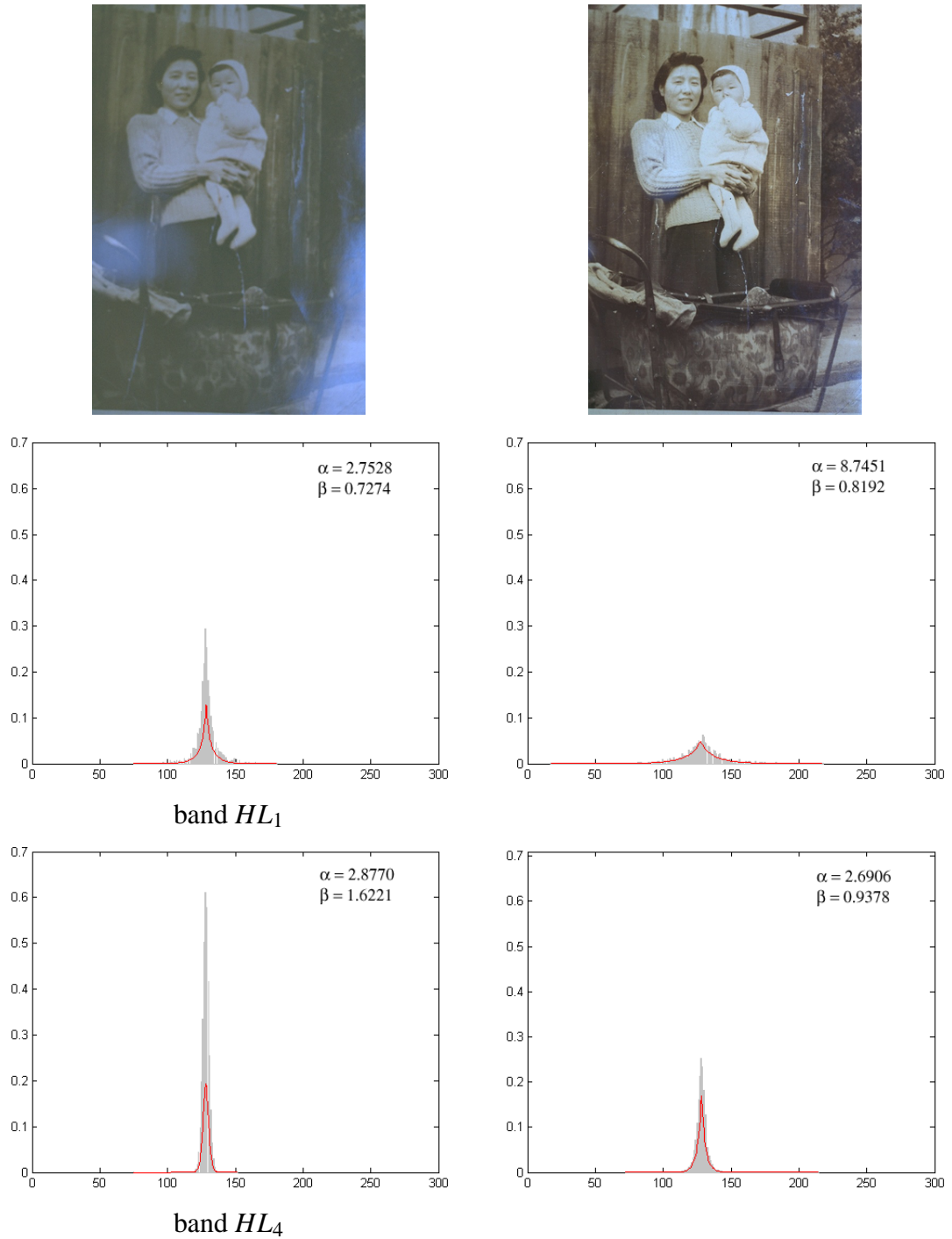


Figure 4.20: Subbands of similar images having different quality

4.6. SUMMARY

5. To reduce retrieval complexity and the amount of data stored, the subband coefficient distributions can be modelled as GGDs. A closed form KLD for GGDs is available, so the similarity between images can be measured accurately. The fitting quality of the GGD is well except for certain distributions where large smooth and very textured regions appear in the same image. Retrieval speed is greatly improved by the GGD method.
6. GMMs are better suited to model all types of wavelet histogram shapes. In this work, only a Monte Carlo approximation of the KLD for judging the similarity between images was used, so an improvement could only be observed from a quality point of view. Retrieval speed was worse than for GGDs, but at least better than with direct histogram comparison (KLD and HI).
7. Compound queries using both texture and color features perform best. For this purpose, two specialized queries are carried out and the results are merged. Merging the results using their scores and then normalizing again performed superior in these tests.

Chapter 5

Integration in GIFT

The retrieval methods developed in the previous chapter were integrated into another image retrieval system: the GNU Image Finding Tool (GIFT). The benefits of integrating an algorithm into an existing framework are obvious: all features offered by the GIFT can be used. For example, since GIFT is a client server based system, now various users can access the retrieval methods over the internet. As a communication protocol GIFT uses MRML (Multimedia Retrieval Markup Language), which allows arbitrary applications that support MRML to communicate with the system. So not only user interfaces can connect to the server, but also other applications as, for example, the benchmarking program used in the next chapter for evaluation purposes.

5.1 GIFT

GIFT is a tool for CBIR written in C++. It was developed by the *Viper* group at the University of Geneva and can be obtained from [9]. It is distributed under the GNU General Public License (GPL). A detailed description on how to install and use the software can be found in [41, 42]. [45] describes how GIFT works and how it can be extended:

GIFT is basically a system that administers a set of plug-ins. It creates them when needed, supports the infrastructure used by them and destroys them when they are superfluous. There are two major types of plug-ins: query processors

and query accessors. The task of query processors is to receive requests and to process them. They can consist of several other query processors, so queries are not restricted to one particular type. For that purpose a tree structure of query engines is built, and the nodes either process the queries or they assemble the results. Query accessors deal with everything that has to do with accessing image characteristics, for example, feature data. Several query processors can share one accessor.

The architecture of GIFT makes it possible to quickly add new components simply by providing new plug-ins. They have to be implemented as shared libraries and stored in the library directory of the server. When they meet certain requirements, for example, the library names satisfy certain rules (they have to start with “libGIFTAc” or “libGIFTQc” and end with “.so”) and they provide certain functions, then they are identified as valid plug-ins.

5.2 Configuration and startup

This is a brief overview about configuring and starting GIFT as far it is necessary to understand the development of the plug-in in the next section. A far more detailed description is given by the document “configuring-and-hacking-the-gift” provided with the GIFT software package.

When starting the server, a config-file (`gift-config.mrml`) must be provided to determine the configuration. The main purpose of the configuration file is to record the image collections and the retrieval methods (algorithms) available on the server along with the necessary parameters. Parts of the configuration file are sent to the client to provide this information. As already hinted by the ending “.mrml” the syntax used is MRML. MRML is based on XML (Extensible Markup Language) and was especially developed for multimedia retrieval. It is used as communication protocol between client and server as well as configuration language. More information on MRML can be found in [20].

<algorithm-list>

In the configuration file, the tag `<algorithm-list>` marks the beginning of a list of `<algorithm>` tags. Each `<algorithm>` tag contains information necessary for correctly executing a retrieval algorithm on the server. An example `<algorithm>` tag used for the JP2 plug-in can be seen in the following listing:

5.2. CONFIGURATION AND STARTUP

```
<algorithm cui-base-type="jp2" c-weight="1" s-weight="1"
  algorithm-id="a-jp2GGDLLoKLD"
  algorithm-type="a-jp2GGDLLoKLD"
  algorithm-name="JPEG2000"
  featureset-prefix="tsrfts_compGGDLLoHI">

  <query-paradigm-list>
    <query-paradigm type="jp2"/>
  </query-paradigm-list>

  <property-sheet property-sheet-id="jp2prop"
    property-sheet-type="subset"
    send-type="none" minsubsetsizes="2" maxsubsetsizes="2">
    <property-sheet property-sheet-id="jp2prop-1-1"
      caption="color-weight"
      property-sheet-type="numeric" send-type="attribute"
      send-name="c-weight" from="0" to="10" step="1"
      send-value="1"/>
    <property-sheet property-sheet-id="jp2prop-1-2"
      caption="structure-weight"
      property-sheet-type="numeric" send-type="attribute"
      send-name="s-weight" from="0" to="10" step="1"
      send-value="1"/>
  </property-sheet>
</algorithm>
```

The attribute `cui-base-type` is used by GIFT to determine which query processor class has to be instantiated to handle the query. In our case it is the JP2 query processor: CQuJP2. The attribute `algorithm-type` determines the specific type of the algorithm. In the development process of the JP2 indexing method different feature sets and weighting schemes were developed. In order to specify a certain method, this tag is used. A query engine matching this algorithm type gets constructed. The `algorithm-id` denotes a certain instance of an algorithm in case there are more algorithms of the same type in a query tree. Since the JP2 plug-in developed only consists of a single element, the `id` is the same as the type. Users might not cope well with algorithm IDs and types, so the attribute `algorithm-name` is used to provide a human readable name for the algorithm to display in a user client.

In order to decide which algorithm can be used for which collection, the tag `<query-paradigm>` can be used. `<collection>` elements also have those tags, and an algorithm should only be applicable to a collection when the query paradigms match. However, the implementor of a new plug-in has the responsibility to ensure this or to take other precautions that only matching pairs can be used.

5.2. CONFIGURATION AND STARTUP

As explained in the previous chapter, the user should have the freedom to specify weights for color and texture. This can be done by using a property sheet. The property sheet is used by a client to create a menu which has the selection possibilities described by the property-sheet. When a user now selects values, they get transmitted with the query. If no values are specified, the default values recorded in the `c-weight` and `s-weight` attributes of the `algorithm` tag in the `gift-config.mrml` file are used.

Since different JPEG2000 algorithms might use different feature sets (for example, GMMs or GGDs), the attribute `featureset-prefix` was added to the standard `algorithm` tag. It is used by the JP2 accessor to get a feature set of a particular type, since feature files for different types of features are stored in individual sub-directories. `featureset-prefix` specifies this subdirectory.

<collection-list>

Similarly to the `<algorithm-list>` tag, the `<collection-list>` tag in the `gift-config` file is followed by a list of elements, here of course `<collection>` elements. These elements contain all information needed by the accessor to obtain collection data. A sample tag for a collection element looks like this:

```
<collection collection-id="c-11-2-63-44-11-000-6-361-5"
            collection-name="TSR2500"
            cui-base-dir="/home/alex/gift-indexing-data/ftsTSR"
            cui-feature-description-location="FileFeatureDesc.db"
            cui-feature-file-location="url2fts.xml">

    <query-paradigm-list>
        <query-paradigm type="jp2"/>
    </query-paradigm-list>
</collection>
```

The `collection-id` attribute is used by the server to uniquely identify a collection, where the `collection-name` is used for display in the user client. The attribute `cui-base-dir` denotes the root directory for the collection. The location of the files used for getting feature data (`cui-feature-description-location` and `cui-feature-file-location`) are relative to that path. The file `FileFeatureDesc.db` is basically a table where feature IDs and corresponding types are stored. The type denotes, whether a feature is a histogram feature, a GGD feature or a GMM feature. This information is necessary to determine the right weighting function for calculating the image similarity as described in 4.2. The feature files themselves only contain the feature ID and the feature values. The file

5.3. JP2 PLUG-IN

specified by `cui-feature-file-location` is very important since it holds the location of all image, feature, and thumbnail data used for displaying the image in the client and calculating image similarity. The contents of an `url2fts`-file for an tiny image collection consisting of 2 images could look like this:

```
<image-list>
  <image url-postfix="http://viper.unige.ch/images/TSR/tsr1.jp2"
        thumbnail-url-postfix="http://viper.unige.ch/images/TSR/tsr1.jpg"
        feature-file-name="/home/alex/gift-idata/ftsTSR/*/tsr1.fts"/>
  <image url-postfix="http://viper.unige.ch/images/TSR/tsr2.jp2"
        thumbnail-url-postfix="http://viper.unige.ch/images/TSR/tsr2.jpg"
        feature-file-name="/home/alex/gift-idata/ftsTSR/*/tsr2.fts"/>
</image-list>
```

The star in the attribute `feature-file-name` is replaced by the JP2 accessor with the `featureset-prefix` value in order to find the right feature file, as explained above. To start up GIFT one has to simply specify the port on which GIFT should listen and the directory where the configuration file is:

```
gift 12790 ~/GIFT/gift-home
```

Now the server accepts requests and sends answers back to the client.

5.3 JP2 plug-in

To create a plug-in for GIFT, the base files can be generated by a perl script called `“gift-plugin-maker.pl”`. Parameters needed are the name of the plug-in and the location where the directory structure created by the script should go. The files created are the header files and files with empty method bodies required by GIFT. It also installs files to correctly configure and make the empty plug-in. When examining the code generated, the main types of plug-ins can be found again: the query processor and the accessor. Two new shared libraries, `libGIFTQuJP2` and `libGIFTAcJP2` have now to be filled with function.

5.3.1 libGIFTQuJP2

The main class in this library is CQJP2, derived from the class CQuery, which does most of the initialization needed for a query processor. CQuery is already provided by the GIFT framework. A CQJP2 instance is created by GIFT when a user wants to query using the JPEG2000 retrieval method. This is the case when he selects an algorithm where the value of the attribute `cui-base-type` is “jp2”. The constructor gets two parameters:

```
CQJP2::CQJP2(CAccessorAdminCollection& inAccessorAdminCollection
             CAlgoritihm& inAlgorithm):
             CQuery(inAccessorAdminCollection, inAlgorithm)
```

The parameter `inAccessorAdminCollection` is used by the base class to do initialization work. For example, the member variable `mAccessorAdmin` is initialized. It can be used to request an appropriate accessor. The `mAccessorAdmin` decides if a new accessor has to be constructed, or if an existing one can be reused.

The `inAlgorithm` element contains the information for the selected algorithm. The default values from the config-file were eventually overridden by the property-sheet values sent by the client. This information can be used to initialize the query object.

Once the query processor and an appropriate accessor are instantiated, the main function called when an actual query arrives is the `query`-method:

```
CXMLElement* CQJP2:query(const CXMLElement& inQuery)
```

5.3. JP2 PLUG-IN

The input parameter `inQuery` is an XML element that contains a `query-step` element:

```
<query-step algorithm-id="a-jp2GGDLLoHI"
            collection="c-55-2-6-361-11" result-cutoff="0.0" result-size="10"
            session-id="d7a8f840-6a75-489b-a0f3-9999bf624469">
  <user-relevance-element-list >
    <user-relevance-element
      image-location="http://viper.unige.ch/images/TSR/tsr0128.jpeg"
      user-relevance="1" />
    <user-relevance-element
      image-location="http://viper.unige.ch/images/TSR/tsr2177.jpeg"
      user-relevance="0" />
    <user-relevance-element
      image-location="http://viper.unige.ch/images/TSR/tsr0273.jpeg"
      thumbnail-location="" user-relevance="-1" />
  </user-relevance-element-list>
</query-step>
```

The `algorithm-id` and `collection` attributes determine which collection should be used and which type of algorithm should be executed. The `result-size` tells how many images the client awaits as a response.

Within the `user-relevance-element-list` tag, a list of images with relevance information provided by the user is contained. The parameter `image-location` is important: this information is used to identify an individual image in the collection by the accessor. The user can specify the relevance of an image via the parameter `user-relevance`. A relevance value of 1 indicates that the user is looking for an image that looks like this, where a value of -1 means that the image is not similar to the image the user is looking for. 0 states that the image is neutral to the query. The original query engines of GIFT can handle queries with multiple images with different user relevance values. The query is performed after an “artificial” input image is generated consisting of information of all relevant and irrelevant input images. The JP2 query processor however is not capable of such queries, since it is not obvious how to calculate a compound query image for GGD or GMM parameters. Future work could deal with this topic. If a query with multiple images containing relevance information arrives, all irrelevant images are ignored by the JP2 query processor and only the first relevant image is considered.

Once the query image is determined, the feature set of the image is fetched by the JP2 accessor, which is then used to query for images. The retrieval process itself is

5.3. JP2 PLUG-IN

very similar to the one used by the JP2FeatureFinder described in section 4.2. All necessary classes were ported from Java to C++. Please see the javadoc documentation provided with the JP2FeatureFinder for details. The system is as flexible as the JP2FeatureFinder concerning the possibility of combining different query engines that deal with specific types of features. Weighting functions are used to determine the similarity of images and the results are stored in a score board. The scores are normalized and a list of relevant images is obtained. Only the feature sets of the two currently compared images are held in memory, afterwards they get destroyed.

In order to send the result back to the server, a `<query-result>` element must be constructed. A result set for 3 images could look like this:

```
<query-result >
  <query-result-element-list >
    <query-result-element
      calculated-similarity="1.000000"
      image-location="http://viper.unige.ch/images/TSR/tsr2099.jp2"
      thumbnail-location="http://viper.unige.ch/images/TSR/tsr2099.jpeg" />
    <query-result-element
      calculated-similarity="0.681717"
      image-location="http://viper.unige.ch/images/TSR/tsr1540.jp2"
      thumbnail-location="http://viper.unige.ch/images/TSR/tsr1540.jpeg" />
    <query-result-element
      calculated-similarity="0.640418"
      image-location="http://viper.unige.ch/images/TSR/tsr0340.jp2"
      thumbnail-location="http://viper.unige.ch/images/TSR/tsr0340.jpeg" />
  </query-result-element-list>
</query-result>
```

The purpose of the attributes can be easily guessed from their names. The user client can use the thumbnail file to display the image in a browser, where as the `image-location` tells where to get the image itself. The information needed to construct the tag is fetched from the JP2 accessor, which has access to the file that contains this information.

5.3.2 libGIFTAcJP2

The accessor CAcJP2 has the purpose to fetch image data and make it available to JP2 query engines. It is derived from the class CAcURL2FTS already existent in the GIFT software package. This base class provides functions to access elements of an `url2fts.xml` file. The elements contained in this file are explained above. The only parameter passed to the CAcJP2 constructor is the `inCollectionElement`. It contains all information of a `<collection>` tag.

```
CAcJP2::CAcJP2(const CXMLElement& inCollectionElement);
```

The core function provided by the AcJP2 accessor is:

```
JP2FeatureSet* getFeatureSet(TID tid, string fs_prefix);
```

The `getFeatureSet` function constructs a feature set for images identified by an ID. The ID of an image gets determined on construction time of the accessor, when the `url2fts.xml` file is parsed. The ID of an image can be obtained by calling the function `URLToID(const string& inURL)` provided by the base class.

To create the feature set, the information contained in the `FileFeatureDescription.db` and the feature files specified by the `url2fts.xml` file are used. In order to get the right feature set, the feature file names of the `url2fts.xml` are completed by the string `fs_prefix`. As already mentioned, the `featureset-prefix` denotes the subdirectory where specific feature files can be found. A reference to the newly constructed feature set is sent back to the calling query processor.

Besides the creation of the feature sets, information about the image and thumbnail-image location used to construct the result set are provided by the accessor. This information is again taken from the `url2fts.xml` file. So, the JP2 query accessor provides all data necessary for the query processor to answer a request.

Chapter 6

Performance evaluation

To judge the quality of the indexing methods developed, the performance had to be evaluated. In order to obtain unbiased results, it is not sufficient to only view the first few images returned by the query. Even if the images retrieved are very similar and the tester of the system seems pleased therefore, the performance might still not be well. There could be more relevant images having a low score and thus not appearing in the result set.

It is necessary to find a way to judge retrieval performance more objectively. Several aspects have to be considered, for example, the test image database, the way relevance information is obtained or how results are evaluated. In the following section, these topics are briefly discussed, using material from [43, 50].

6.1 Image databases

It is necessary to have an image collection to work with. Usually existing compositions like the Corel image collections for arbitrary images or the VisTex database for texture images are used. There is a huge variety of different image collections and subsets of them in use. So it is difficult to compare the results obtained. Another problem is that many of those databases are copyrighted and thus the experiments can not be reproduced easily by other research groups [43]. It would be desirable to have an universal, royalty free database for benchmarking CBIRS in order to obtain comparable results. Some institutions already offer such databases,

for example, the University of Washington [13]. The benchathlon network [2] has more ambitious goal: to set up a joint CBIR evaluation framework. One task among others is to create an image database with the properties mentioned above. The images used so far in this thesis are from the benchathlon collection.

For the tests in section 6.4 it was possible to use the TSR2500 image database of the *Viper* group of the university of Geneva [11]. This database is a subset of the TSR10000 collection that was created from analog tapes provided by the Télévision Suisse Romande (TSR), a Swiss TV channel, for research issues. Individual images were generated from these tapes by using a frame grabber [4] and scaled to the size 128x128. All images remain property of the TSR.

6.2 Relevance information

An image collection alone is not sufficient. It must be determined which images are similar to which other images. For this purpose, “ground truth” data has to be acquired. Various definitions of this term can be found, depending on the area in which the term is used. For example, geologists use this term to describe information collected in the field as opposed to data remotely obtained by satellites [3]. In the image processing area ground truth denotes reliable information which images are similar.

Of course it is difficult if not impossible to obtain universally valid ground truth data. [43] p.5. describes several different methods to obtain this data. The best way seems to be the most time consuming: Obtaining relevance information by user judgements. A user is given a query image and then has to browse the entire image collection to determine which images are relevant. The results can of course differ from user to user.

Ground truth for the benchathlon collection is not available yet, but already on the project schedule. Relevance information for some of the test images was created for this thesis, in order to judge the first results in chapter 4. Since it is always problematic if the designer of a system and the creator of the relevance information is the same person, this data was used only in the beginning. For extensive tests however, it is better to have unbiased data.

For the TSR2500 collection relevance information is available. It was created by 3 different test users who each had to search for images similar to 14 test images. The test users were members of the *Viper* group. The selection of those 14 images was random. For each query image, the whole collection was browsed and similar images were registered.

6.3 Performance measures

Once an image collection is chosen and relevance information is available, performance can be measured. There exist a variety of different performance evaluation methods. [43] p.6 et seqq. gives an overview. Most of the techniques used are derived from text retrieval evaluation. The most commonly used measure is:

Precision and recall (P/R): Precision is the fraction of the images returned that are relevant to the query, where as recall is the fraction of the total number of relevant images that are returned ([54] p.185):

$$precision = \frac{|\mathbf{R} \cap \mathbf{T}|}{|\mathbf{T}|} \quad (6.1)$$

$$recall = \frac{|\mathbf{R} \cap \mathbf{T}|}{|\mathbf{R}|} \quad (6.2)$$

where \mathbf{R} is the set of images that are relevant to the query, \mathbf{T} is the set of returned images and $|\mathbf{A}|$ is the cardinality of set \mathbf{A} .

Precision and recall have to be evaluated together, since the recall of a query depends heavily on the number of images retrieved [43]. For this purpose *P/R graphs* can be used as shown in the following tests. In order to create a P/R graph, several levels of recall l_1, \dots, l_m are established. \mathbf{T}_i is the smallest set of returned images that satisfy recall level l_i . Precision is measured for each \mathbf{T}_i and the results are used to create a P/R curve ([54] p.185). Usually this process is repeated for different queries and then averaged. Evaluating a P/R graph, it is possible to judge the performance of a query precisely. In general, high precision rates indicate good performance, this means the higher a curve is, the better the retrieval result. But more information can be obtained. For example, if the curve

drops early, it means that few relevant images appear in the first result positions. When the precision at high recall rates is very low, it means that a lot of images have to be retrieved until the last relevant image is found.

Another way of relating precision and recall is to use $P(10)$, $P(20)$, $P(N_R)$. This is the precision achieved after the first 10, 20 or N_R images. N_R denotes the number of relevant images in the database according to this query. The testFramework used $P(5)$, $P(10)$ and $P(20)$ to evaluate the quality of the different methods. It is also possible to consider recall at a given precision rate.

6.4 Evaluation

The following tests were made using the TSR2500 image collection with manually generated ground truth as described above. The benchmark itself was performed by a benchmarking harness (snakemeter, [44] p.115 et seqq.) which communicates with the GIFT server via MRML. Query requests are sent to the server and the results are stored in a database. Using relevance information, P/R graphs can be generated.

The various test results for the different JPEG2000 feature sets and similarity measures were compared to the default query algorithm used in GIFT. In GIFT, the features are created in the following way: the images are first scaled to 256x256 pixel, then 4 types of features are extracted [41]:

1. **Color histogram:** As a base for the global color histogram a *HSV* color space using 18 hue, 3 saturation and 3 value partitions plus 3 grey values is used. This yields 166 different colors a pixel might have.
2. **Color block:** To be able to represent local color properties, the images are recursively divided into regions until the smallest blocks are 16x16 pixels wide. In each region, the most frequent color is recorded.
3. **Gabor histogram:** In order to capture texture properties, GIFT uses histograms of Gabor filtered data. 12 different Gabor filters are used: filters in 4 different directions and 3 different resolutions.
4. **Gabor block:** Gabor blocks follow the same objective as color blocks: they restore the loss of spatial information when using histograms. Unlike the color blocks, only the smallest blocks (16x16) are considered.

6.4. EVALUATION

Each feature group is evaluated separately by a specialized query engine, and the results are then combined. Usually GIFT uses relevance feedback to improve query results. Since the JPEG2000 retrieval method is not capable of doing so, the default GIFT algorithm queries were also performed without feedback.

Concluding the results of chapter 4, features used for JPEG2000 image retrieval were:

1. **Color histogram** The lowest resolution subband (LL_0) of all three components was used to create a color histogram. The YD_bD_r color space was partitioned into $5 \times 8 \times 8$ parts and the original colors were quantized to this reduced color space. The color histogram was both compared using HI and KLD.
2. **Texture histogram** The wavelet coefficients of each subband at all resolution levels of component Y except the lowest resolution subband (LL_0) were used to create histograms. 80 bins were used to model the distribution. Similarly to the color histogram, the texture histograms were compared using HI and KLD.
3. **Texture GGD** Generalized Gaussian Densities were used to represent the subband coefficient distributions. Here the KLD served as a distance measure.
4. **Texture GMM** Gaussian Mixture Models with 4 Gaussians were also used to capture the wavelet coefficient distribution of each subband. A Monte Carlo approximation of the KLD was used to estimate the similarity.

Compound feature sets consisting of features belonging to one color and one texture feature type are used to retrieve images. The different types of features are evaluated separately and the results are merged using their score.

In the tests of chapter 4, JPEG2000 images with 5 resolution layers were used. Since the TSR2500 images are rather small (only 128×128 pixels), the application of 5 decomposition stages in the wavelet transform is too much to get good statistics in the lower resolution subbands. The smallest subbands then consist of only $8 \times 8 = 64$ widely distributed pixels making it difficult for the GGD to model the distribution accurately. For this reason, additional tests with 4 and 3 resolution levels were performed. In the following tests, the number of resolution layers used is marked; when nothing is stated, the number of resolution levels is 4.

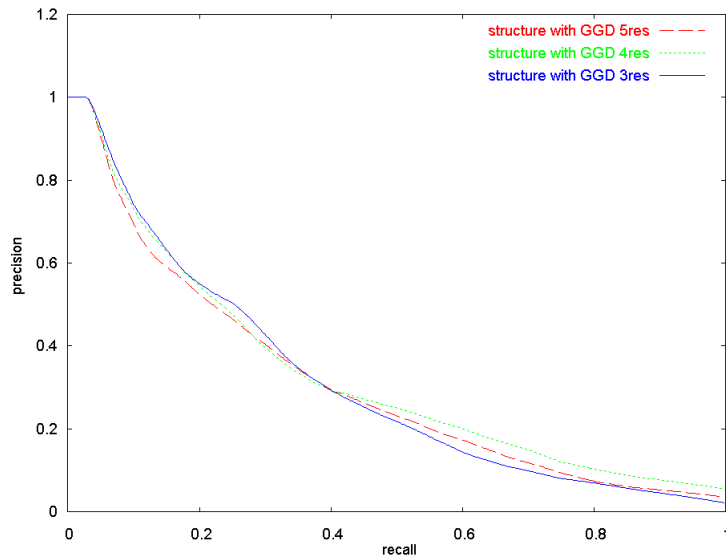


Figure 6.1: Structure query with different types of resolution layers

6.4.1 Evaluation of texture retrieval

In this section, the performance of the different texture query methods is examined. As described above, ground truth for the TSR2500 was created for color and texture features together. Although in our opinion it is difficult to use this ground truth for color and texture features separately, the following tests were made to at least estimate the influence of different methods on retrieval performance.

Figure 6.1 shows the effect of using different JPEG2000 resolution levels. GGDs were chosen as an example, being most sensitive to the distribution of wavelet coefficients. It can be seen that retrieval performance differs, with 4 resolution levels being the best on average. 3 resolution levels only perform better up to a recall rate of 0.4, then performance drops below the curve for 4 and 5 resolution levels. So 4 resolution levels were chosen as default value for the tests.

Looking at figure 6.2, it can be seen that the comparison of wavelet coefficient histograms using HI performs much better at all recall rates than the comparison using KLD. This is very surprising, since the KLD is more accurate from a probabilistic point of view. However, this observation might be explained by a property of the feature data: many histogram values are zero. Subband histograms contain around 15% - 40% zero values depending on the amount of texture in the

6.4. EVALUATION

image. Very smooth images might have zero values around 90%. When the KLD is calculated, the logarithm of the fraction of the feature probability values must be calculated (see equation 2.10). These zero values are a problem, both for the division process and the logarithm. In the current implementation all zero values are replaced by very small numbers (0.0000001) to be able to compute the KLD. Of course this introduces small errors, which may accumulate. Histogram intersection on the other hand is not affected by zero values, because only bins being not zero are considered.

The retrieval performance of the GGD features is good. It is similarly precise as direct comparison of the subband histograms using HI. The latter one performs a bit better at lower recall rates, the former at higher ones. At recall rates of 50% and higher, GGD even performs best of all texture retrieval methods tested.

The feature set using GMMs performs slightly worse than the GGD equivalent, remaining below the subband histogram (HI) and GGD curve at all recall rates. This does not mean the method is worse in general, since the images from the TSR2500 collection are all either very textured or - when showing logos or color bars - very smooth. The type of images where the GMM approximates the coefficient distribution better (hard partitioned smooth and textured images) are seldom found in the TSR2500 collection. So the GMM could not benefit from its advantage. Since no exact computation of the KLD was available for GMMs, the Monte Carlo approximation described in 4.4.6 was used.

The results obtained were also compared with the structure retrieval properties of the default GIFT query engine. Both Gabor block and Gabor histogram features were used. The results can be seen in figure 6.3. In order not to put too many graphs in the plot, the *Viper* P/R graph was only compared with the two JPEG2000 methods that performed best: GGDs and subband histograms with HI. Retrieval performance is very similar, with *Viper* performing best up to a recall value of 0.15, subband histograms between 0.15 - 0.5 and GGDs at recall rates of 0.5 and higher.

6.4. EVALUATION

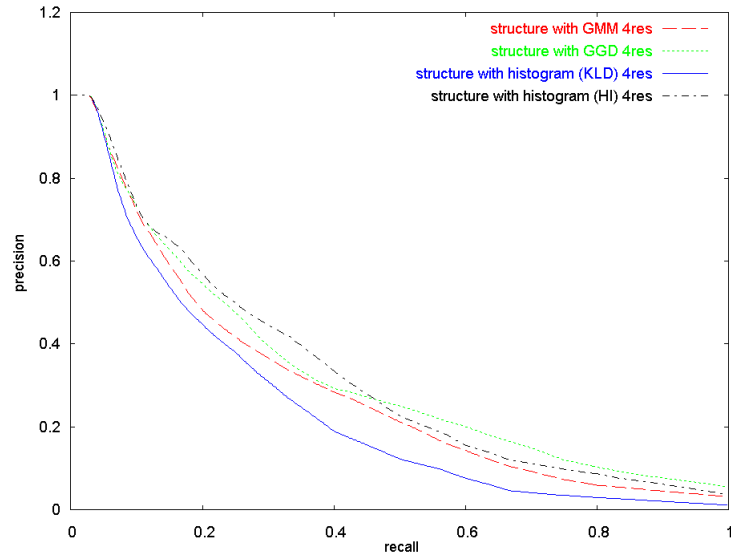


Figure 6.2: Structure query with different types of features

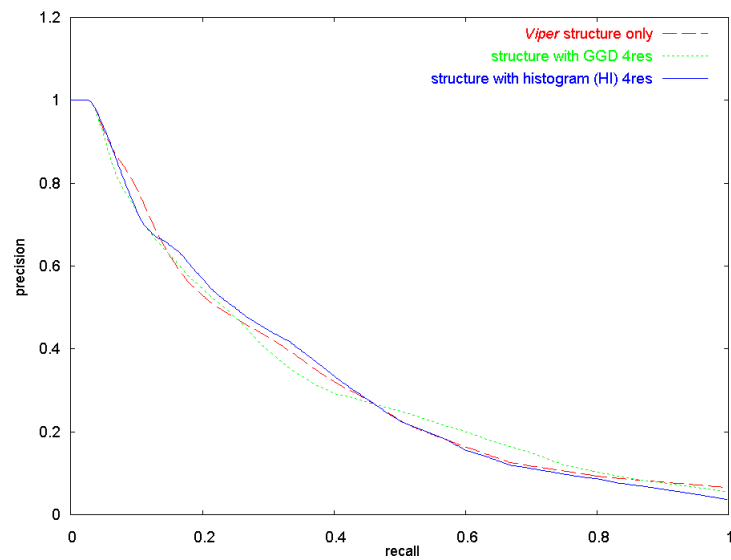


Figure 6.3: Structure query compared to *Viper*

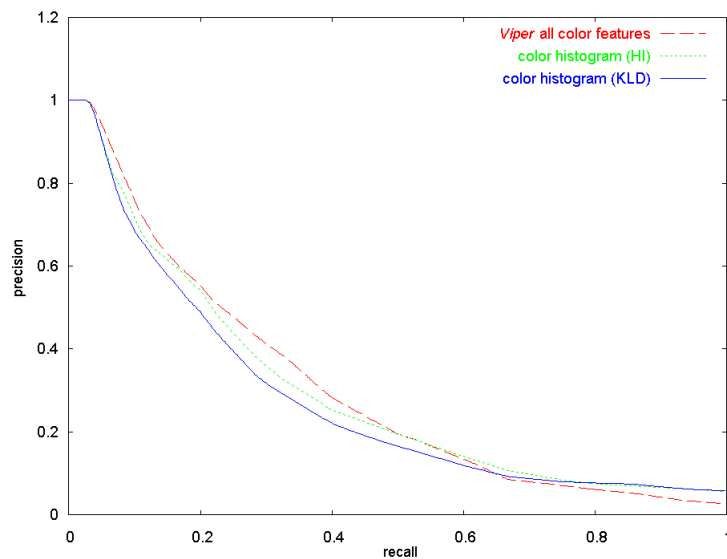


Figure 6.4: Color histogram compared with KLD and HI

6.4.2 Evaluation of color retrieval

In order to measure color similarity, color histograms were compared. As already discovered when examining texture retrieval performance, the retrieval rate using KLD compared to HI is worse. The explanation is the same as above: too many zero values might spoil the accuracy of the KLD. The corresponding plots are shown in figure 6.4.

The performance of the GIFT/*Viper* color features are also compared in figure 6.4. It can be seen that the GIFT color features perform better than the JPEG2000 color features up to a recall of 55%. This is mainly due to the application of color blocks. When only comparing the *Viper* color histogram features to the JPEG2000 color histogram features the results do not differ that much. It might be beneficial to also introduce color block features to the JPEG2000 feature set.

The influence of the color space is examined in figure 6.5, where retrieval performance of color histogram features based on the YD_bD_r color model is compared to *HSV* features. The experiment confirmed the results observed with the benchmark collection earlier: similarity is judged better in the *HSV* space, but the difference is not outstanding. Since the creation of *HSV* features needs additional processing time, YD_bD_r features are used further.

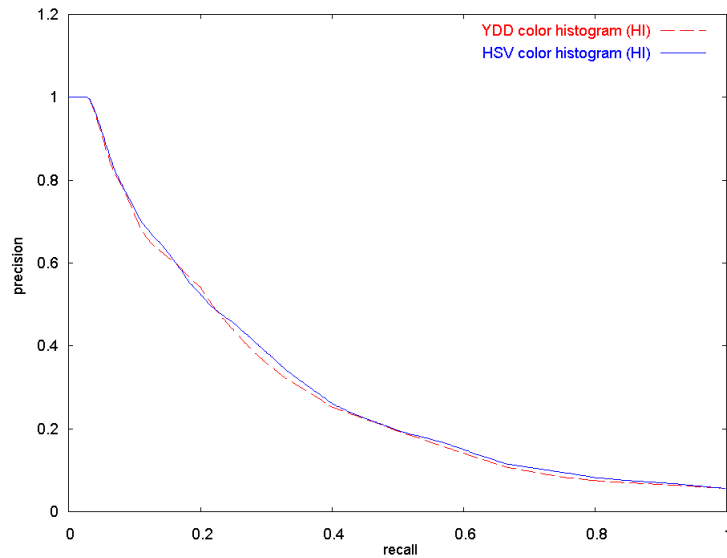


Figure 6.5: HSV color query compared to $YCbCr$ color query

6.4.3 Evaluation of the compound approach

Having investigated the influence of color and texture retrieval performance separately, both types were now evaluated together. The performance achieved here is most important, since this is what the user will mainly use in order to query for images.

An observation made by the *Viper* team could be reproduced: retrieval performance is improved when the different feature groups (here color and texture) are compared and normalized separately. This means each feature group is compared by a specialized query engine producing a result set which is then normalized to the range of $[0;1]$. The final score is obtained by merging the individual results and normalizing again. This procedure was called “separate normalization” by the members of the group (see [44] p. 67 for further explanation). Figure 6.6 shows the effect. The structure is modelled using subband histograms, color with a color histogram. For both types HI is taken as similarity measure. When evaluating the features together (i.e. the scores are simply summarized for each image) the retrieval performance at all recall rates is worse than when using separate normalization.

6.4. EVALUATION

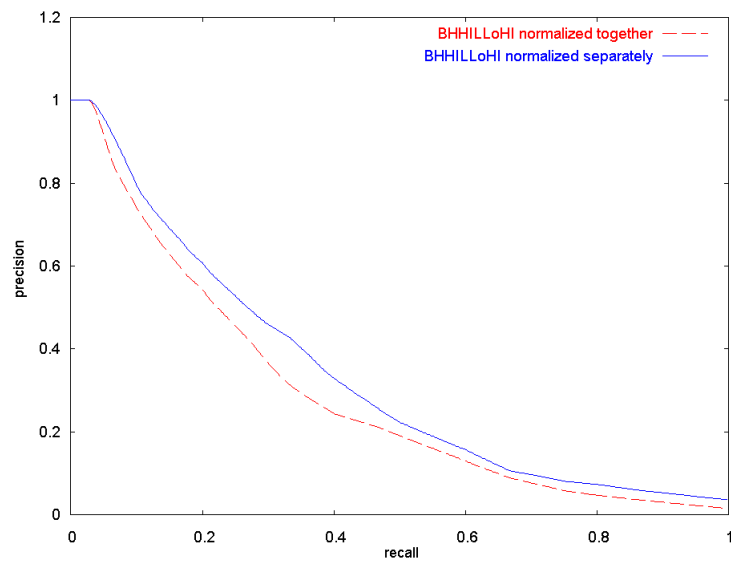


Figure 6.6: Features normalized together and separately

The default case for the JPEG2000 retrieval method is to process the different types of features individually. The evaluation method just summing the scores is only used here to show the effect.

Figure 6.7 shows the retrieval performance of different JPEG2000 feature sets in comparison to the default GIFT query engine (without feedback). It shows that the method using GGDs for texture and color histograms with HI for color performs best, even better as the default GIFT algorithm. It is only outperformed by the other methods at very small recall rates (up to a recall rate of 0.15). Since this method is also the fastest one developed and also the one using the least features, it will be the default JPEG2000 query method in the GIFT JPEG2000 plug-in.

6.4. EVALUATION

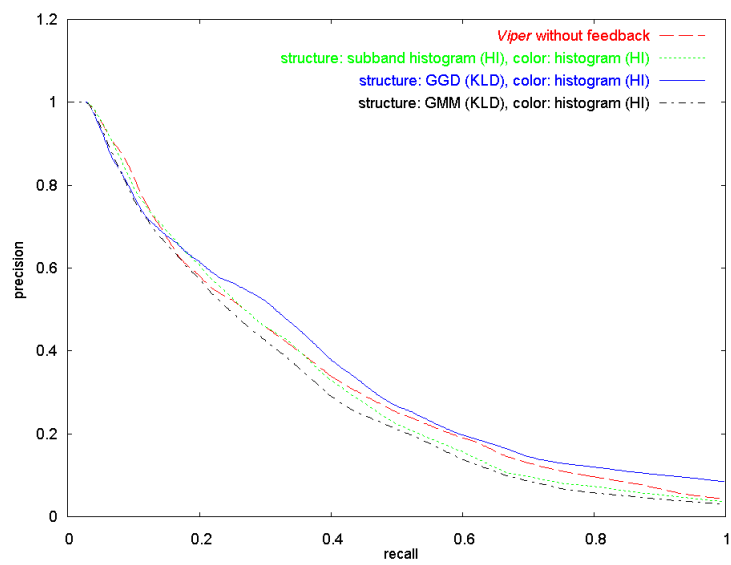


Figure 6.7: Color and texture features evaluated together.

Chapter 7

Conclusion

7.1 Summary

In this thesis, the development process of JPEG2000 retrieval methods beginning with obtaining the data to extensively testing the results has been shown.

In order to be able to make well founded decisions, fundamentals of CBIR and JPEG2000 were addressed in chapter 2 and 3.

Different JPEG2000 codec implementations were evaluated and the `jasPer` transcoder chosen to be modified to obtain wavelet coefficients. The `wcextract` application program was implemented and its functionality described in section 3.4.

In chapter 4 the core issue of this thesis is described: the development of retrieval methods using JPEG2000 wavelet coefficients. Existing methods were reviewed, and own solutions were researched. A test framework (`JP2FeatureFinder`) especially tailored for wavelet based data was designed to quickly develop and test feature sets.

The wavelet coefficient distribution of corresponding subbands was compared. It was most successful to model color and texture independently and to give the user the freedom to emphasize either. The complexity problem of directly comparing histograms was addressed by using Generalized Gaussian Densities. Although retrieval speed was improved tremendously, it showed that certain wavelet coef-

efficient distributions could not be modelled accurately by GGDs. An alternative approach was searched instead and found in Gaussian Mixture Models. They are capable of capturing the probability density better. Due to the lack of a fast method of calculating the KLD on Mixture Models however, this was only an advantage from a quality point of view. Retrieval speed was not sufficient.

The different retrieval methods developed were then incorporated in the GIFT framework by creating a JP2 plug-in. The functionality and the creation of the plug-in are described in chapter 5.

Using the GIFT JP2 plug-in, the retrieval methods developed were evaluated. For this purpose, the TSR2500 image collection consisting of images from a Swiss TV channel (TSR) with manually created ground truth data was used. The evaluation process is described in detail in chapter 6. It turned out that the retrieval method performing best on this data is the compound query method using GGDs for texture features and a color histogram of subbands LL_0 with HI.

7.2 Remaining issues and future work

Although good results have been obtained, there still exist a number of remaining issues. They could not be addressed due to the time constraints a thesis has. Especially the following topics which emerged during this work could either not be treated accurately or not at all. Future work is possible on any of these topics:

- A method has to be found to compare JPEG2000 images with arbitrary coding options. At present, only images with the same number of resolution levels and without tiling can be processed. Tests were only performed with images that were coded reversibly.

Moreover, sophisticated image retrieval systems should be able to compare JPEG2000 images to images in any other format found in the database. This is especially important if one thinks about an image search engine for the internet.

- In order to be able to give relevance feedback to a CBIRS, combining different JPEG2000 images to a joint query would be beneficial. At the moment a query can only consist of a single image.

7.2. REMAINING ISSUES AND FUTURE WORK

- A better solution for accurately modelling the wavelet coefficient distribution and achieving high retrieval speed might be found. Currently, using GGDs seems to be the best trade-off for both requirements. When using GMMs, a faster similarity measure than the Monte Carlo approximation of the KLD has to be found.
- The topic of storing and accessing the JPEG2000 feature data in a database was completely omitted. For real-world systems fast access to the data is crucial.
- As described in section 4.5, retrieval of similar images with different quality is difficult. It seems interesting to explore whether retrieval performance can be improved.
- In section 4.4.3 it was shown that the α parameter of the GGD is Zipf distributed. Exploiting this fact for improving retrieval performance and speed seems worth a try.

Declaration

“I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it contains no material from other persons or sources except where due acknowledgment has been made in the text. This thesis was not subject to any other submission yet.”

Alexandra Teynor

Abbreviations

API	Application Programming Interface
CBIR	Content Based Image Retrieval
CBIRS	Content Based Image Retrieval System
CDI	Compressed Domain Indexing
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
CRF	Canon Research Centre France
DTD	Document Type Declaration
DWT	Discrete Wavelet Transform
EBCOT	Embedded Block Coding with Optimal Truncation
EM	Expectation-Maximization
EOC	End Of Code stream marker segment
GGD	Generalized Gaussian Density
GIFT	GNU Image Finding Tool
GMM	Gaussian Mixture Model
GPL	General Public License
GSL	GNU Scientific Library
GUI	Graphical User Interface
HMM	Hidden Markow Model
HSV	Hue Saturation Value
HVS	Human Visual System
IJG	Independent JPEG Group
IR	Information Retrieval
JPEG	Joint Photographic Experts Group
KLD	Kullback Leibler Divergence
KLT	Karhunen-Loeve Transform
MRML	Multimedia Markup Language
PDF	Probability Density Function

QMF	Quadrature Mirror Filters
RGB	Red Green Blue
ROI	Region of Interest Coding
SPIHT	Set Partitioning In Hierarchical Trees
SOD	Start Of Data marker segment
SIZ	Size marker segment
TSR	Télévision Suisse Romande
VQ	Vector Quantization
XML	Extensible Markup Language

Bibliography

- [1] AAT - art and architecture thesaurus. <http://www.getty.edu/research/tools/vocabulary/aat/>.
- [2] The benchathlon network. <http://www.benchathlon.org>.
- [3] CNES glossary of terms - ground truth. <http://ceos.cnes.fr:8100/cdrom-98/ceos1/science/glossary/glossg.htm>.
- [4] Collections of the viper group University of Geneva. <http://viper.unige.ch/demo/localCollections.html>.
- [5] COLT scientific library. <http://hoschek.home.cern.ch/hoschek/colt/>.
- [6] Course notes on color theory. <http://www.cs.sfu.ca/CourseCentral/365/li/material/notes/Chap3/Chap3.3/Chap3.3.html>.
- [7] Distance measures. <http://viror.wiwi.uni-karlsruhe.de/webmining/script/6/Distanzmasse-1.xml>.
- [8] Eric Weissteins world of mathematics. <http://mathworld.wolfram.com/>.
- [9] GIFT - GNU Image Finding Tool. <http://www.gnu.org/software/gift/gift.html>.
- [10] GSL - GNU Scientific Library. <http://sources.redhat.com/gsl/>.
- [11] Home page of the *Viper* group of the university of geneva. <http://viper.unige.ch>.
- [12] IJG - Independent JPEG Group. <http://www.ijg.org>.

BIBLIOGRAPHY

- [13] Image database of the Univeristy of Washington. <http://www.cs.washington.edu/research/imagetatabase/groundtruth/>.
- [14] The JasPer project home page. <http://www.ece.uvic.ca/~mdadams/jasper>.
- [15] The JJ2000 image de/encoder. <http://jj2000.epfl.ch>.
- [16] JPEG2000 site of the JPEG. <http://www.jpeg.org/JPEG2000.html>.
- [17] The JPEG2000 standard. <http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=27687\&ICS1=35\&ICS2=40\&ICS3=>.
- [18] The kakadu image de/encoder. <http://www.kakadusoftware.com/>.
- [19] List of CBIRS. http://viper.unige.ch/other_systems.
- [20] MRML - Multimedia Retrieval Markup Language. <http://www.mrml.net>.
- [21] Newton method. <http://math.usask.ca/maclean/110/00/Printables/BW/Newton.pdf>.
- [22] OpenGL. <http://www.opengl.org>.
- [23] Ressource page about the Zipf distribution. <http://linkage.rockefeller.edu/wli/zipf/>.
- [24] Smoothed histograms. <http://www.ai.univie.ac.at/~elias/sdh/>.
- [25] Michael D. Adams. *The JPEG-2000 Still Image Compression Standard*. ISO/IEC JTC 1/SC 29/WG 1, N2412 edition, September 2001. Available: <http://www.ece.uvic.ca/~mdadams/papers/jpeg2000.pdf>.
- [26] Michael D. Adams. *JasPer Software Reference Manual (Version 1.600.0)*. ISO/IEC JTC 1/SC 29/WG 1, N2415 edition, October 2002. Available: <http://www.ece.uvic.ca/~mdadams/jasper/jasper.pdf>.
- [27] Samy Bengio. An introduction to statistical machine learning - EM for GMMs. Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP). Available: http://www.idiap.ch/~bengio/lectures/tex_gmm.pdf.
- [28] Jeff A. Blimes. A gentle tutorial on the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report ICSI-TR-97-021, University of Berkeley, 1997. Available: <http://citeseer.nj.nec.com/bilmes98gentle.html>.

BIBLIOGRAPHY

- [29] Vidacovic Brani and Peter Müller. Wavelets for kids - a tutorial introduction. Duke University. Available: <http://ftp.isds.duke.edu/WorkingPapers/94-13-1.ps>.
- [30] A. Cohen, Mantegna R. N., and Havlin S. Can Zipf analyses and entropy distinguish between artificial and natural language texts?, 1996. Available: <http://citeseer.nj.nec.com/212418.html>.
- [31] Lukasz Debowski. Zipf's law: What and why? Institute of Computer Science, Polish Academy of Sciences, December 2000. Available: <http://www.ipipan.waw.pl/~ldebowsk/manuskrypty/zipfwhy.pdf>.
- [32] Ronald A. DeVore and Bradley J. Lucier. Wavelets, 1992. Available: <http://www.math.purdue.edu/~lucier/692/wavelet.pdf>.
- [33] Minh N. Do. Fast Approximation of Kullback Leibler Distance for Dependence Trees and Hidden Markov Models. submitted to IEEE Signal Processing Letters. Available: http://www.ifp.uiuc.edu/~minhdo/publications/KLD_HMM.pdf.
- [34] Minh N. Do and Martin Vetterli. Wavelet-Based Texture Retrieval Using Generalized Gaussian Density and Kullback-Leibler Distance. In *IEEE Transactions on Image Processing*, volume 11, pages 146–157, February 2002. Available: <http://www.ifp.uiuc.edu/~minhdo/publications/WaveStat.pdf>.
- [35] John P. Eakins and Margaret E. Graham. Content-based image retrieval. Technical report, Institute for Image Data Research, University of Northumbria at Newcastle, 10 1999. Available: <http://www.jtap.ac.uk/reports/htm/jtap-039.html>.
- [36] Michael J. Gormish. Gormish notes on JPEG2000, September 2002. Available: <http://www.crc.ricoh.com/~gormish/jpeg2000.html>.
- [37] M. K. Mandal, T. Aboulnasr, and S. Panchanathan. Image indexing using moments and wavelets. In *IEEE Transactions on Consumer Electronics*, volume 42, pages 557–565, August 1996. Available: <http://citeseer.nj.nec.com/mandal96image.html>.
- [38] M. K. Mandal, F. Idris, and S. Panchanathan. Image and video indexing in the compressed domain: a critical review. Available: <http://citeseer.nj.nec.com/49471.html>.

BIBLIOGRAPHY

- [39] B.S. Manjunath and Wei-Ying Ma. Texture features for image retrieval. In Vittorio Castelli and Lawrence D. Bergmann, editors, *Image Databases - Search and Retrieval of Digital Imagery*, chapter 12, pages 313–344. John Wiley and Sons, Inc., 2002.
- [40] T. Minka and R. Picard. Interactive learning using a ‘society of models’. In *Proceeding of IEEE Conference on Computer Vision and Pattern Recognition (CVPR-1996)*, pages 447–452, 1996. Available: <http://citeseer.nj.nec.com/minka96interactive.html>.
- [41] Henning Müller. Jäger des verlorenen Fotos. *c’t*, (6):252–257, March 2002.
- [42] Henning Müller. Suchen ohne Worte. *c’t*, (15):162–167, July 2002.
- [43] Henning Müller, Wolfgang Müller, David McG. Squire, Marchand-Maillet Stéphane, and Thierry Pun. Performance evaluation in content-based image retrieval: Overview and proposals. submitted to Elsevier preprint, August 2000. Available: <http://citeseer.nj.nec.com/333946.html>.
- [44] Wolfgang Müller. *Design and implementation of a flexible Content Based Image Retrieval framework - The GNU Image Finding Tool*. PhD thesis, University of Geneva, Geneva, 2001. Thesis No. 3287.
- [45] Wolfgang Müller. Bildersuchbaukasten. *c’t*, (17):190–195, August 2002.
- [46] Latha Pillai. *Color Space Converter - Application Note: Virtex-II Family*. Xilinx Inc., June 2001. Available: <http://www.xilinx.com/xapp/xapp283.pdf>.
- [47] J. Puzicha, Y. Rubner, C. Tomasi, and J. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. In *Proceedings the IEEE International Conference on Computer Vision (ICCV-1999)*, pages 1165–1173, 1999. Available: <http://citeseer.nj.nec.com/puzicha99empirical.html>.
- [48] Simone Santini. *Exploratory Image Databases*. Academic Press, 2001.
- [49] Ben Schouten. *Giving eyes to ICT! or How does a computer recognize a cow?* PhD thesis, University of Amsterdam, March 2001.
- [50] John R. Smith. Color for image retrieval. In Vittorio Castelli and Lawrence D. Bergmann, editors, *Image Databases - Search and Retrieval of Digital Imagery*, chapter 11, pages 285–312. John Wiley and Sons, Inc., 2002.

BIBLIOGRAPHY

- [51] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, part 1". *IEEE Computer Graphics and Applications*, 15(3):76–84, May 1995. Available: <http://citeseer.nj.nec.com/34100.html>.
- [52] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, part 2". *IEEE Computer Graphics and Applications*, 15(4):75–85, July 1995. Available: <http://citeseer.nj.nec.com/stollnitz95wavelets.html>.
- [53] David S. Taubmann and Michael W. Marcellin. *JPEG2000 - Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, 2002.
- [54] Nuno Vasconcelos. *Bayesian Models for Visual Information Retrieval*. PhD thesis, Massachusetts Institute of Technology, June 2000.
- [55] Remo c. Veltcamp, Mirela Tanase, and Danielle Sent. Features in content-based image retrieval systems: A survey. In Remo C. Veltkamp, Hans Burkhardt, and Hans-Peter Kriegel, editors, *State-of-the-Art in Content-Based Image and Video Retrieval*, chapter 5, pages 97–124. Kluwer Academic Publishers, Utrecht University, Department of Computer Science, Utrecht, The Netherlands, 2002.
- [56] Remo C. Veltkamp, Hans Burkhardt, and Hans-Peter Kriegel, editors. *State-of-the-Art in Content-Based Image and Video Retrieval*. Kluwer Academic Publishers, 2001.
- [57] Colin C. Venters and Matthew Cooper. A review of content based image retrieval systems. Technical report, Institute for Image Data Research, University of Northumbria at Newcastle, 10 2000. Available: <http://www.jtap.ac.uk/reports/htm/jtap-054.html>.
- [58] Ziyou Xiong and Thomas S. Huang. Subband-based, memory-efficient JPEG2000 images indexing in compressed-domain. *Fifth IEEE Symposium on Image Analysis and Interpretation (SSIAI'02)*, 2002.

Index

- CBIR systems, 12
- Color, 15, 54
- Complexity, 61
- Compressed domain, 16

- Distance, 19

- Evaluation results, 96–102

- Feature, 12, 48
 - color, 15
 - other, 16
 - representation, 18
 - texture, 16
- Feature set
 - default GIFT, 95
 - experiments, 53
 - JP2 plug-in, 96
- Filter bank, 29

- Gaussian Mixture Model, 70
- Generalized Gaussian Density, 61
 - parameter distribution, 63
- GIFT, 83
 - configuration, 84

- Histogram, 18–19
 - definition, 18
 - disadvantages, 18
 - intersection, 21

- Image database, 92
- Image quality, 76
- Image retrieval
 - definition, 10
 - in pixel domain, 15
 - in the compressed domain, 16
 - levels of, 10
 - techniques, 15

- JasPer, 38

- JJ2000, 37
- JP2 plug-in, 87
- JP2FeatureFinder, 48
- JPEG2000, 23–37
 - codec, 25
 - properties, 24
 - wavelets in, 33

- Kakadu, 38
- Kullback-Leibler divergence
 - for GGDs, 63
 - for GMMs, 74
 - general definition, 21

- L_1 norm, 20

- Minkowski form distance, 20
- Multi-resolution analysis, 28

- Normalizer, 49

- P/R graph, 95
- Performance measure, 94
- Pixel domain, 15
- Precision, 94

- Quadrature Mirror Filter, 29
- Query
 - by example, 14
 - by region, 14
 - by sketch, 14
 - submission, 14
- Query engine, 49

- Recall, 94
- Relevance, 93
- Relevance feedback, 13

- Score board, 49
- Semantic gap, 11
- Similarity measure, 19

INDEX

Texture, 16, 54

Wavelet transform

 2D, 30

 discrete, 30

Wavelets, 27–34

 coefficient distribution, 33

 fundamentals, 27

wcextract, 40–45

Weighting function, 49

Zipf, 63

Appendix A

Mathematical Glossary

In this appendix mathematical concepts used in this thesis are briefly reviewed. The material cites [8] if not otherwise stated. If terms in this glossary are printed italic, more information on these topics can be found within the glossary.

- Bin** An interval into which a given data point does or does not fall.
- Compact set** The *set* S is compact if, from any sequence of elements X_1, X_2, \dots of S , a subsequence can always be extracted which tends to some limit element X of S . Compact sets are therefore sets which are both closed and bounded.
- Compact support** A function has compact *support* if it is zero outside of a *compact set*. A function with compact support is only interesting in a bounded domain. Alternatively, one can say that a function has compact support if its support is a compact set.
- Covariance** Given n sets of *variates* denoted $\{X_1\}, \dots, \{X_n\}$ the covariance $\sigma_{ij} \equiv cov(x_i, x_j)$ of x_i and x_j is defined by

$$\begin{aligned} cov(x_i, x_j) &\equiv \langle (x_i - \mu_i)(x_j - \mu_j) \rangle \\ &= \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle \end{aligned}$$

where $\mu_i = \langle x_i \rangle$ and $\mu_j = \langle x_j \rangle$ are the *means* of x_i and x_j , respectively. The matrix V_{ij} of the quantities $V_{ij} = cov(x_i, x_j)$ is called covariance matrix. [...]

The covariance of two variates X_i and X_j provides a measure of how strongly correlated these variables are [...].

Distribution function

The distribution function $D(x)$ (also called the cumulative density function (CDF) or probability distribution function), describes the probability that a *variate* X takes on a value less than or equal to a number x . [...]

The distribution function is therefore related to a continuous *probability function* $P(x)$ by

$$D(x) = P(X \leq x) \equiv \int_{x_{min}}^x P(x') dx',$$

so $P(x)$ (when it exists) is simply the derivative of the distribution function.

Estimate

An estimate is an educated guess for an unknown quantity or outcome based on known information. The making of estimates is an important part of statistics, since care is needed to provide as accurate an estimate as possible using as little input data as possible. Often, an estimate for the uncertainty ΔE of an estimate E can also be determined statistically. A rule that tells how to calculate an estimate based on the measurements contained in a sample is called an *estimator*.

Estimator

An estimator is a rule that tells how to calculate an estimate based on the measurements contained in a sample.

Expectation value

The expectation value of a function $f(x)$ in a variable x is denoted $\langle f(x) \rangle$ or $Ef(x)$. For a single discrete variable, it is defined by

$$\langle f(x) \rangle = \sum_x f(x)P(x)$$

For a single continuous variable it is defined by

$$\langle f(x) \rangle = \int f(x)P(x)dx$$

The expectation value satisfies

$$\langle ax + by \rangle = a\langle x \rangle + b\langle y \rangle$$

$$\langle a \rangle = a$$

$$\langle \sum x \rangle = \sum \langle x \rangle$$

Hilbert space

A Hilbert space is a vector space H with an inner product $\langle f, g \rangle$ such that the *norm* defined by

$$|f| = \sqrt{\langle f, f \rangle}$$

turns H into a complete metric space. If the inner product does not so define a norm, it is instead known as an inner product space.

Examples of finite-dimensional Hilbert spaces include

1. The real numbers \mathbb{R}^n with $\langle v, u \rangle$ the vector dot product of v and u .
2. The complex numbers \mathbb{C}^n with $\langle v, u \rangle$ the vector dot product of v and the complex conjugate of u .

An example of an infinite-dimensional Hilbert space is L^2 , the *set* of all functions $f: \mathbb{R} \rightarrow \mathbb{R}$ such that the integral of f^2 over the whole *real line* is finite. In this case, the inner product is

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x)g(x)dx$$

Likelihood

The hypothetical probability that an event which has already occurred would yield a specific outcome. The concept differs from that of a probability in that a probability refers to the occurrence of future events, while a likelihood refers to past events with known outcomes.

Linearization

The following explanation cites [21].

If a function f is differentiable at a its linearization L is the linear function

$$L(x) = f(a) + f'(a)(x - a)$$

The graph of the linearization is just the tangent line to $y = f(x)$ at $(a, f(a))$, and it has the properties $L(a) = f(a)$ and $L'(a) = f'(a)$. This tangent line crosses the x -axis at $a - \frac{f(a)}{f'(a)}$ [...].

Maximum likelihood

The procedure of finding the value of one or more parameters for a given statistic which makes the known *likelihood* distribution a maximum. The maximum likelihood estimate for a parameter μ is denoted $\hat{\mu}$.

Mean

The quantity commonly referred to as “the” mean is the arithmetic mean, also called the average. The mean (or, more specifically, the population mean) of a *probability function* $P(x)$ is the first *raw moment* μ'_1 , defined by

$$\mu \equiv \langle xP(x) \rangle$$

where $\langle f \rangle$ is the expectation value. For a continuous distribution, this can be written

$$\mu = \int xP(x)dx$$

where the integral is taken over the domain of $P(x)$, and for a discrete distribution, is given by the sum

$$\mu = \sum_i x_i P(x_i)$$

[...] The sample mean is the mean of a set $\{x_1, \dots, x_n\}$ of n observations from a given distribution,

$$m \equiv \frac{1}{n} \sum_{k=1}^n x_k$$

and is an unbiased estimator for the population mean μ .

Moment

The n^{th} raw moment μ'_n (i.e., moment about zero) of a distribution $P(x)$ is defined by

$$\mu'_n = \langle x^n \rangle,$$

where

$$\langle f(x) \rangle = \begin{cases} \sum f(x)P(x) & \text{discrete distribution} \\ \int f(x)P(x)dx & \text{continuous distribution} \end{cases}$$

μ'_1 , the *mean*, is usually simply denoted $\mu = \mu_1$. If the moment is instead taken about a point a ,

$$\mu_n(a) = \langle (x-a)^n \rangle = \sum (x-a)^n P(x)$$

The moments are most commonly taken about the mean. These so-called central moments are denoted μ_n and are defined by

$$\begin{aligned} \mu_n &\equiv \langle (x-\mu)^n \rangle, \\ &= \int (x-\mu)^n P(x)dx, \end{aligned}$$

with $\mu_1 = 0$. The second moment about the mean is equal to the *variance*

$$\mu_2 = \sigma^2,$$

where $\sigma = \sqrt{\mu_2}$ is called the standard deviation.

Newton-Raphson method

The Newton method (also called Newton-Raphson method) is an iterative *root* finding algorithm.

The following explanation is taken from [21]. The basic idea of the Newton method to find the roots of a function f is to pick a value x_0 which is suspected to be close to a root of f and then to compute the value x_1 where the tangent line to $y = f(x)$ at $(x_0, f(x_0))$ crosses the x -axis. For this purpose the equation of *linearization* is used:

$$y = f(x_0) + f'(x_0)(x - x_0)$$

It crosses the x -axis when $y=0$ and $x = x_1$:

$$0 = f(x_0) + f'(x_0)(x_1 - x_0)$$

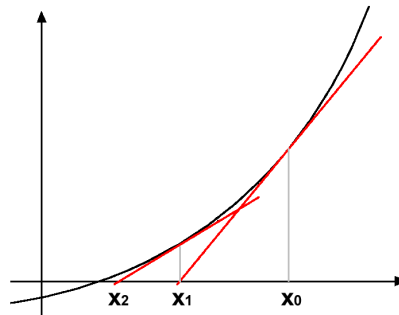
and can be solved for x_1 :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

The value of x_1 should be closer to the root than x_0 . The procedure is repeated recursively to find ever better estimates for the root:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Following figure illustrates the procedure:

**Norm**

The norm of a mathematical object is a quantity that in some (possibly abstract) sense describes the length, size, or extent of the object. Norms exist for complex numbers (the complex modulus, sometimes also called the complex norm or simply "the norm"), quaternions (quaternion norm), vectors (vector norms), and matrices (matrix norms). [...]

The term "norm" is often used without additional qualification to refer to a particular type of norm, most commonly the flavor of vector norm technically known as the L2-norm. This norm is variously denoted $\|x\|_2$ or $|x|$ and gives the length of an n -Vector (x_1, x_2, \dots, x_n) . It can be computed as

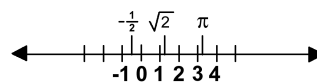
$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

Probability density function (pdf)

The probability function $P(x)$ (also called the probability density function (PDF) or density function) of a continuous distribution is defined as the derivative of the (cumulative) *distribution function* $D(x)$ [...].

Real line

A line with a fixed scale so that every real number corresponds to a unique point on the line. The generalization of the real line to two dimensions is called the complex plane.



Roots

The roots (sometimes also called "zeros") of an equation $f(x) = 0$ are the values of x for which the equation is satisfied.

Set

A set is a finite or infinite collection of objects in which order has no significance, and multiplicity is generally also ignored (unlike a list or multiset).

Set closure

A set S and a binary operator $*$ are said to exhibit closure if applying the binary operator to two elements [of] S returns a value which is itself a member of S .

Statistical distribution

The distribution of a variable is a description of the relative numbers of times each possible outcome will occur in a number of trials. The function describing the distribution is called the *probability function*, and the function describing the cumulative probability that a given value or any value smaller than it will occur is called the *distribution function*.

Support

The *set closure* of the set of arguments of a function f for which f is not zero.

Variate

A variate is a generalization of the concept of a random variable that is defined without reference to a particular type of probabilistic experiment. It is defined as the set of all random variables that obey a given probabilistic law.

Appendix B

JP2FeatureFinder

On the following two pages, the class diagram for the JP2FeatureFinder is displayed. The purpose of each class or class group is explained in section 4.2.

